VAX DATATRIEVE
User's Guide

Order No. AA-K080D-TE

December 1985

This manual is a guide to the interactive use of VAX DATATRIEVE. It describes how to use DATATRIEVE to manipulate data and its use with forms and database management products. It also includes information on improving performance and working with remote data.

OPERATING SYSTEM: **VMS**
MicroVMS

SOFTWARE VERSION: **VAX DATATRIEVE V3**

digital equipment corporation, maynard, massachusetts

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by DIGITAL or its affiliated companies.

The postage-paid READER'S COMMENTS form on the last page of this document requests your critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

ACMS	DECUS	UNIBUS
CDD	MicroVAX	VAX
DATATRIEVE	MicroVMS	VAXcluster
DEC	PDP	VAX Information Architecture
DECgraph	Rdb/ELN	VMS
DECnet	Rdb/VMS	VT
DECslide	TDMS	

digital™

42306

Contents

Part 2 Manipulating Data

2 Writing Record Selection Expressions

3 Entering New Data

4 Modifying Data

Using View Domains

Using Hierarchies

Part 3 Programming in DATATRIEVE

Using DATATRIEVE Procedures

Part 4 Optimizing DATATRIEVE

0 Restructuring Data

1 Designing Better Records

2 Improving DATATRIEVE Performance

4 Using DATATRIEVE with DBMS

15 Using DATATRIEVE with Rdb

6 Accessing Remote Data

Name Recognition and Single Record Context

Sample Database Definitions and Procedures

Index

Examples

Figures

Tables

How to Use This Manual

is manual explains the concepts and terminology of the VAX DATATRIEVE
tware, also referred to as DATATRIEVE in this manual. It discusses how to
ine domains, records, tables, and procedures and how to catalog them in the
X Common Data Dictionary, also referred to simply as CDD. It describes var-
s ways of managing data stored in RMS files, VAX Rdb/VMS (also referred to
Rdb), and VAX DBMS (also referred to as DBMS) databases and how to
rieve information from them.

tended Audience

is manual is intended for people who either:

- Have read or done the examples in the *VAX DATATRIEVE Handbook*
- Have experience using DATATRIEVE-11
- Have experience in applications programming

ou have no prior experience with DATATRIEVE, the *VAX DATATRIEVE*
ndbook provides information on the basic tasks of managing information with
TATRIEVE and can help you get started with DATATRIEVE applications.

erating System Information

verify which versions of your operating system are compatible with this ver-
1 of VAX DATATRIEVE, check the most recent copy of the following:

- For the VMS operating system -- *VAX/VMS Optional Software Cross Reference Table, SPD 25.99.xx*
- For the MicroVMS operating system -- *MicroVMS Optional Software Cross Reference Table, SPD 28.99.xx*

Structure

This manual is divided into five major parts, two appendixes, and an index:

Part 1	Understanding DATATRIEVE Explains basic terminology and concepts of VAX DATATRIEVE (Chapter 1).
Part 2	Manipulating Data Describes how to write record selection expressions, store and modify data, and use view domains and hierarchies (Chapters 2 through 6).
Part 3	Programming in DATATRIEVE Illustrates programming in DATATRIEVE through use of procedures, command files, and variables (Chapters 7 through 9).
Part 4	Optimizing DATATRIEVE Explains how to change record and file definitions, design the most efficient records, and improve performance of DATATRIEVE applications (Chapters 10 through 12).
Part 5	DATATRIEVE and the VAX Information Architecture Explains how to use TDMS and FMS forms in a DATATRIEV application, how to access data in DBMS and in Rdb databases and how to access remote data (Chapters 13 through 16).

Appendix A presents a detailed discussion of DATATRIEVE context; Appendix lists the sample data definitions created during the DATATRIEVE installation procedure.

Related Manuals

For other information on the topics covered in this book, see:

VAX DATATRIEVE Handbook

VAX DATATRIEVE Guide to Writing Reports

VAX DATATRIEVE Reference Manual

VAX DATATRIEVE Guide to Using Graphics

VAX DATATRIEVE Guide to Programming and Customizing

VAX Common Data Dictionary Utilities Reference Manual

onventions

nce CDD Version 3.1, CDD path names include a leading underscore. For
ample:

```
R> SHOW DICTIONARY
e default directory is _CDD$TOP.DTR32.WEAGER
```

amples of output in DATATRIEVE manuals do not reflect this change. You do
t need to enter CDD path names with the leading underscore.

mbols used in examples:

RET>	This symbol tells you to press the RETURN key on the keyboard of your terminal.
TAB>	This symbol tells you to press the TAB key on the keyboard of your terminal.
CTRL/x>	This symbol tells you to press the CTRL (control) key and a letter key (Z, C, or Y) at the same time. If you press CTRL/Z, the word Exit appears in reverse video; if you press CTRL/Y, the word Interrupt appears in reverse video. Examples of video output in this book do not include either word; instead the conventions ^Z and ^Y are used.
lor	Colored ink in examples shows user input. Unless otherwise indicated, you enter each input line by pressing the RETURN key. A vertical ellipsis in an example means that information not directly related to the example has been omitted.

mbols and conventions used in syntax formats:

PERCASE WORDS	Uppercase words are DATATRIEVE keywords. Enter them exactly as shown.
ercase words	Lowercase words are generic terms that indicate entries you must provide. Braces mean you must choose one, but no more than one, of the enclosed entries.

[ ]	Brackets mean you have the option of choosing one, but no more than one, of the enclosed entries.
...	A horizontal ellipsis means you have the option of repeating the preceding element of the syntax format.
. . .	A vertical ellipsis in a syntax format means you can repeat the syntax element on the preceding line.
" "	These are called double quotation marks.
' '	These are called single quotation marks.

Technical Changes and New Features

ıis section describes the new features for DATATRIEVE documented in this anual.

ɜrsion 3.3 of DATATRIEVE provides limited support for the Rdb segmented ring data type. This support lets you store and retrieve Rdb records that contain ɨlds with the segmented string data type.

ɜe Section 15.9 for more information on using segmented strings with ∀TATRIEVE.

ɜe the Release Notes for a description of all technical changes to ∀TATRIEVE.

Part 1
Understanding DATATRIEVE

Understanding DATATRIEVE 1

ıis chapter reviews basic DATATRIEVE concepts that were presented in hapter 1 of the *VAX DATATRIEVE Handbook* and explained in greater detail later chapters of the Handbook. If you have read the Handbook, you will find ost information in this chapter familiar to you. If you have some programming ıckground, this chapter provides basic information about DATATRIEVE and ıe range of functions it performs. In addition, there is a section that explains ɔtional software products you can use with DATATRIEVE that might be avail-ɔle on your system.

.1 Starting and Ending a DATATRIEVE Session

ɔ start a DATATRIEVE session, begin at DCL command level and use the com-and RUN SYS$SYSTEM:DTR32xx. The suffix xx is sometimes necessary to entify the image of DATATRIEVE you want to run. See the person who stalled DATATRIEVE on your system to determine if you need to specify a suf-ɛ and, if you do, what characters you type in place of xx. For example, if you ant to run the version of DATATRIEVE that uses TDMS, you may need to clude the suffix TD:

```
RUN SYS$SYSTEM:DTR32TD
.X Datatrieve V3
:C Query and Report System
pe HELP for help
R>
```

he startup banner shows that you have successfully invoked DATATRIEVE. ote that the message you receive in response to your request should specify the ɛrsion of VAX DATATRIEVE you are running. If it does not, consult the person sponsible for VAX DATATRIEVE at your site.

To simplify this command, you can assign a symbol to it in your LOGIN.COM file. To use DTR32 to invoke the version of DATATRIEVE in the previous example, enter:

```
$ DTR32 :==  $SYS$SYSTEM:DTR32TD
$ DTR32
VAX Datatrieve V3
DEC Query and Report System
Type HELP for help
DTR>
```

To end your DATATRIEVE session if you are at a DTR> prompt, type EXIT and press the RETURN key or use CTRL/Z. Either returns you to the VMS system prompt. If you are at a CON>, DFN>, or RW> prompt, use CTRL/Z to get to the DTR> prompt. Then use CTRL/Z again or type EXIT and press the RETURN key.

1.2 Writing a DATATRIEVE Session to a Log File

You can use the DATATRIEVE OPEN command to create a record of your DATATRIEVE session in a file in a VMS directory. At any point in the session you can issue the CLOSE command and save all of the session up to that point. If you do not end the creation of the log file with a CLOSE command, DATATRIEVE automatically closes the file when you EXIT from DATATRIEVE.

In this example the log file name is MONTHLY_RPT.LOG, but you can specify any file name you want:

```
DTR> OPEN MONTHLY_RPT.LOG
DTR> :MONTHLY_RPT
                .
                .
                .
DTR> CLOSE              ! Do not enter the file name following CLOSE.
DTR> EXIT
```

See the *VAX DATATRIEVE Reference Manual* for more information about the OPEN and CLOSE commands.

3 DATATRIEVE Concepts and Terminology

ıe following sections review these concepts and terms:

Databases

DATATRIEVE domains

Common Data Dictionary

Commands and statements

Procedures

Command files

DATATRIEVE view domains

Tables

Distributed data

3.1 Databases

ATATRIEVE can access three different types of databases:

File-structured databases that you set up with DATATRIEVE

Databases that you create using VAX Rdb

Databases that you create using VAX DBMS

‹amples in this book show you how to create your own file-structured databases. ıapters 14 and 15 explain how to access data stored in DBMS and Rdb ıtabases.

3.2 DATATRIEVE Domains

hen you manage information with DATATRIEVE, you access data through con-ructs called *domains*. A domain definition establishes a name for a set of data ıd tells DATATRIEVE where that data is described and where the data is ored. A domain definition contains the name of the domain, the name of a cord (data description), and the name of a data file.

the domain is a DATATRIEVE DBMS domain or DATATRIEVE Rdb domain, contains the name of a DBMS or Rdb database.

You create the domain definition, and the record definition and file that the domain uses. You can do this by invoking ADT (Application Design Tool), or you can create all three definitions using different forms of the DEFINE command. Chapter 1 of the *VAX DATATRIEVE Handbook* discusses using ADT to create a database. Chapters 10, 11, and 12 of the handbook discuss defining domains, records, and files without ADT.

If you want to use DATATRIEVE to access a database managed by VAX Rdb or VAX DBMS, you do not create record or file definitions because the database is not file-structured. The database has already been created using VAX Rdb or VAX DBMS. However, you may want to create a domain definition to let DATATRIEVE know how and where your data is stored. Chapter 14 and Chapter 15 describe DBMS and Rdb domains in greater detail.

After you have created a DATATRIEVE domain, you refer to its data by using the domain name in your DATATRIEVE commands and statements.

1.3.3 Common Data Dictionary

DATATRIEVE uses the VAX Common Data Dictionary (CDD) to store data definitions and procedures. The CDD is a VAX software product that you always have on your system if you are using DATATRIEVE.

In an information management system, reliable data definitions are as important as the data itself. The manager of the system or the person in charge of data administration must know how data is represented and how it is used by different applications running on the system. Shared data definitions must be unambiguous, and sensitive data definitions must be protected. The CDD assists in these tasks by providing a central storage place for data definitions and a data security system for their protection.

The CDD is actually a hierarchy of dictionaries. A dictionary is to the CDD what a directory is to the VMS operating system. Just as you enter a default directory when you log in to your VMS system, you enter a default dictionary when you start DATATRIEVE. As you can create new VMS directories and move from one to another, you can create new dictionaries and move among them using DATATRIEVE commands.

Remember, however, that dictionaries do not contain data files. They store only definitions and the security information connected with those definitions. Data files always reside in VMS directories.

3.4 Commands and Statements

ou use commands and statements to manage information with DATATRIEVE. ost of the commands deal with the CDD and perform the data description func- ons of DATATRIEVE. The following two commands, for instance, tell ATATRIEVE you want CDD$TOP.DTR$USERS.WARTON to be your current ctionary, and you plan to store records in a domain named PERSONNEL:

```
R> SET DICTIONARY CDD$TOP.DTR$USERS.WARTON
R> READY PERSONNEL WRITE
R>
```

her examples of commands are DEFINE DOMAIN, REDEFINE DOMAIN, DIT, DELETE, FINISH, and RELEASE.

atements, on the other hand, perform the query, report, and data manipulation nctions of DATATRIEVE. The two statements after the READY command in e following example store a record in PERSONNEL and then display that cord. In the store operation, DATATRIEVE prompts you to enter each field in e record:

```
R> READY PERSONNEL WRITE
R> STORE PERSONNEL
ter ID: 99039
ter EMPLOYEE_STATUS: TRAINEE
ter FIRST_NAME: MAYBEL
ter LAST_NAME: STREP
ter DEPT: T32
ter START_DATE: <TAB><RET>
ter SALARY: 20456
ter SUP_ID:  23456

R> PRINT PERSONNEL WITH ID = "99039"

                FIRST       LAST            START              SUP
D     STATUS    NAME        NAME    DEPT    DATE     SALARY    ID

039 TRAINEE     MAYBEL      STREP   T32              $20,456 23456

R>
```

her DATATRIEVE statements include FIND, MODIFY, DISPLAY, SELECT, d so on.

ou can combine statements into compound statements (BEGIN-END, THEN), mplex logical structures with loops (FOR, REPEAT, WHILE), and conditional

transfers (IF-THEN-ELSE, ABORT). The following statements retrieve and ther display records of all the employees in department T32:

```
DTR> FOR PERSONNEL WITH DEPT EQUAL "T32" PRINT

                    FIRST       LAST                START              SUP
 ID     STATUS      NAME        NAME      DEPT      DATE      SALARY   ID

38462 EXPERIENCED BILL        SWAY        T32    5-May-1980 $54,000 00012
48573 TRAINEE     SY          KELLER      T32    2-Aug-1981 $31,546 87289
83764 EXPERIENCED LES         WHART       T32    4-Apr-1980 $41,029 87289

DTR>
```

See the *VAX DATATRIEVE Reference Manual* for a description of all the DATATRIEVE commands and statements.

1.3.5 Procedures

Many applications of VAX DATATRIEVE involve sequences of commands and statements that recur frequently. You can avoid retyping such a sequence by sto ing it in the CDD as a procedure. With the DEFINE PROCEDURE command, you give the recurring sequence a name and enter both the name and the sequence into the CDD. You invoke the procedure by typing either a colon (:) or EXECUTE, followed by the procedure name. DATATRIEVE then executes the statements and commands in the procedure. Refer to Chapter 7 for a discussion of DATATRIEVE procedures.

1.3.6 DATATRIEVE Command Files

You can use DATATRIEVE command files in much the same way you use DATATRIEVE procedures. Procedures are stored in the Common Data Dictionary, and command files are stored in a VMS directory. Both contain only DATATRIEVE commands and statements. You invoke command files at the DTR> prompt by typing the at sign (@), followed by the command file specification.

1.3.7 DATATRIEVE View Domains

You might want to access data repeatedly that is in more than one domain or restrict someone's access to a subset of the information that is in a domain. You can create a special type of DATATRIEVE domain, a view domain, to help accon plish these functions. The definition of a view domain is stored in the CDD, and you can use a view domain in much the same way as you would a "simple" domain. Chapter 5 tells you how to create and use view domains.

3.8 DATATRIEVE Tables

ıother type of definition you can create with DATATRIEVE and store in the)D is a table definition. DATATRIEVE tables let you:

Specify one value and retrieve another associated with it

Validate data according to the presence or absence of a data item in the table

\TATRIEVE lets you define and use two types of tables: dictionary tables and main tables. See Chapter 12 of the *VAX DATATRIEVE Handbook* and the apter on designing better records in this manual for more information on fining and using tables.

3.9 DATATRIEVE Collections

\TATRIEVE collections are temporary groups of records that you pull together)m a larger set of data. The following example creates a collection named :PTT32 that contains all employee records from Department T32 and then iso- ;es the records of those employees earning more than $30,000:

```
R> FIND DEPTT32 IN PERSONNEL WITH DEPT EQ "T32"
 records found]
R> FIND WELLPAID IN DEPTT32 WITH SALARY GT 30000
 records found]
R> PRINT
 record selected, printing whole collection.

                FIRST       LAST                START              SUP
)    STATUS     NAME        NAME     DEPT       DATE      SALARY   ID

462 EXPERIENCED BILL        SWAY     T32    5-May-1980 $54,000 00012
573 TRAINEE     SY          KELLER   T32    2-Aug-1981 $31,546 87289
764 EXPERIENCED JIM         MEADER   T32    4-Apr-1980 $41,029 87289
```

ıe collections DEPTT32 and WELLPAID are available for use until you decide remove them or until you exit from DATATRIEVE. Collections are useful ıen you are learning DATATRIEVE or first thinking about an application. ;ing FIND statements parallels the way most of us think when retrieving data; at is, we use a series of steps to narrow down a group of records to just the ones want.

3.10 Distributed Data

ith VAX DATATRIEVE, you can easily access domains defined on other sys- ns that are linked to yours by DECnet. The other system must have VAX \TATRIEVE, DATATRIEVE-11, or DATATRIEVE-20 installed.

1.4 What DATATRIEVE Can Do for the Programmer

You do not have to have a programming background to use DATATRIEVE. However, if you do have programming experience, you probably want to know how DATATRIEVE is different from the languages you have used before.

DATATRIEVE is a *fourth-generation language.* Its syntax is more "English-like' than that of COBOL or BASIC, and it has a strong nonprocedural aspect. It executes commands as you type them, and you can often simply tell DATATRIEVE what information you want by name, instead of specifying how to obtain that information.

DATATRIEVE provides the same data storage capabilities that you have with other languages. It can store and retrieve data using existing RMS data files of any type. It can also create sequential and multikey indexed files. However, you cannot create a relative file with DATATRIEVE.

DATATRIEVE allows you to set up data hierarchies (as in a COBOL group item) and repeating fields (as in a COBOL OCCURS clause). Retrieving data from repeating fields (called *lists* in DATATRIEVE terminology) is not as easy as retrieving data from other types of fields. Be sure to take this fact into consideration before you decide to use DATATRIEVE's OCCURS clause.

In COBOL or BASIC, each program describes the structure of input and output records. DATATRIEVE lets you define records and store record definitions separately from a program. Then you can write any number of programs that use the records you have defined, without redefining the record each time.

DATATRIEVE also handles other common language functions automatically, without the need for language statements. For instance, DATATRIEVE:

- Finds data files, opens them, and performs input/output operations
- Labels columns in an output display
- Converts data types
- Formats data for output
- Handles conditions like end-of-file and matching

s a result, you can save many lines of code, get applications running quickly, and ıve code that is more readable than languages such as COBOL or BASIC.

sing the DATATRIEVE Call Interface, you can also include DATATRIEVE .nctions in a program written in another language. The Call Interface is used ost often in two ways:

You can use the linkage section of your program to do file access entirely through DATATRIEVE. In this way, the calling program does not need to specify the structure of the data, and you do not need to relink programs when the data files change.

You can write a program that passes commands and statements to DATATRIEVE. The program can present the user with a customized interface, such as a menu. In this way, you can "hide" DATATRIEVE from users who do not know how to use its commands and statements.

ıe *VAX DATATRIEVE Guide to Customizing and Programming* explains how ıu can use DATATRIEVE with other languages.

.5 The Sample Domains, Records, and Data Files

ıe VAX DATATRIEVE installation kit includes several sample domains: AMILIES, PERSONNEL, and YACHTS, among others. The domain definitions ıd the record definitions (FAMILY_REC, PERSONNEL_REC, and YACHT, for :ample) are stored in the Common Data Dictionary in the directory called DD$TOP.DTR$LIB.DEMO.

ıe data files, such as FAMILY.DAT, PERSON.DAT, and YACHT.DAT, are ored in the library directory DTR$LIBRARY.

you want to use these domains to follow the examples in the VAX ATATRIEVE documentation and to practice using DATATRIEVE commands ıd statements, you should copy the data files to your default VMS directory. ɔu can use the VMS COPY command to transfer the data files to your default MS directory. For example, to copy FAMILY.DAT, enter:

```
COPY DTR$LIBRARY:FAMILY.DAT    *
```

ı get access to the domain and record definitions stored in the Common Data ctionary, you can set your default dictionary directory to the DEMO directory. ter you invoke VAX DATATRIEVE, enter this command:

```
R> SET DICTIONARY CDD$TOP.DTR$LIB.DEMO
R>
```

To see that the sample domain and record definitions are in place, use the SHOV command:

```
DTR> SHOW DOMAINS, RECORDS
Domains:
    ANNUAL_REPORT;1    FAMILIES;1          KETCHES;2          OWNERS;1
    OWNERS_SEQUENTIAL;1                    PAYABLES;1         PERSONNEL;1
    PETS;1             PROJECTS;1          SAILBOATS;1        SALES;1
    YACHTS;1           YACHTS_SEQUENTIAL;1

Records:
    ANNUAL_REC;1       DAB;1               FAMILY_REC;1       OWNER_RECORD;1
    PAYABLES_REC;1     PERSONNEL_REC;1     PET_REC;1          PROJECT_REC;1
    SALES_REC;1        YACHT;1

DTR>
```

The results of this command vary from one system to another, but you should be sure that the domain and record definitions you need are listed among those that DATATRIEVE displays on your terminal.

If you cannot copy the data files from DTR$LIBRARY, cannot get access to the DEMO directory, or cannot find the domain and record definitions in the DEMO directory, see the person responsible for VAX DATATRIEVE on your system.

1.6 Using SET Commands to Control Output

When you invoke VAX DATATRIEVE, it sets several characteristics that contr your display of input and output. You can display these settings with the SHOW SET_UP command:

```
DTR> SHOW SET_UP
Set-up:
    Columns-page: 80
    No abort
    Prompt
    No search
    Form
    No verify
    No semicolon
    No lock wait
DTR>
```

The settings shown in the example are the default settings. You can change thes characteristics at any time during a DATATRIEVE session by using the forms c the SET command discussed in the following pages.

1.6.1 Changing the Columns-Page Setting

The default for the columns-page setting is 80 characters, the width of most vid display screens. You can change this setting to fit your application and terminal characteristics.

6.1.1 Increasing the Columns-Page Setting -- You may want to increase the olumns-page setting if you have a VT100-family or hardcopy terminal. Before ou can display lines more than 80 characters long, you must set your terminal to low the display of long lines. This is a two-step process:

1. Use the DCL SET TERMINAL command to tell your system to increase the width of lines it can send you:

```
$ SET TERMINAL/WIDTH=132
$
```

   Alternatively, within DATATRIEVE you can use the function:

```
DTR> FN$WIDTH(132)
DTR>
```

2. Use the SET COLUMNS_PAGE command to increase the length of the line DATATRIEVE can display on your terminal. The maximum limit on the columns-page setting is 255.

```
DTR> SET COLUMNS_PAGE = 132
DTR>
```

hatever the column setting on your terminal, you can continue a long input line using a hyphen (-) at the end of the line. When you use a hyphen, ATATRIEVE does not check the syntax of your input until you press RETURN ter a line that does not end in a hyphen. If the line you want to extend ends th a complete word, separate the hyphen from the word by entering a space. therwise, DATATRIEVE considers the characters at the beginning of the next ne to be part of the same character string.

ou cannot enter more than 255 characters on an extended input line.

6.1.2 Decreasing the Columns-Page Setting -- To decrease the number of lumns displayed, simply enter a SET COLUMNS_PAGE command:

```
R> SET COLUMNS_PAGE = 60
R>
```

1.6.2 Using SET ABORT

When DATATRIEVE executes an ABORT statement in a command file or proce dure while SET NO ABORT is in effect, it affects only the compound statement containing the ABORT statement. If SET ABORT is in effect, DATATRIEVE terminates the remainder of the command file or procedure. The same rules appl if you enter a CTRL/Z in response to a prompt.

If DATATRIEVE encounters a syntax or logical error in a command file or proce dure, it returns you to the DTR> prompt whether or not you have used SET ABORT.

SET NO ABORT is the default setting when you invoke DATATRIEVE.

See Chapters 7 and 8 for a discussion of using ABORT and NO ABORT in controlling procedures and command files.

1.6.3 Using SET PROMPT

When you invoke DATATRIEVE, SET PROMPT is in effect. If you press RETURN before finishing a command or statement, DATATRIEVE prompts yo for the remaining required elements of that command or statement.

The following sequence of commands and statements shows how DATATRIEVE responds when SET PROMPT is in effect. After the line of text indicates the next required element, DATATRIEVE displays the CON> (continuation) prompt As long as the syntax of a command or statement is incomplete, DATATRIEVE uses CON> to tell you it is ready for further input.

```
DTR> READY
[Looking for dictionary path name]
CON> YACHTS
DTR> FIND
[Looking for "FIRST", domain name, or collection name]
CON> FIRST
[Looking for value expression]
CON> 1
[Looking for name of domain, collection, or list]
CON> YACHTS
[1 record found]
DTR>
```

Notice that DATATRIEVE stops prompting as soon as you enter elements that comprise a syntactically complete command or statement. For example, READY YACHTS is complete, and DATATRIEVE does not prompt for any further element. Similarly, when you enter FIND FIRST 1 YACHTS, DATATRIEVE does not prompt you for a CROSS clause, a Boolean expression, or a SORTED BY clause.

hen SET NO PROMPT is in effect, DATATRIEVE does not display the text out the next required element. It does, however, use the CON> prompt when e syntax is incomplete. This identical sequence of inputs shows how ATATRIEVE responds when SET NO PROMPT is in effect:

```
R> SET NO PROMPT
R> FIND
N> FIRST
N> 1
N> YACHTS
 record found]
R>
```

)te that SET NO PROMPT does not suppress the messages DATATRIEVE ;plays about the results of commands and statements.

3.4 Using SET SEARCH

)u activate the DATATRIEVE Context Searcher with the SET SEARCH com- and. The Context Searcher is a facility to help you get easy access to list items d DBMS sets. It automatically searches lists and DBMS sets when necessary resolve the names of data items. NO SEARCH is the default setting. See apter 6 for a discussion of the Context Searcher.

3.5 Using SET FORM

:T FORM is the default when you start a DATATRIEVE session. You can con-)l whether or not DATATRIEVE uses forms to display records on your VT52-, '100- or VT200-family terminal. SET FORM must be in effect both when you t access to a domain whose definition includes a FORM clause and when you ter PRINT, STORE, or MODIFY commands related to that domain. SET NO)RM prevents DATATRIEVE from using its forms interface. See Chapter 13 · a discussion of using forms with DATATRIEVE.

3.6 Using SET VERIFY

ATATRIEVE's SET VERIFY command displays lines from DATATRIEVE nmand files when those command files are invoked. It also displays the con- its of the edit buffer when you exit an edit buffer from within DATATRIEVE.

:T VERIFY does not display lines from command files run from the DCL)mpt in an invocation command line, unless the command file itself contains a :T VERIFY command. And SET VERIFY does not display lines from ATATRIEVE procedures stored in the CDD, whether they are run from within ATATRIEVE or from the DCL prompt. (See the section later in this chapter for)re information on issuing DATATRIEVE commands from the DCL prompt.)

SET NO VERIFY turns off display of command file lines. The SHOW SET_UP command tells you whether SET VERIFY or SET NO VERIFY is in effect durin your DATATRIEVE session.

There are comparable DCL settings, SET VERIFY and SET NOVERIFY. If DC SET VERIFY is in effect, lines from DATATRIEVE command files run from the DCL prompt are displayed.

You may want to use SET VERIFY as a debugging tool for DATATRIEVE proc dures. You can do this by either:

- Creating your DATATRIEVE procedures in a command file at VMS level and executing them at the DTR> prompt
- Writing your DATATRIEVE procedures to a command file from an edit buffer or with the EXTRACT command, deleting the DEFINE PROCEDURE or REDEFINE PROCEDURE syntax, and executing that command file from DATATRIEVE

If there is a syntax error, you can see exactly where it occurs in the procedure. After you have debugged a procedure, you can then store it in the CDD.

1.6.7 Using SET SEMICOLON

When you change the setting to SET SEMICOLON, DATATRIEVE requires that you put a semicolon at the end of every statement you enter. DATATRIEV returns a CON> prompt each time you press RETURN to tell you the statemen is still incomplete. SET NO SEMICOLON is the default at the start of your DATATRIEVE session.

If you are entering a compound statement incorporating more than one logically complete statement, SET SEMICOLON enables you to end a component statement at the end of a line without having DATATRIEVE execute the statement immediately. When you have entered all the parts of the statement, you enter a semicolon and DATATRIEVE executes the entire statement.

1.6.8 Using SET LOCK WAIT

SET NO LOCK_WAIT is the default setting. With this setting, DATATRIEVE tries to access a locked record for 12 seconds. If the record does not become accessible during that period, you receive an error message and DATATRIEVE continues. If you have LOCK_WAIT set, DATATRIEVE keeps accessing a locke record indefinitely until it becomes available. For a discussion of locked records, see the material on the READY command in the *VAX DATATRIEVE Referenc Manual.*

.7 Controlling the Input of Dates and Currency

'ou can define VMS logical names to control the way DATATRIEVE handles the ıterpretation of dates and currency symbols. You can control the format)ATATRIEVE uses to interpret the input of dates by defining the Logical Name)TR$DATE_INPUT. This logical name affects only the interpretation of the ıput of dates and has nothing to do with the edit strings used for the output of ates.

'ou define DTR$DATE_INPUT with a three-character string containing one D)r day, one M for month, and one Y for year. Enclose the three-character string ı quotation marks.

'he command DEFINE DTR$DATE_INPUT "MDY" defines a date input of 3/12/09 as March 12, 1909. (MDY is the default interpretation DATATRIEVE ses for input of dates if you do not define DTR$DATE_INPUT.)

'able 1-1 shows the different combinations you can use.

able 1-1: Defining the Logical Name DTR$DATE_INPUT

Format	Input	Definition
"MDY"	03/12/09	March 12 1909
"DMY"	03/12/09	December 3 1909
"YDM"	03/12/09	September 12 1903
"YMD"	03/12/09	December 9 1903
"DYM"	03/12/09	September 3 1912
"MYD"	03/12/09	March 9 1912

he format you choose also controls the format DATATRIEVE uses to convert x-digit numeric strings (such as 810210) to dates.

ou can define DTR$DATE_INPUT at DCL command level (indicated by the dol- r sign prompt), or you can put the appropriate DEFINE command in your login)mmand file.

Table 1-2 shows the three logical names you can define to control the currency defaults of the VMS operating system.

Table 1-2: Currency Symbols

Logical Name	Default
SYS$CURRENCY	$
SYS$DIGIT_SEP	,
SYS$RADIX_POINT	.

The following examples demonstrate the effects of redefining these logical names In the first example, you redefine the default values:

```
$ DEFINE SYS$CURRENCY "#"
$ DEFINE SYS$DIGIT_SEP "."
$ DEFINE SYS$RADIX_POINT ","
```

In the next example, you can see the results of the changed definitions:

```
DTR> DECLARE NUM PIC 9(6)V99.
DTR> NUM = 12345.67
DTR> PRINT NUM USING $$$,$$$.99

   NUM

#12.345,67

DTR>
```

1.8 Issuing DATATRIEVE Commands from DCL Command Level

When you define a DCL symbol for invoking VAX DATATRIEVE, make sure you use the dollar sign ($) instead of the DCL RUN command. For example, define the name as the equivalent of SYSSYSTEM:DTR32, not RUN SYS$SYSTEM:DTR32. Defining the symbol this way lets you include DATATRIEVE commands and statements on the same line you use to start VAX DATATRIEVE. For example:

```
$ DTR32 :== $SYS$SYSTEM:DTR32
$ DTR32 READY YACHTS; PRINT FIRST 3 YACHTS
```

DCL command lines that contain DATATRIEVE commands and statements are also called **invocation command lines**. On a command line like this, you can put any legal combination of DATATRIEVE commands and statements, including th invocations of DATATRIEVE procedures and command files. Separate each command or statement with a semicolon.

ATATRIEVE executes the commands and statements in the sequence you enter em. When DATATRIEVE finishes executing those commands and statements, automatically ends your session and returns to the DCL prompt.

ıis example shows how the invocation command line operates:

```
DTR32 READY YACHTS; PRINT FIRST 3 YACHTS

                                LENGTH
                                 OVER
NUFACTURER    MODEL      RIG     ALL    WEIGHT BEAM  PRICE

LBERG         37 MK II   KETCH   37     20,000  12  $36,951
LBIN          79         SLOOP   26      4,200  10  $17,900
LBIN          BALLAD     SLOOP   30      7,276  10  $27,500
```

your invocation command line contains any of the following elements, ATATRIEVE prompts you for input before returning you to DCL command vel:

STORE and MODIFY statements without USING clauses

Prompting value expressions

ADT

SET GUIDE

EDIT

the DCL command line that invokes DATATRIEVE is embedded in a DCL ımmand procedure *and* DATATRIEVE prompts for input, you must reassign ıe logical name SYS$INPUT. See the section on using DCL command files in hapter 8 for more information.

.9 Using a DATATRIEVE Startup Command File

you frequently start your DATATRIEVE sessions with the same series of com- ands and statements, you can put them in a DATATRIEVE command file:

.. With a text editor, create a file containing the DATATRIEVE commands and statements just as you would enter them in an interactive session but without any of the DATATRIEVE prompts (DTR>, CON>, DFN>, RW>).

Here is a sample command file:

```
DECLARE X USAGE DATE.
X = "NOW"
IF FN$HOUR(X) BT 6 11 THEN PRINT SKIP, "GOOD MORNING." ELSE
   IF FN$HOUR(X) BT 12 16 PRINT SKIP, "GOOD AFTERNOON." ELSE
      PRINT SKIP, "YOU STILL HERE?"
READY CDD$TOP.DTR$LIB.DEMO.YACHTS,
      CDD$TOP.DTR$LIB.DEMO.OWNERS,
      CDD$TOP.DTR$LIB.DEMO.FAMILIES
SHOW READY
SHOW DICTIONARY
```

2. In your LOGIN.COM file or at DCL level, use the DCL DEFINE command to define DTR$STARTUP as a logical name for your startup command file:

   ```
   $ DEFINE DTR$STARTUP "device:[username]DTRSTART.COM"
   ```

   When you invoke DATATRIEVE, it translates the logical name DTR$STARTUP and executes the command file, including any output it generates, before it displays the first DTR> prompt on your terminal.

You can include in your DTR$STARTUP file any of the SET commands to establish the default settings for your DATATRIEVE session. You can also establish synonyms for any DATATRIEVE keywords, using the DECLARE SYNONYM command. See the description of DECLARE SYNONYM in the *VAX DATATRIEVE Reference Manual.*

Part 2
Manipulating Data

Writing Record Selection Expressions 2

ıce you have defined and stored your data, you probably want to manipulate it some way. Typical operations that you perform when you access data are:

- Displaying a group of records (PRINT, LIST, REPORT, or PLOT statements)
- Forming a temporary collection of records (FIND statement)
- Updating or changing a group of records (MODIFY statement)

ɛfore performing any of these operations, you must first decide which records ıu want to work with. You select those target records using a **record selection ɹpression** (RSE). The RSE identifies which records you want to work with and rms **a record stream**, that is, a group of records from a domain or collection. ATATRIEVE performs the specified operation on every record in the record ream.

ıe selected records can come from any of the following sources:

- Domains
- Collections
- Lists
- Rdb relations
- DBMS records

The RSE determines the content of the record stream. By including various clauses in the RSE, you can:

- Specify the number of records in the record stream (FIRST n, ALL clauses)
- Limit the record stream to records that meet a conditional test (WITH clause)
- Reduce the records in a record stream to unique values (REDUCED TO clause)
- Sort the records according to the values of one or more fields (SORTED BY clause)
- Join records from one or more domains or collections (CROSS clause)
- Access member or owner records of a DBMS set, (MEMBER, OWNER, or WITHIN clauses)

You may want to select records using a combination of these ways. An RSE can include any or all of the clauses listed. The following FIND statement includes a record selection expression that combines records from different sources, limits the records in the record stream to those that meet a particular condition, and sorts the records according to the value of two fields:

```
FIND EMPLOYEES CROSS
  JOB_HISTORY OVER
  EMPLOYEE_ID WITH
    JOB_END MISSING SORTED BY
    DEPARTMENT_CODE, LAST_NAME
```

When you are writing complex RSEs like this one, it is useful to indent parts of the expression so that you can easily see which relations are being crossed, which field values are used for selecting the records, and which fields are used for the sorting. The indentation helps you to see different parts of the RSE; it does not affect how the statement executes.

This chapter presents many examples to teach you how to use RSEs, an important tool of the DATATRIEVE language. It begins with simple RSEs and shows you how to use each clause of the RSE so that you can build complex RSEs. The examples show RSEs in PRINT statements, but you can also use them in all of the following DATATRIEVE statements:

- ERASE
- FIND
- FOR
- LIST

MATCH

MODIFY

PLOT

REPORT

Restructure

addition, you can use RSEs to specify subsets of records when you define view mains. See Chapter 5 for examples of RSEs in view domain definitions.

is chapter illustrates all of the previously listed operations except for accessing ords in a DBMS set, which is discussed in Chapter 14. In addition, a form of e RSE allows you to access list items from hierarchical records. This is dis- ssed in Chapter 6.

1 Displaying All the Records in a Domain

a domain does not contain many records, you may want to display all of the ords. In that case, you use the simplest form of an RSE, the domain name by elf. Because you want DATATRIEVE to print all of the records in that domain, u do not use any clauses of the RSE to select specific records. For example:

```
t> READY PERSONNEL
t> PRINT PERSONNEL

                FIRST       LAST                START          SUP
)    STATUS     NAME        NAME      DEPT      DATE      SALARY  ID

)12 EXPERIENCED CHARLOTTE   SPIVA     TOP   12-Sep-1972 $75,892 00012
391 EXPERIENCED FRED        HOWL      F11    9-Apr-1976 $59,594 00012
)43 EXPERIENCED CASS        TERRY     D98    2-Jan-1980 $29,908 39485
543 TRAINEE     JEFF        TASHKENT  C82    4-Apr-1981 $32,918 87465
132 TRAINEE     THOMAS      SCHWEIK   F11    7-Nov-1981 $26,723 00891
156 TRAINEE     HANK        MORRISON  T32    1-Mar-1982 $30,000 87289
162 EXPERIENCED BILL        SWAY      T32    5-May-1980 $54,000 00012
165 EXPERIENCED JOANNE      FREIBURG  E46   20-Feb-1980 $23,908 48475
185 EXPERIENCED DEE         TERRICK   D98    2-May-1977 $55,829 00012
175 EXPERIENCED GAIL        CASSIDY   E46    2-May-1978 $55,407 00012
573 TRAINEE     SY          KELLER    T32    2-Aug-1981 $31,546 87289
)01 EXPERIENCED DAN         ROBERTS   C82    7-Jul-1979 $41,395 87465
343 TRAINEE     BART        HAMMER    D98    4-Aug-1981 $26,392 39485
)23 EXPERIENCED LYDIA       HARRISON  F11   19-Jun-1979 $40,747 00891
'64 EXPERIENCED JIM         MEADER    T32    4-Apr-1980 $41,029 87289
375 EXPERIENCED MARY        NALEVO    D98    3-Jan-1976 $56,847 39485
```

(continued on next page)

```
87289 EXPERIENCED LOUISE     DEPALMA    G20  28-Feb-1979 $57,598 00012
87465 EXPERIENCED ANTHONY    IACOBONE   C82   2-Jan-1973 $58,462 00012
87701 TRAINEE     NATHANIEL  CHONTZ     F11  28-Jan-1982 $24,502 00891
88001 EXPERIENCED DAVID      LITELLA    G20  11-Nov-1980 $34,933 87289
90342 EXPERIENCED BRUNO      DONCHIKOV  C82   9-Aug-1978 $35,952 87465
91023 TRAINEE     STAN       WITTGEN    G20  23-Dec-1981 $25,023 87289
99029 EXPERIENCED RANDY      PODERESIAN C82  24-May-1979 $33,738 87465

DTR>
```

After you ready the domain, the PRINT PERSONNEL statement displays all th records in the PERSONNEL domain. The source for the RSE is PERSONNEL, the name of the domain.

To indicate clearly that you want the record stream to include all the records, yo can include the keyword ALL before the source of the RSE. Because the ALL is optional, PRINT ALL PERSONNEL is equivalent to PRINT PERSONNEL.

2.2 Limiting the Number of Records in the Record Stream

There are several ways to limit the number of records in the record stream. One way is to restrict the record stream to the first n records in the domain or collection. This type of RSE is useful when you know the order of records and the exa number of records you wish to access.

To specify the number of records in the record stream, type FIRST followed by a number before typing the source for the RSE. For example:

```
DTR> PRINT FIRST 5 PERSONNEL

                    FIRST       LAST                START              SUP
 ID     STATUS      NAME        NAME       DEPT     DATE       SALARY  ID

00012 EXPERIENCED CHARLOTTE   SPIVA      TOP   12-Sep-1972 $75,892 00012
00891 EXPERIENCED FRED        HOWL       F11    9-Apr-1976 $59,594 00012
02943 EXPERIENCED CASS        TERRY      D98    2-Jan-1980 $29,908 39485
12643 TRAINEE     JEFF        TASHKENT   C82    4-Apr-1981 $32,918 87465
32432 TRAINEE     THOMAS      SCHWEIK    F11    7-Nov-1981 $26,723 00891

DTR>
```

In this case, the RSE is FIRST 5 PERSONNEL. DATATRIEVE displays the first five records in PERSONNEL, according to their order in the data file. An RSE can have the form FIRST n domain-name or FIRST n collection-name. If n is larger than the number of records in the domain or collection, DATATRIEVE displays all the records in that source.

3 Identifying the Records That Meet a Test

ten you are interested in grouping similar records together, regardless of their ysical position in the data file. You can restrict the record stream to those :ords that satisfy a specific condition by using the WITH clause of the RSE. fferent types of WITH clauses reflect different types of relationships between e values of the same field for different records. Records can be grouped if they e related by the following conditions:

There is a pattern to the characters comprising the field values

The field values fall into a specified range

The value for a field is or is not missing

A field value can or cannot be found in a table

3.1 Comparing Records by Pattern Recognition

ıu can group records if the characters of a field value match or do not match a ecified value. For example:

```
?> PRINT YACHTS WITH RIG = "MS"

                                    LENGTH
                                     OVER
NUFACTURER    MODEL      RIG        ALL    WEIGHT BEAM  PRICE

MERICAN       26-MS      MS         26      5,500  08  $18,895
ASTWARD       HO         MS         24      7,000  09  $15,900
ORD           MS 33      MS         33     14,000  11
NDSEY         39         MS         39     14,500  12  $35,900
GGER FD       M/S        MS         35     17,600  11

?>
```

is statement causes DATATRIEVE to examine each record of the YACHTS main, displaying only those records with the value MS for the RIG field. ATATRIEVE tests each record of YACHTS, identifying and then displaying :h record that meets the specified condition. The WITH clause lets you limit e record stream to the records you wish to access.

e expression RIG = "MS" is a Boolean expression. A Boolean expression con- ls a comparison between value expressions. A Boolean expression is either true false, depending on the values of the field and the value expression specified. e term that relates the value expressions is called a **relational operator**. In this ample, the relational operator is an equal sign (=).

When you use EQUAL (or = or EQ), NOT EQUAL (NE), or CONTAINING (CONT), you can list more than one value expression in the same Boolean. The following queries specify a group of value expressions for DATATRIEVE to compare with each field value:

```
DTR> PRINT YACHTS WITH BUILDER = "ALBIN", "ALBERG"

                                LENGTH
                                 OVER
MANUFACTURER    MODEL      RIG   ALL    WEIGHT BEAM  PRICE

 ALBIN          79         SLOOP  26      4,200  10  $17,900
 ALBIN          BALLAD     SLOOP  30      7,276  10  $27,500
 ALBIN          VEGA       SLOOP  27      5,070  08  $18,600
 ALBERG         37 MK II   KETCH  37     20,000  12  $36,951

DTR> PRINT YACHTS WITH RIG NE "SLOOP", "KETCH"

                                LENGTH
                                 OVER
MANUFACTURER    MODEL      RIG   ALL    WEIGHT BEAM  PRICE

 AMERICAN       26-MS      MS     26      5,500  08  $18,895
 EASTWARD       HO         MS     24      7,000  09  $15,900
 FJORD          MS 33      MS     33     14,000  11
 LINDSEY        39         MS     39     14,500  12  $35,900
 ROGGER FD      M/S        MS     35     17,600  11

DTR>
```

Note that the EQUAL (=) and NOT_EQUAL operators are case-sensitive:

```
DTR> FIND YACHTS WITH BUILDER = "Albin"
[0 records found]
DTR> FIND YACHTS WITH BUILDER NOT_EQUAL "Albin"
[113 records found]
```

Because the names of builders in the YACHTS domain were entered in uppercase letters, DATATRIEVE does not find any record for a builder named "Albin."

However, the CONT or CONTAINING operator is indifferent to the case of the letters and searches only for a particular sequence of letters. This operator also finds matches if there is agreement with a substring derived from the field value. The CONT operator finds the "ALBIN" records if you specify "Albin" or "bin" (a three-letter substring) or any other string of letters unique to ALBIN:

```
DTR> FIND YACHTS WITH BUILDER CONT "Albin"
[3 records found]
DTR> FIND YACHTS WITH BUILDER CONT "bin"
[3 records found]
```

the next example, DATATRIEVE finds and displays each record that contains ;her the substring "alb" or the substring "pears" in the value for BUILDER:

```
R> PRINT YACHTS WITH BUILDER CONT "alb", "pears"

                                LENGTH
                                 OVER
NUFACTURER    MODEL      RIG     ALL    WEIGHT BEAM  PRICE

LBERG         37 MK II   KETCH   37     20,000  12   $36,951
LBIN          79         SLOOP   26      4,200  10   $17,900
LBIN          BALLAD     SLOOP   30      7,276  10   $27,500
LBIN          VEGA       SLOOP   27      5,070  08   $18,600
EARSON        10M        SLOOP   33     12,441  11
EARSON        26         SLOOP   26      5,400  08
EARSON        26W        SLOOP   26      5,200  09
EARSON        28         SLOOP   28      7,850  09
EARSON        30         SLOOP   30      8,320  09
EARSON        35         SLOOP   35     13,000  10
EARSON        36         SLOOP   37     13,500  11
EARSON        365        KETCH   36     17,700  11
EARSON        39         SLOOP   39     17,000  12
EARSON        419        KETCH   42     21,000  13

R>
```

hen you want to find records with a field value starting with a particular bstring, use the STARTING WITH relational operator. For example, you might ınt to display data on all builders beginning with the letter "A" or with the bstring "Al":

```
R> PRINT YACHTS WITH BUILDER STARTING WITH "A"

                                LENGTH
                                 OVER
NUFACTURER    MODEL      RIG     ALL    WEIGHT BEAM  PRICE

LBERG         37 MK II   KETCH   37     20,000  12   $36,951
LBIN          79         SLOOP   26      4,200  10   $17,900
LBIN          BALLAD     SLOOP   30      7,276  10   $27,500
LBIN          VEGA       SLOOP   27      5,070  08   $18,600
MERICAN       26         SLOOP   26      4,000  08    $9,895
MERICAN       26-MS      MS      26      5,500  08   $18,895
```

(continued on next page)

```
DTR> PRINT YACHTS WITH BUILDER STARTING WITH "AL"

                                LENGTH
                                 OVER
MANUFACTURER    MODEL      RIG   ALL   WEIGHT BEAM  PRICE

 ALBERG        37 MK II   KETCH  37    20,000  12  $36,951
 ALBIN         79         SLOOP  26     4,200  10  $17,900
 ALBIN         BALLAD     SLOOP  30     7,276  10  $27,500
 ALBIN         VEGA       SLOOP  27     5,070  08  $18,600

DTR>
```

Note that the STARTING WITH relational operator is case-sensitive. If you do not specify the correct case of each character in the substring, DATATRIEVE does not find the record:

```
DTR> FIND YACHTS WITH BUILDER STARTING WITH "al"
[0 records found]
DTR>
```

2.3.2 Grouping Records When Values Fall Within a Range

DATATRIEVE lets you use a variety of relational operators to test if a field valu for a record falls within a specified range. These operators are:

- GREATER_THAN (>, GT, or AFTER)
- GREATER_EQUAL (GE)
- LESS_THAN (<, LT, or BEFORE)
- LESS_EQUAL (LE)
- BETWEEN (BT)

For example:

```
DTR> PRINT PERSONNEL WITH SALARY GREATER_THAN 54000

                  FIRST       LAST              START              SUP
 ID     STATUS    NAME        NAME     DEPT     DATE      SALARY   ID

00012 EXPERIENCED CHARLOTTE  SPIVA      TOP  12-Sep-1972 $75,892 00012
00891 EXPERIENCED FRED       HOWL       F11   9-Apr-1976 $59,594 00012
39485 EXPERIENCED DEE        TERRICK    D98   2-May-1977 $55,829 00012
48475 EXPERIENCED GAIL       CASSIDY    E46   2-May-1978 $55,407 00012
84375 EXPERIENCED MARY       NALEVO     D98   3-Jan-1976 $56,847 39485
87289 EXPERIENCED LOUISE     DEPALMA    G20  28-Feb-1979 $57,598 00012
87465 EXPERIENCED ANTHONY    IACOBONE   C82   2-Jan-1973 $58,462 00012
```

```
R> PRINT PERSONNEL WITH SALARY GREATER_EQUAL 54000

                    FIRST       LAST                START              SUP
D     STATUS        NAME        NAME       DEPT     DATE       SALARY  ID

012 EXPERIENCED CHARLOTTE   SPIVA       TOP   12-Sep-1972 $75,892 00012
891 EXPERIENCED FRED        HOWL        F11    9-Apr-1976 $59,594 00012
462 EXPERIENCED BILL        SWAY        T32    5-May-1980 $54,000 00012
485 EXPERIENCED DEE         TERRICK     D98    2-May-1977 $55,829 00012
475 EXPERIENCED GAIL        CASSIDY     E46    2-May-1978 $55,407 00012
375 EXPERIENCED MARY        NALEVO      D98    3-Jan-1976 $56,847 39485
289 EXPERIENCED LOUISE      DEPALMA     G20   28-Feb-1979 $57,598 00012
465 EXPERIENCED ANTHONY     IACOBONE    C82    2-Jan-1973 $58,462 00012
```

ote the difference between the two record streams. Bill Sway, who earns exactly 4,000, is included in the record stream when the Boolean expression is ALARY GREATER_EQUAL 54000." But he is excluded when the REATER_THAN operator is used. In other words, GREATER_EQUAL cludes a record if a field value is equal to the value expression specified, but REATER_THAN leaves the record out.

e LESS_THAN and LESS_EQUAL operators work in a similar manner. The SS_EQUAL operator includes a record if a field value is either less than or ual to the value expression specified.

e BETWEEN operator is the equivalent of the GREATER_EQUAL and SS_EQUAL operators combined. It searches for records with field values that within the range specified or equal to either of the value expressions that termine the range. In the following example, the Boolean expression identifies ecord stream that includes records with values for SALARY of $30,000 and 4,000:

```
R> PRINT PERSONNEL WITH SALARY BETWEEN 30000 AND 54000

                    FIRST       LAST                START              SUP
)     STATUS        NAME        NAME       DEPT     DATE       SALARY  ID

343 TRAINEE       JEFF        TASHKENT    C82    4-Apr-1981 $32,918 87465
156 TRAINEE       HANK        MORRISON    T32    1-Mar-1982 $30,000 87289
462 EXPERIENCED   BILL        SWAY        T32    5-May-1980 $54,000 00012
573 TRAINEE       SY          KELLER      T32    2-Aug-1981 $31,546 87289
001 EXPERIENCED   DAN         ROBERTS     C82    7-Jul-1979 $41,395 87465
023 EXPERIENCED   LYDIA       HARRISON    F11   19-Jun-1979 $40,747 00891
764 EXPERIENCED   JIM         MEADER      T32    4-Apr-1980 $41,029 87289
001 EXPERIENCED   DAVID       LITELLA     G20   11-Nov-1980 $34,933 87289
342 EXPERIENCED   BRUNO       DONCHIKOV   C82    9-Aug-1978 $35,952 87465
029 EXPERIENCED   RANDY       PODERESIAN  C82   24-May-1979 $33,738 87465
```

o additional relational operators that separate records according to ranges are FORE and AFTER. These operators are useful for comparing values for date

fields. BEFORE can be used interchangeably with LESS THAN, and AFTER can be substituted for GREATER THAN. For example:

```
DTR> PRINT PERSONNEL WITH START_DATE AFTER "1-Jan-1981"

                FIRST       LAST                START              SUP
 ID     STATUS  NAME        NAME      DEPT      DATE      SALARY   ID

12643 TRAINEE  JEFF        TASHKENT  C82    4-Apr-1981 $32,918 87465
32432 TRAINEE  THOMAS      SCHWEIK   F11    7-Nov-1981 $26,723 00891
34456 TRAINEE  HANK        MORRISON  T32    1-Mar-1982 $30,000 87289
48573 TRAINEE  SY          KELLER    T32    2-Aug-1981 $31,546 87289
49843 TRAINEE  BART        HAMMER    D98    4-Aug-1981 $26,392 39485
87701 TRAINEE  NATHANIEL   CHONTZ    F11   28-Jan-1982 $24,502 00891
91023 TRAINEE  STAN        WITTGEN   G20   23-Dec-1981 $25,023 87289

DTR>
```

This query finds all employees who started after January 1, 1981. If there had been an employee who started on that date, the record would not have been included.

2.3.3 Grouping Records Based on a MISSING VALUE

If a missing value for a field is defined in the record definition using the MISSING VALUE IS field definition clause, you can search for records that either have or do not have the missing value.

For example, in the PERSONNEL domain, a MISSING VALUE clause has been included in the record definition of the SUP_ID field. That missing value is set as zero. You can form an RSE that asks DATATRIEVE to search for any records containing the MISSING VALUE:

```
DTR> FIND PERSONNEL WITH SUP_ID MISSING
[0 records found]
```

You can also ask DATATRIEVE to search for records in which a field does not contain the MISSING VALUE you specified in the record definition:

```
DTR> FIND PERSONNEL WITH SUP_ID NOT MISSING
[23 records found]
DTR>
```

).4 Grouping Records by Reference to a Table

me domains are associated with domain or dictionary tables that refer to one of
: fields in the record. You can form an RSE that causes DATATRIEVE to look
the field value in the table. If the field value is in the table, DATATRIEVE
ludes the record in the record stream. An example of a table-based RSE is:

```
tSONNEL WITH SUP_ID IN SUP_TABLE
```

cords with supervisor identification numbers in the SUP_TABLE are included
the record stream.

).5 Summary of the Relational Operators

ble 2-1 summarizes all of the relational operators available to form Boolean
pressions in the WITH clause of an RSE.

ble 2-1: Conditional Comparisons for an RSE

Type of omparison	Relationship of Values in Boolean	Relational Operator	Boolean Expression
Pattern ecognition	Exact match (case-sensitive).	= EQUAL EQ	BUILDER = "ALBIN" "ALBIN" = BUILDER
	No match (case-sensitive).	NE NOT_EQUAL NOTEQUAL	BUILDER NE "ALBIN" "ALBIN" NE BUILDER
	Substring matches (not case-sensitive).	CONT CONTAINING	BUILDER CONT "bin"
	Beginning substring matches (case-sensitive).	STARTING WITH	BUILDER STARTING WITH "AL"

(continued on next page)

Table 2-1: Conditional Comparisons for an RSE (Cont.)

Type of Comparison	Relationship of Values in Boolean	Relational Operator	Boolean Expression
Value Within a Range	First value is greater.	> GT GREATER_THAN	PRICE > 50000 50000 > PRICE
	Date field value is later than the value expression.	AFTER	START_DATE AFTER "1-Jan-1981"
	First value is greater than or equal.	GE GREATER_EQUAL	PRICE GE 50000 50000 GE PRICE
	First value is less.	< LT LESS_THAN	PRICE < 20000 20000 < PRICE
	Date field value is earlier than the value expression.	BEFORE	START_DATE BEFORE "1-Jan-1981"
	First value is less than or equal.	LE LESS_EQUAL	PRICE LE 20000
	First value is between the two values or equal to one.	BT BETWEEN	PRICE BETWEEN 30000 AND 54000
Field Value Missing	Field value is the MISSING VALUE.	MISSING	PRICE MISSING
Look Up in Table	Field value is in the table.	IN table-name	RIG IN RIG_TABLE

2.3.6 Setting Up Multiple Tests with Compound Booleans

To set up multiple tests for records, you can join two or more Boolean expressions. The expressions that join Boolean expressions are called **Boolean operator**

There are four Boolean operators:

AND OR NOT BUT

sing AND, OR, and BUT, you can join two or more Boolean expressions to form single Boolean expression. NOT allows you to reverse the value of a Boolean pression.

you link Boolean expressions with AND or BUT, the resulting Boolean expres- on is true only if all the Boolean expressions linked with AND or BUT are true.

you link Boolean expressions with OR, the resulting Boolean expression is true any one of the Boolean expressions linked with OR is true.

you precede a Boolean expression with NOT, the resulting Boolean expression true if the Boolean expression following NOT is false. The reverse is also true: e resulting Boolean expression is false if the Boolean expression following NOT true.

ıe following examples show the use of Boolean operators:

```
R> PRINT PERSONNEL WITH START_DATE BEFORE "1-Jan-1979" AND
N> SALARY LT 36000

                  FIRST       LAST              START              SUP
D     STATUS      NAME        NAME      DEPT    DATE       SALARY  ID

342 EXPERIENCED BRUNO       DONCHIKOV  C82   9-Aug-1978 $35,952 87465
```

ıe first query shows that Bruno Donchikov is the only employee who started fore January 1, 1979 and is earning less than $36,000.

```
R> PRINT PERSONNEL WITH DEPT = "TOP" OR SALARY > 54000

                  FIRST       LAST              START              SUP
D     STATUS      NAME        NAME      DEPT    DATE       SALARY  ID

012 EXPERIENCED CHARLOTTE  SPIVA       TOP  12-Sep-1972 $75,892 00012
891 EXPERIENCED FRED       HOWL        F11   9-Apr-1976 $59,594 00012
485 EXPERIENCED DEE        TERRICK     D98   2-May-1977 $55,829 00012
475 EXPERIENCED GAIL       CASSIDY     E46   2-May-1978 $55,407 00012
375 EXPERIENCED MARY       NALEVO      D98   3-Jan-1976 $56,847 39485
289 EXPERIENCED LOUISE     DEPALMA     G20  28-Feb-1979 $57,598 00012
465 EXPERIENCED ANTHONY    IACOBONE    C82   2-Jan-1973 $58,462 00012

R>
```

ıe second query displays data on all employees who are either in the TOP partment or earning more than $54,000.

```
DTR> PRINT PERSONNEL WITH SALARY > 54000 BUT DEPT = "TOP"

                    FIRST       LAST              START              SUP
 ID     STATUS      NAME        NAME     DEPT     DATE      SALARY   ID

00012 EXPERIENCED CHARLOTTE  SPIVA       TOP   12-Sep-1972 $75,892 00012

DTR>
```

The third query displays data on all employees who earn more than $54,000 but who also are in the department TOP.

2.4 Joining Records from Two or More Sources

RSEs let you work with records from different sources. The CROSS clause of th RSE lets you form record streams by combining data from two or more sources records. It forms temporary relationships between records stored in different da files based on the relationship between field values in the different files. Joining records with the CROSS clause allows you to treat the data as though it derived from one data file.

With the CROSS clause, you can:

- Combine records from several domains, collections, or both.
- Compare and combine records from one domain.
- Substitute a single statement for nested FOR loops when comparing records.
- Flatten hierarchical domains to ease access to the items in hierarchical lists Chapter 6 discusses hierarchical domains.

This is the format of the RSE containing the CROSS clause:

```
[ FIRST n ]
[ ALL     ]   [context-var IN] rse-source

[CROSS [context-var IN] rse-source [OVER field-name]] [...]

[WITH boolean-expression] [REDUCED TO reduce-key [,...]]

[SORTED BY sort-key [,...]]
```

The format for rse-source is:

```
{ domain-name     }
{ collection-name }
{ list            }
{ rdb-relation-name }
{ dbms-record-name }  [ { MEMBER } [OF] [context-name.set-name] ]
                      [ { OWNER  }                              ]
                      [ { WITHIN }                              ]
```

2.4.1 Using CROSS to Combine Two Domains

Suppose you want to find the prices of individual boats in the YACHTS domain that belong to boat owners stored in the OWNERS domain. You want to combine OWNERS records with YACHTS records that have the same MODEL and MANUFACTURER. The RSE that forms this temporary combination of records is on the second input line of the following PRINT statement. The group field TYPE, which includes both MANUFACTURER and MODEL, is the primary key for the YACHT.DAT file. It is defined as NO DUP; as a result, no two boats can have the same value for TYPE.

```
DTR> PRINT NAME, TYPE, PRICE OF
CON> YACHTS CROSS OWNERS OVER TYPE

   NAME     MANUFACTURER    MODEL      PRICE

STEVE        ALBIN         VEGA        $18,600
HUGH         ALBIN         VEGA        $18,600
JIM          C&C           CORVETTE
ANN          C&C           CORVETTE
JIM          ISLANDER      BAHAMA       $6,500
ANN          ISLANDER      BAHAMA       $6,500
STEVE        ISLANDER      BAHAMA       $6,500
HARVEY       ISLANDER      BAHAMA       $6,500
TOM          PEARSON       10M
DICK         PEARSON       26
JOHN         RHODES        SWIFTSURE

DTR>
```

The OVER TYPE phrase takes the place of WITH OWNERS.TYPE = YACHTS.TYPE. This RSE forms a record stream of 11 records.

2.4.2 Joining Records from Collections Based on the Same Domain

In many cases, you will want to combine and compare records from the same domain. For instance, you may want to find those yachts built by different builders but with the same kind of rigging, or you may want to find any trainees who make more than experienced employees. In crosses like these, you must distinguish separate record selection expressions that refer to the same domain. DATATRIEVE provides several ways to perform such crosses. One way is to use an alias to rename a domain. This operation temporarily creates two domains from one so you can ready and join them as if they were two separate sources.

When you use the CROSS clause to form and combine two collections from the same domain, you must establish context for both collections. In order for DATATRIEVE to join records from collections based on a single domain, you must ready the domain twice, once using an alias. Otherwise, DATATRIEVE does not include records from both sources in the join.

This example shows what happens when you ready YACHTS once, form two collections, AMERICAN_YACHTS and ALBIN_YACHTS, and join the collections with a CROSS clause:

```
DTR> READY YACHTS
DTR> FIND AMERICAN_YACHTS IN YACHTS WITH BUILDER = "AMERICAN"
[2 records found]
DTR> FIND ALBIN_YACHTS IN YACHTS WITH BUILDER = "ALBIN"
[3 records found]
```

(continued on next page)

```
R> LIST AMERICAN_YACHTS CROSS ALBIN_YACHTS OVER RIG

NUFACTURER     : ALBIN
DEL            : 79
G              : SLOOP
NGTH_OVER_ALL  : 26
SPLACEMENT     :  4,200
AM             : 10
ICE            : $17,900
NUFACTURER     : ALBIN
DEL            : 79
G              : SLOOP
NGTH_OVER_ALL  : 26
SPLACEMENT     :  4,200
AM             : 10
ICE            : $17,900
          .
          .
          .
```

ATATRIEVE does not include the records with BUILDER = "AMERICAN" in e join.

you ready the source domain twice, once under an alias, DATATRIEVE cor- ctly joins the records from both sources. In the following example, ATATRIEVE treats each collection as though it originated from a different main:

```
R> READY YACHTS, YACHTS AS EXTRA
R> FIND AMERICAN_YACHTS IN YACHTS WITH BUILDER = "AMERICAN"
 records found]
R> FIND ALBIN_YACHTS IN EXTRA WITH BUILDER = "ALBIN"
 records found]
R> LIST AMERICAN_YACHTS CROSS ALBIN_YACHTS OVER RIG

NUFACTURER     : AMERICAN
DEL            : 26
G              : SLOOP
NGTH_OVER_ALL  : 26
SPLACEMENT     :  4,000
AM             : 08
ICE            :  $9,895
NUFACTURER     : ALBIN
DEL            : 79
G              : SLOOP
NGTH_OVER_ALL  : 26
SPLACEMENT     :  4,200
AM             : 10
ICE            : $17,900
          .
          .
          .
```

If at any time you forget how you have used an alias, use the SHOW READY command to see the domain name behind the alias. For example:

```
DTR> READY YACHTS, YACHTS AS EXTRA
DTR> SHOW READY
Ready sources:
   EXTRA:  Domain, RMS sequential, protected read
           <CDD$TOP.DTR$LIB.DEMO.YACHTS;1>
   YACHTS: Domain, RMS sequential, protected read
           <CDD$TOP.DTR$LIB.DEMO.YACHTS;1>
No loaded tables.
```

2.4.3 Using CROSS to Cross a Domain with Itself

Another way to compare and combine records from the same source is to use either the CROSS clause (without aliases), nested FOR loops, or view domains. See the FOR statement section in the *VAX DATATRIEVE Reference Manual* for more information about nested FOR loops. View domains are discussed in th views chapter in this book. This section explains how to use CROSS to compare records from the same domain.

Consider the question of how to find the yachts whose manufacturers make boa with more than one type of rigging. To do this, you need to loop through the YACHTS domain twice. First, you must search through all yachts and group them by manufacturer. Then, you must search through these collections to find those yachts with different riggings.

You can cross and compare the necessary record streams in a single RSE conta ing a CROSS clause:

```
DTR> PRINT BUILDER, A.RIG, RIG OF A IN YACHTS CROSS
[Looking for name of domain, collection, or list]
CON> YACHTS OVER BUILDER WITH A.RIG GT RIG

MANUFACTURER  RIG     RIG

 AMERICAN     SLOOP   MS
 CHALLENGER   SLOOP   KETCH
 CHALLENGER   SLOOP   KETCH
 GRAMPIAN     SLOOP   KETCH
          .
          .
          .
 PEARSON      SLOOP   KETCH

DTR>
```

The variable A (A IN YACHTS) is called a context variable. A context variable i a temporary name that identifies a record stream to DATATRIEVE . See Chapter 9 for an extended discussion of how to use context variables. Appendix contains detailed information about how DATATRIEVE establishes and interprets context variables.

ı the above example, DATATRIEVE establishes two sources, one called A IN ACHTS and the other called YACHTS. The OVER clause controls the compari- ɔn of records from the two sources. For each record from the source A IN ACHTS, DATATRIEVE retrieves only the records from the source YACHTS ıat have the same BUILDER value as the record from A IN YACHTS. The oolean expression WITH A.RIG GT RIG selects from the record stream the airs of records that have different values for RIG. The resulting record stream ɔntains information only about builders who make more than one type of rig.

ou could use the Boolean expression WITH A.RIG NE RIG to select the records ith two different RIG values. But if you use NE instead of GR, you get two com- nations for every pair of records that meet the criteria of the RSE. Using the T operator eliminates this duplication.

ne advantage this method has over nested FOR loops is that the statement with ıe CROSS clause is shorter than an equivalent statement with a FOR loop. (The vo methods take approximately the same amount of time to process.)

.5 Finding the Unique Field Values in the Record Stream

requently, a record stream contains several records that have the same values ır a specific field. To find the unique field values (that is, to eliminate duplicate eld values from the record stream), use the REDUCED TO clause of the RSE.

p to this point, the RSE clauses have let you limit the number of records in the ɔcord stream. The REDUCED TO clause of the RSE lets you limit the fields ithin each record in the record stream.

or example, if you want to know the name of all of the departments in the ERSONNEL domain, you can use this query:

```
:R> FIND PERSONNEL REDUCED TO DEPT
' records found]
:R> PRINT CURRENT

:PT

)2
)8
!6
.1
!0
)2
)P
```

ɔ process the RSE, DATATRIEVE searches the values for DEPT and finds ɔven unique values. DATATRIEVE then generates a collection of seven records ith values for the DEPT field only.

ɔmetimes you want to know all the unique combinations of values for several

fields in the record. To find the combinations of values for DEPT and STATUS, use the RSE "PERSONNEL REDUCED TO DEPT, STATUS":

```
DTR> FIND PERSONNEL REDUCED TO DEPT, STATUS
[12 records found]
DTR> PRINT CURRENT

DEPT   STATUS

C82  EXPERIENCED
C82  TRAINEE
D98  EXPERIENCED
D98  TRAINEE
E46  EXPERIENCED
F11  EXPERIENCED
F11  TRAINEE
G20  EXPERIENCED
G20  TRAINEE
T32  EXPERIENCED
T32  TRAINEE
TOP  EXPERIENCED
```

DATATRIEVE finds 12 unique combinations of values and forms a collection wit 12 records. Each record in the collection has values for only two fields, DEPT an STATUS.

The REDUCED TO clause is a powerful tool for forming relational queries. For example, the following query uses two RSEs to display the names of all the supe visors and the departments they manage:

```
DTR> FOR A IN PERSONNEL REDUCED TO SUP_ID
[Looking for statement]
CON> PRINT DEPT, NAME, ID OF PERSONNEL WITH ID = A.SUP_ID

       FIRST        LAST
DEPT   NAME         NAME      ID

TOP  CHARLOTTE  SPIVA       00012
F11  FRED       HOWL        00891
D98  DEE        TERRICK     39485
E46  GAIL       CASSIDY     48475
G20  LOUISE     DEPALMA     87289
C82  ANTHONY    IACOBONE    87465

DTR>
```

This query finds every employee who is a supervisor, that is, whose ID equals one of the values specified by the REDUCED TO clause. The RSE "A IN PERSONNEL REDUCED TO SUP_ID" asks DATATRIEVE to develop a record stream (A) with all of the supervisor IDs. Then for each supervisor ID, DATATRIEVE searches through all the PERSONNEL records again for matche on the ID field. When DATATRIEVE finds a match, it displays the ID, NAME, and DEPT of the employee.

o do this, the RSE must include a context variable, A, to refer to the SUP_ID of ıe first record stream. The context variable is then used in the Boolean expres- on ID = A.SUP_ID. If you used the Boolean expression ID = SUP_ID,)ATATRIEVE would consider SUP_ID to be a field in the records of the second ɔcord stream. DATATRIEVE would then find all employees whose personal ID is ıe same as their supervisor's ID. (That is, all employees who supervise them- ɛlves.) The value expression A.SUP_ID unambiguously refers to a field value om records in the first record stream. See Chapter 9 and Appendix A for more ıformation about context variables.

.6 Sorting the Record Stream by Field Values

/hen you use a PRINT statement to display a record stream, the order of the ɔcords is determined by the keys defined for the data file. However, you can use ıe SORTED BY clause of the RSE to impose a different sort order on the record ;ream.

or example, the records in PERSONNEL are already sorted by ID, the primary ɛy for the data file. But if you are interested in the employees for each depart- ıent, you can sort the records by DEPT. To break down each department into ɔperienced workers and trainees, specify STATUS as an additional sort key. The ıllowing query sorts the first nine PERSONNEL records according to DEPT and TATUS:

```
[R> PRINT FIRST 9 PERSONNEL SORTED BY DEPT, STATUS

                    FIRST       LAST                START              SUP
[D     STATUS       NAME        NAME      DEPT      DATE       SALARY  ID

'465 EXPERIENCED ANTHONY     IACOBONE    C82    2-Jan-1973 $58,462 00012
)342 EXPERIENCED BRUNO       DONCHIKOV   C82    9-Aug-1978 $35,952 87465
)029 EXPERIENCED RANDY       PODERESIAN  C82   24-May-1979 $33,738 87465
)001 EXPERIENCED DAN         ROBERTS     C82    7-Jul-1979 $41,395 87465
ɔ643 TRAINEE     JEFF        TASHKENT    C82    4-Apr-1981 $32,918 87465
ɔ943 EXPERIENCED CASS        TERRY       D98    2-Jan-1980 $29,908 39485
)485 EXPERIENCED DEE         TERRICK     D98    2-May-1977 $55,829 00012
ı375 EXPERIENCED MARY        NALEVO      D98    3-Jan-1976 $56,847 39485
)843 TRAINEE     BART        HAMMER      D98    4-Aug-1981 $26,392 39485

:R>
```

he SORTED BY clause overrides the order of the records in the data file, but it)es not change the physical order of the records in the data file.

ou can also sort a record stream according to a value expression based on a field ılue. For example, you could sort by the year of START_DATE by using the

value expression FN$YEAR (START_DATE) as a sort key:

```
DTR> READY PERSONNEL
DTR> FIND FIRST 9 PERSONNEL SORTED BY FN$YEAR (START_DATE)
DTR> PRINT ID, NAME, SALARY,
CON>     (FN$YEAR (START_DATE)) ("EMPLOYED"/"SINCE") USING 9999

        FIRST        LAST              EMPLOYED
 ID     NAME         NAME     SALARY    SINCE

00012 CHARLOTTE   SPIVA       $75,892    1972
87465 ANTHONY     IACOBONE    $58,462    1973
84375 MARY        NALEVO      $56,847    1976
00891 FRED        HOWL        $59,594    1976
39485 DEE         TERRICK     $55,829    1977
48475 GAIL        CASSIDY     $55,407    1978
90342 BRUNO       DONCHIKOV   $35,952    1978
99029 RANDY       PODERESIAN  $33,738    1979
87289 LOUISE      DEPALMA     $57,598    1979

DTR>
```

The SORTED BY clause lets you produce reports with data records divided into groups. In the last example, using the value expression FN$YEAR (START_DATE) as a sort key lets you report on employees grouped by the year they were first employed. For more information on creating such control group reports, see the *VAX DATATRIEVE Guide to Writing Reports.* For information on DATATRIEVE functions such as FN$YEAR, see the *VAX DATATRIEVE Reference Manual.*

Entering New Data 3

his chapter explains how to enter new data in RMS domains using the STORE atement, prompting value expressions, and the TAB key. In addition, refer to ıe following chapters for advanced topics:

Using Forms with DATATRIEVE -- for information on storing data with forms

Using DATATRIEVE with DBMS -- for information on storing data in DBMS databases

Using DATATRIEVE with Rdb -- for information on storing data in Rdb databases

.1 Using the STORE Statement

ɔu can create a record in the data file with the STORE statement. You can also ıe the STORE statement to assign values to fields. When you enter a STORE atement followed by a domain name, DATATRIEVE prompts you for the values each field in the record. If you enter a field list or the USING clause, ATATRIEVE prompts you to enter only the specified fields. DATATRIEVE ıes not prompt you to enter REDEFINES or COMPUTED BY fields.

```
R> READY OWNERS WRITE<RET>
R> STORE OWNERS<RET>
ter NAME: BILL<RET>
ter BOAT_NAME: GLOOM<RET>
ter BUILDER: DOWN EAST<RET>
ter MODEL: 32T<RET>
R> FIND OWNERS WITH BOAT_NAME = "GLOOM"<RET>
 record found]
```

```
DTR> SELECT; PRINT<RET>

                BOAT
   NAME         NAME          BUILDER      MODEL

BILL       GLOOM              DOWN EAST   32T

DTR>
```

When you respond to a DATATRIEVE prompt, you must supply a value, a space, or a TAB character, not a value expression. You cannot supply the name of a variable or a field and expect DATATRIEVE to use the value associated with the variable or the value associated with the field. DATATRIEVE interprets the name in either case as a character string literal and uses the literal as the value when making the assignment.

3.2 The Effect of TAB on Prompts from STORE Statements

If you respond with a TAB and RETURN to a prompt from a STORE statement:

- If the field has a default value specified in its field definition, DATATRIEVE uses the default value to initialize the field.
- If the field has a missing value but not a default value specified in its field definition, DATATRIEVE uses the missing value to initialize the field.
- If the field has a default value and a missing value specified in its field definition, DATATRIEVE uses the default value to initialize the field.
- If the field has neither a default value nor a missing value specified in its field definition, DATATRIEVE initializes numeric fields as 0 and alphabetic and alphanumeric fields as spaces.

3.3 Using Direct Assignments

With the USING clause of the STORE statement, you can limit the number of fields to which you assign new values. In the USING clause, you specify only those fields you want to change.

When you store values in the fields of a new record with the USING clause of a STORE statement, DATATRIEVE uses the values you assign to initialize the fields specified in the USING clause. To initialize the fields that you do not include in the USING clause, DATATRIEVE takes one of three actions:

- If a field has a default value specified in its field definition, DATATRIEVE uses the default value to initialize the field, whether or not there is also a missing value specified.

If a field has a missing value but not a default value specified in its field definition, DATATRIEVE uses the missing value to initialize the field.

If a field has neither a DEFAULT VALUE clause nor a MISSING VALUE clause, DATATRIEVE initializes the field with spaces if it is alphabetic or alphanumeric or with 0 if it is numeric.

ıis example shows the different actions DATATRIEVE takes when you assign a ıited number of values with the USING clause of a STORE statement:

```
R> SHOW TEST_1<RET>
MAIN TEST_1 USING TEST_REC ON TEST1.DAT;<RET>

R> SHOW TEST_REC<RET>
CORD TEST_REC USING
 TOP.
 03 DEF_VAL1 PIC X(7)
    DEFAULT VALUE IS "DEFAULT".
 03 MISS_VAL1 PIC X(7)
    MISSING VALUE IS "MISSING".
 03 BOTH_1 PIC X(7)
    DEFAULT VALUE IS "DEFAULT"
    MISSING VALUE IS "MISSING".
 03 NEITHER_STR PIC X(3).
 03 NEITHER_NUM PIC 999.
 03 DEF_VAL2 PIC X(7)
    DEFAULT VALUE IS "DEFAULT".
 03 MISS_VAL2 PIC X(7)
    MISSING VALUE IS "MISSING".
 03 BOTH_2 PIC X(7)
    DEFAULT VALUE IS "DEFAULT"
    MISSING VALUE IS "MISSING".


ı> READY TEST_1 WRITE<RET>
ı> STORE TEST_1 USING<RET>
ɔoking for statement]
\> BEGIN<RET>
ɔoking for statement]
\>     DEF_VAL1 = "ONE"<RET>
\>     MISS_VAL1 = "TWO"<RET>
\>     BOTH_1 = "THREE"<RET>
\> END<RET>
ı> FIND TEST_1<RET>
 record found]
```

(continued on next page)

```
DTR> PRINT ALL<RET>

 DEF      MISS     BOTH    NEITHER NEITHER  DEF      MISS     BOTH
VAL1     VAL1      1        STR      NUM     VAL2     VAL2     2

ONE      TWO      THREE                 000    DEFAULT MISSING DEFAULT

DTR> STORE TEST_1<RET>
Enter DEF_VAL1: FOUR<RET>
Enter MISS_VAL1: FIVE<RET>
Enter BOTH_1: SIX<RET>
Enter NEITHER_STR: <TAB><RET>
Enter NEITHER_NUM: <TAB><RET>
Enter DEF_VAL2: <TAB><RET>
Enter MISS_VAL2: <TAB><RET>
Enter BOTH_2: <TAB><RET>
DTR> FIND TEST_1;SELECT LAST; PRINT<RET>

 DEF      MISS     BOTH    NEITHER NEITHER  DEF      MISS     BOTH
 VAL1     VAL1      1        STR      NUM     VAL2     VAL2     2

FOUR     FIVE     SIX                   000    DEFAULT MISSING DEFAULT

DTR>
```

3.4 Using Prompting Expressions in STORE Statements

In the USING clauses of STORE, you can use prompting value expressions to control your input to records in data files. You can use two forms of prompting value expressions: *.prompt and **.prompt. These value expressions let you control DATATRIEVE prompts for input.

Both forms of prompting value expressions require you to respond by entering values, not value expressions. You cannot enter the names of variables or fields, and you cannot enter expressions from DATATRIEVE tables or arithmetic, statistical, or concatenated expressions. You must enter numeric or character string literals appropriate to the data type of the field for which you are supplying a value. Do not enclose character string literals in quotation marks when you supply a value to a prompt. If you do, DATATRIEVE treats the quotation marks as part of the value.

If a *.prompt is part of a USING clause in a STORE statement, DATATRIEVE prompts you for a value each time it executes the statement. If the STORE statement is in a REPEAT, FOR, or WHILE loop, DATATRIEVE prompts you each time it executes the loop.

: a **.prompt is part of a USING clause in a STORE statement and if the TORE statement is in a REPEAT, FOR, or WHILE loop, DATATRIEVE rompts you only once, regardless of how many times it executes the loop. The *.prompt is useful for assigning one value to a number of records when you have) assign unique values to other fields in each of those records. The following xample shows the difference between the two types of prompting value xpressions:

```
[R> SET NO PROMPT
[R> READY PHONES WRITE
[R> REPEAT 2
)N>      STORE PHONES USING
)N>      BEGIN
)N>           DEPARTMENT = **.DEPARTMENT
)N>           LOCATION = **.LOCATION
)N>           NAME = *.NAME
)N>           NUMBER = *.NUMBER
)N>      END
ıter DEPARTMENT: CED<RET>
ıter LOCATION: MK3<RET>
ıter NAME: Gardens, Marvin<RET>
ıter NUMBER: 555-1776<RET>
ıter NAME: D'Ecor, Espree<RET>
ıter NUMBER: 555-1812<RET>
[R>
```

Modifying Data 4

'his chapter explains how to use the MODIFY statement to change data in exist-
ıg records. In addition, refer to the following chapters for advanced topics:

Using Hierarchies -- for information on modifying records with repeating fields (fields defined with an OCCURS clause in the record definition)

Using Forms with DATATRIEVE -- for supplementary information on modifying data with forms

Using DATATRIEVE with DBMS -- for supplementary information on modifying data stored in DBMS databases

Using DATATRIEVE with Rdb -- for supplementary information on modifying data stored in Rdb databases

he MODIFY statement has the following formats:

ormat 1

```
IODIFY [ALL] field-name [,...]
    USING statement-1
    [VERIFY [USING] statement-2]
    [OF rse]
```

ormat 2

```
IODIFY [ALL] rse
    USING statement-1
    [VERIFY [USING] statement-2]
```

When you modify records, you must ready the associated domain for modify or write access. Then perform the following five steps:

- Decide on a record source (domain or collection).
- Specify the records you want from the record source.
- Specify the fields whose values you want to change.
- Assign new values to those fields.
- Optionally, specify any validation requirements that are not part of the record definition.

The order of these steps may be the easiest and most logical, but keep in mind that when you write the DATATRIEVE statements that carry out a modify operation, the order in which you specify the logical steps can vary. DATATRIEVE also lets you choose among alternative methods to accomplish the same logical step. In addition, you do not have to include syntax for all the steps in the MODIFY statement itself.

This abundance of alternatives can be confusing when you first start to use DATATRIEVE. If you always think about DATATRIEVE syntax and examples in terms of which statement or clause accomplishes which logical step, you will find it easier to write the statements best suited to what you want to do.

The following examples illustrate some of the ways to modify data. The text preceding each example describes the objective of the modify operation. Comments to the right of the DATATRIEVE input lines identify the logical steps in the modify operation. In the first two examples, the NO PROMPT setting is in effect so DATATRIEVE's "Looking for..." prompts do not appear. These informational prompts do not affect the outcome of the statements and commands you enter.

- Change one DEPT value in the PERSONNEL domain. In this case, you work with records directly from the domain and change all records containing the value. Note that you specify the record source and the record you want in the MODIFY statement itself:

```
DTR> READY PERSONNEL MODIFY
DTR> PRINT LAST_NAME, DEPT OF PERSONNEL WITH DEPT = "F11"

  LAST
  NAME     DEPT

HOWL        F11
SCHWEIK     F11
HARRISON    F11
CHONTZ      F11
```

```
DTR> MODIFY PERSONNEL -      ! <--- Specify record source
CON> WITH  DEPT = "F11"      ! <--- Select records
CON> USING DEPT =            ! <--- Specify field
CON> "F12"                   ! <--- Assign value
DTR>
DTR> PRINT LAST_NAME, DEPT OF PERSONNEL WITH DEPT = "F12"

  LAST
  NAME     DEPT

HOWL       F12
SCHWEIK    F12
HARRISON   F12
CHONTZ     F12
```

Modify the price of one record in YACHTS. Because you are working with a selected record from the CURRENT collection, you specify in the MODIFY statement only the name of the field you want to change:

```
DTR> READY YACHTS MODIFY
DTR> FIND YACHTS WITH BUILDER = "ALBIN"
[3 records found]
DTR> PRINT MANUFACTURER, MODEL, PRICE
No record selected, printing whole collection.

MANUFACTURER   MODEL      PRICE

 ALBIN         79         $17,900
 ALBIN         BALLAD     $27,500
 ALBIN         VEGA       $18,600

DTR> SELECT 3
DTR> MODIFY PRICE          ! <--- Specify field
Enter PRICE: 20,000        ! <--- Assign new value
DTR>
DTR> PRINT MANUFACTURER, MODEL, PRICE OF CURRENT

MANUFACTURER   MODEL      PRICE

 ALBIN         79         $17,900
 ALBIN         BALLAD     $27,500
 ALBIN         VEGA       $20,000
```

- Modify the records of all yachts manufactured by Albin to reflect a 10 percent price increase. Because you are working with the CURRENT collection, you do not include the name of the record source in the MODIFY statement itself. You do, however, include the keyword ALL in the MODIFY statement so that DATATRIEVE knows you are not modifying a selected record:

```
DTR> READY YACHTS MODIFY
DTR> FIND YACHTS WITH BUILDER = "ALBIN"
[3 records found]
DTR> PRINT MANUFACTURER, MODEL, PRICE
No record selected, printing whole collection.

MANUFACTURER    MODEL      PRICE

 ALBIN          79         $17,900
 ALBIN          BALLAD     $27,500
 ALBIN          VEGA       $18,600

DTR> MODIFY ALL -                    ! <--- Specify records
[Looking for statement]
CON> USING  PRICE =                  ! <--- Specify field
[Looking for value expression]
CON> PRICE * 1.1                     ! <--- Assign values
DTR>
DTR> PRINT MANUFACTURER, MODEL, PRICE OF CURRENT

MANUFACTURER    BEAM       PRICE

 ALBIN          79         $19,690
 ALBIN          BALLAD     $30,250
 ALBIN          VEGA       $20,460

DTR>
```

As you can see, DATATRIEVE gives you a great deal of flexibility when you modify records. You can write statements so that DATATRIEVE prompts you to enter values for the fields you want to change, or you can specify field values directly. Your MODIFY statement can process records in an RSE, or it can default to the CURRENT collection.

The following sections explain:

- Modifying records in a collection
- Modifying records in a record selection expression
- Avoiding common mistakes in modifying records

Using DATATRIEVE prompts when modifying data

Checking for valid values in the MODIFY statement

ote that the optional clause VERIFY [USING] validation-statement clause is cluded in all syntax formats so that you can determine where you include that ause in the MODIFY statement. Section 4.5 discusses what the VERIFY clause n contain and provides examples of its use.

ote also that no matter what form of the MODIFY statement you choose, you nnot modify the value of an index key field that has been defined so that anges are not allowed. For example, you can never modify the value of a field at is the primary key for an indexed data file.

.1 Modifying Records in the CURRENT Collection

rming a collection and then using that collection as the record source for a odify operation is generally easier than trying to include the record selection ntax in the same DATATRIEVE statement that specifies fields and assigns lues. The FIND statement forming the collection specifies the record source d does most of the work to select the records you want. You can then simply pe PRINT ALL and press the RETURN key to check the contents of the entire llection before and after the modify operation. You can also select a record and en type PRINT and press RETURN to display the same information for a par-ular record.

Note

While it may be easier to use a collection as a record source, DATATRIEVE generally works slower when retrieving records from collections. For more information on optimizing DATATRIEVE queries, see the chapter on improving performance in this manual. See Chapter 14 of the *VAX DATATRIEVE Handbook* for a discussion of the advantages and disadvantages of using collections.

1.1 Modifying a Selected Record in the CURRENT Collection

ter you form a collection and display the records it contains, use the SELECT atement to pick the record to be modified. SELECT 1 specifies the first record splayed, SELECT 2 specifies the second record displayed, and so forth. After u enter your SELECT statement, print the results to make sure you have the cord you want. If you discover you picked the wrong record, you can enter ELECT NONE and reenter a SELECT statement with a corrected record occurnce value.

You have a choice of the following formats to modify the selected record:

MODIFY [VERIFY [USING] validation-statement]

DATATRIEVE prompts once for each elementary field in the record definition and changes the field values to the ones you enter. You can simply type MODIFY and press the RETURN key, and DATATRIEVE gives you the opportunity to change each field in the selected record.

MODIFY field [,...] [VERIFY [USING] validation-statement]

DATATRIEVE prompts once for each elementary field that you name and once for each elementary field that is subordinate to each group field that you name. The values of those fields are changed to the ones you enter. For example, if you want to change only LAST_NAME and ZIP in a PERSONNEL record, you can enter MODIFY LAST_NAME, ZIP. Then DATATRIEVE prompts you only for those fields.

MODIFY USING assignment-statement
    [VERIFY [USING] validation-statement]

The assignment statement in the USING clause may be a series of PRINT and assignment statements that you include in a BEGIN-END block.

If you name an elementary field in an assignment statement, DATATRIEVE changes its value to the one you specify. For example:

```
DTR> MODIFY USING FIRST_NAME = "CASSANDRA"
DTR>
```

If you name a group field, DATATRIEVE gives you an error message stating "illegal assignment to a group field."

You are prompted to enter a value for a field only when the assignment statemer contains a prompting value expression for the field value.

FIND list-field
SELECT n
MODIFY item-field [VERIFY [USING] validation-statement]

Use this format to modify fields subordinate to a field defined with an OCCURS clause in a record definition or a view definition. The chapter on using hierarchie describes this format in detail.

)DIFY [ALL] list-item OF list

:e this format to modify all occurrences of fields subordinate to a field defined :h an OCCURS clause in a record definition or a view definition. The chapter on ing hierarchies describes this format in detail.

1.2 Modifying All Records in the CURRENT Collection

you want to change values in all the records of the CURRENT collection, you ve the choice of the following formats:

)DIFY ALL [VERIFY [USING] validation-statement]

\TATRIEVE prompts you once for each field in the record definition. The val- s you enter for fields change those fields in *every record* in the collection. Use .s format with care when your collection contains more than one record. Rarely you want to make every field in every record identical.

)DIFY ALL field [,...] [VERIFY [USING] validation-statement]

\TATRIEVE prompts once for each elementary item specified or implied by the ld name or names you specify. The values you enter for fields change those lds in every record in the collection.

:e this format with care when the CURRENT collection contains more than one :ord. If you enter the statement MODIFY ALL LAST_NAME and respond to \TATRIEVE's prompt by entering SMITH, every record in the CURRENT col- tion then contains SMITH in the LAST_NAME field. You use this format only change values that you want to be identical among records in the collection. r example, you might want to modify a field like SUPERVISOR_ID in a collec- n of records for employees that share the same supervisor.

)DIFY ALL USING assignment-statement
[VERIFY [USING] validation-statement]

e assignment statement in this format can be a series of PRINT and assign- :nt statements in a BEGIN-END block.

you name an elementary field in an assignment statement, DATATRIEVE anges its value in every record in the collection. If you name a group field, \TATRIEVE changes the value of each elementary field in the group in every :ord in the collection.

u are prompted to enter a value for a field only when the assignment statement ntains a prompting value expression.

This format of the MODIFY statement has the same effect as the preceding format. For example, if you enter the statement MODIFY ALL USING LAST_NAME = "SMITH", every record in the CURRENT collection contains SMITH in the LAST_NAME field.

4.2 Modifying All Records in a Record Selection Expression

The previous section showed that when you work with the CURRENT collection or a selected record, the MODIFY statement does not need to contain a record source or which particular records to modify. You can display records and change values using relatively few keystrokes.

When you do not want the MODIFY statement to assume a CURRENT collectic or a selected record for the modify operation, you have to specify both the record source and exactly which records you want to change as an RSE. You must include the RSE either in the MODIFY statement itself or in the FOR statemen component of a compound statement that also includes the MODIFY statement. If you plan for a display of records before and after the modify operation (definitely a good idea), you must include PRINT statements as well.

Modifying records in an RSE is the best method to use when writing procedures. Usually procedures contain compound statements, and you cannot use the DATATRIEVE statements that create and manipulate collections in compound statements. Writing compound statements that modify data is not difficult if you keep in mind the logical steps to a modify operation and make sure you include a of them in the statements you form.

4.2.1 Modifying Records Controlled by a FOR Statement

The FOR statement lets you modify each record in a record stream. The FOR statement creates a stream of records that are processed, one by one, by the nex statement. In a modify operation, that next statement can be either a MODIFY statement or a BEGIN-END block that includes a MODIFY statement.

You can choose among the following formats:

```
FOR rse MODIFY [VERIFY [USING] validation-statement]
```

DATATRIEVE prompts once for every elementary field in the record definition for each record in the record stream. In short, you can individually change all th fields in every record.

For example, if you enter FOR YACHTS WITH BUILDER = "ALBIN" MODIFY, you can change every field in every record that specifies Albin as the manufacturer.

)R rse MODIFY field [,...]
  [VERIFY [USING] validation-statement]

ATATRIEVE prompts once for each elementary field that you name and once
r each elementary field that is subordinate to each group field that you name in
ıe MODIFY statement. You can change every record, but only the fields
)ecified.

you enter FOR YACHTS WITH BUILDER = "ALBIN" MODIFY PRICE, for
:ample, you can change only the price field in every record that specifies Albin
: the manufacturer.

)R rse MODIFY USING statement
  [VERIFY [USING] validation-statement]

ıe statement in the USING clause can be a BEGIN-END block that contains
e PRINT and assignment statements you want DATATRIEVE to apply to each
cord in the record stream. DATATRIEVE changes values in every record for as
any fields as have assignment statements. DATATRIEVE does not prompt for
ılues unless you include prompting value expressions in the assignment
atements.

ıe following statement uses this format:

```
R PERSONNEL WITH ID = EMPLOYEE_VARIABLE
MODIFY USING
  BEGIN
    PRINT ID, EMPLOYEE_NAME, DEPT, SUP_ID, SKIP
    FIRST_NAME = *."first name (all caps) or TAB character"
    LAST_NAME = *."last name (all caps) or TAB character"
    DEPT = *."department code (all caps) or TAB character"
    SUP_ID = *."supervisor ID number or TAB character"
    PRINT SKIP, ID, EMPLOYEE_NAME, DEPT, SUP_ID
  END
```

ıe FOR rse limits the record stream to the record that has an ID field matching
e contents of a variable called EMPLOYEE_VARIABLE. The statements inside
e BEGIN-END block within the USING clause do three things:

Print the values of the fields that are being changed

Prompt the user to modify only certain fields of the record

Print the new values of the fields that were modified

The procedure FOR_RSE_MODIFY, which includes this statement, uses EMPLOYEE_VARIABLE to check that the user entered an existing employee ID:

```
DTR> SHOW FOR_RSE_MODIFY
PROCEDURE FOR_RSE_MODIFY
SET ABORT
DECLARE EMPLOYEE_VARIABLE PIC 9(5).
EMPLOYEE_VARIABLE = *."employee ID number"
WHILE NOT ANY PERSONNEL WITH ID = EMPLOYEE_VARIABLE
  BEGIN
    PRINT SKIP
    PRINT "Invalid employee number."
    DECLARE GET_OUT PIC X(5).
    GET_OUT = *."any letter if you want to stop, TAB to try again"
    IF GET_OUT NOT = "" THEN
      ABORT "Exit from procedure" ELSE
        EMPLOYEE_VARIABLE = *."employee ID number"
  END
READY PERSONNEL MODIFY
SET NO ABORT
FOR PERSONNEL WITH ID = EMPLOYEE_VARIABLE
  MODIFY USING
    BEGIN
      PRINT ID, EMPLOYEE_NAME, DEPT, SUP_ID, SKIP
      FIRST_NAME = *."first name (all caps) or TAB character"
      LAST_NAME = *."last name (all caps) or TAB character"
      DEPT = *."department code (all caps) or TAB character"
      SUP_ID = *."supervisor ID number or TAB character"
      PRINT SKIP, ID, EMPLOYEE_NAME, DEPT, SUP_ID
    END
FINISH PERSONNEL
END_PROCEDURE

DTR>
```

See the chapter on using procedures in this manual and the *VAX DATATRIEVE Handbook* for information on procedures and compound statements.

FOR rse MODIFY list-rse USING
    assignment-statement [VERIFY [USING] validation-statement]

FOR rse FOR list-rse MODIFY [field-name [,...]]
    [VERIFY [USING] validation-statement]

You can use these two formats to change the values of records with repeating fields. See the chapter on using hierarchies for more information.

If you include the CURRENT collection as the RSE in a FOR statement, you can easily display all the records that are changed by simply entering PRINT ALL.

his is useful when the RSE that gathers the records to be modified can no longer ıcate them after the modify operation.

: you do specify a collection as the record source in a FOR statement, keep in ıind that DATATRIEVE cannot do keyed retrieval on a collection. When you are orking with a large collection, therefore, the FOR statement should not attempt) limit further the records being processed (FOR CURRENT WITH ANY... is ne example). If it does, DATATRIEVE could process the modification more owly for each record than when it can use collection records in sequential order. o optimize performance when you specify a collection as the record source in a OR statement, have the FIND statement that forms the collection do all the ɛcord selection work.

efer to the chapter on improving DATATRIEVE performance for more detailed ıformation about improving DATATRIEVE's response time.

.2.2 Including the RSE Within the MODIFY Statement

ıcluding the records to be modified as part of the MODIFY statement is some-hat trickier than specifying the same information in FIND, FIND and SELECT, : FOR statements. Depending on what you want to do, you must specify the SE immediately after the keyword MODIFY (or MODIFY ALL), or you must rite the RSE at the end of the statement. It is, therefore, easier to make syntax :rors when you try to include an RSE in the MODIFY statement.

ou should not include an RSE within a MODIFY statement that changes hierar-ıical records (records that contain a list field or records from a view domain that ɔcesses more than one simple domain). If you do, DATATRIEVE may trap you ı an endless loop of "Re-enter" prompts for the repeating field values.

ou cannot specify different field values for each record in the MODIFY state-ıent RSE as you can when you modify records using a FOR statement RSE. The [ODIFY statement RSE means you supply only one value for each elementary eld you specify by name or imply with a group field name. The value you enter ɔplies to every record. Therefore, make sure you specify records that should con-ıin identical values for the field or fields you are changing.

eeping these cautions in mind, you can choose among the following formats:

ODIFY [ALL] [VERIFY [USING] validation-statement] OF rse

'se with care. If you simply enter a domain name, you can make every record in ıe domain identical.

```
MODIFY [ALL] field [,...]
    [VERIFY [USING] validation-statement] OF rse

MODIFY [ALL] rse USING
    assignment-statement [VERIFY [USING] validation-statement]

MODIFY [ALL] USING
    assignment-statement [VERIFY [USING] validation-statement]
    OF rse
```

The assignment statement in these formats can also be a series of assignment and PRINT statements in a BEGIN-END block.

4.3 Common Context Errors

Sections 4.1 and 4.2 contain the correct formats to modify the records you want to change from the record source you intend to use. This section describes some problems you can encounter if you inadvertently use the wrong format or combin format elements incorrectly. When you make this kind of mistake, DATATRIEVE either displays an error message or modifies records from the wrong record source, depending on the type of error you make.

4.3.1 Modifying All Records Rather Than Just the Selected Record

If you want to modify a selected record, do not include the keyword ALL in the MODIFY statement.

If you type MODIFY ALL, you are telling DATATRIEVE either to target the entire collection for the modify operation or to expect an RSE in the MODIFY statement. Because there may be times when this is your intention, DATATRIEVE does not display an error message. If you create this situation unintentionally, you can make all the records in the CURRENT collection identical for the field values you supply when you intended to change values in only on of those records.

4.3.2 Modifying the Wrong Selected Record

This error can occur if you forget to enter a SELECT statement for the CURRENT collection. There may be times when you have more than one collection in your workspace, and the collections formed before the CURRENT one have selected records. When you enter a MODIFY statement appropriate for a selected record and your CURRENT collection does not have one, DATATRIEV tries to apply the modify operation to the selected records you do have available. It modifies the most recently selected record to which it can apply your statement. If your statement contains a field name that is not in any of the available selected records, DATATRIEVE does not modify any record but tells you that th field name is used out of context.

ıere may be times when you want to modify a selected record in a collection her than the CURRENT one. In this case, you can enter SELECT NONE atements to "unselect" records associated with any collections formed after the ıe containing the selected record you want to change. In effect, this process leases selected records from the currect collection, and you can repeat it until ıu reach the individual records selected from the target collection.

ıe error you want to avoid in this situation is entering too many or too few ELECT NONE statements. You can use the PRINT statement to see which is e current selected record. You can also use SHOW CURRENT or SHOW OLLECTIONS to make sure that you are working with the collection you want.

3.3 Modifying Records in the Wrong RSE

ıis error occurs when you intend to modify records using a FOR statement SE, but you also intentionally include an RSE or the keyword ALL in the ODIFY statement itself. If you do this, you are telling DATATRIEVE to odify records specified in the MODIFY statement for as many iterations as ere are records in the FOR statement RSE. (The only time you want to do mething like this is when you modify repeating fields in a hierarchical record. hen you modify hierarchical records, however, the RSE in the MODIFY statement specifies an OCCURS field name as the record source, rather than a true cord source, such as a domain or collection.) If you inadvertently include two SEs in the combined statements that carry out a modify operation, the results n be unexpected.

the following example, the user intends to change the last name in the first ERSONNEL record. The superfluous RSE in the FOR statement causes ATATRIEVE to prompt for the field as many times as there are records in the ERSONNEL domain:

```
R> SET NO PROMPT
R> FOR PERSONNEL MODIFY FIRST 1 PERSONNEL USING BEGIN
N> PRINT
N> LAST_NAME = *."last name"
N> END

                  FIRST       LAST                START              SUP
D     STATUS      NAME        NAME     DEPT       DATE      SALARY   ID

012 EXPERIENCED CHARLOTTE  SPIVA      TOP  12-Sep-1972 $75,892 00012
ter last name: WHITE
012 EXPERIENCED CHARLOTTE  WHITE      TOP  12-Sep-1972 $75,892 00012
ter last name: <TAB>
012 EXPERIENCED CHARLOTTE  WHITE      TOP  12-Sep-1972 $75,892 00012
ter last name: <TAB>
012 EXPERIENCED CHARLOTTE  WHITE      TOP  12-Sep-1972 $75,892 00012
                                .
                                .
                                .
```

A correct statement in the previous example, would have been either:

```
MODIFY FIRST 1 PERSONNEL USING . . .
```

or

```
FOR FIRST 1 PERSONNEL MODIFY USING . . .
```

In the following example, the user intends to modify the DEPT field of only the employee record with ID number 34456. Because of the keyword ALL in the MODIFY statement, however, DATATRIEVE uses the value entered to modify the DEPT field of all the records in the CURRENT collection:

```
DTR> FIND PERSONNEL WITH DEPT = "T32"
[4 records found]
DTR>
No record selected, printing whole collection.

                    FIRST       LAST              START              SUP
 ID     STATUS      NAME        NAME     DEPT     DATE      SALARY   ID

34456 TRAINEE     HANK        MORRISON   T32   1-Mar-1982 $30,000 87289
38462 EXPERIENCED BILL        SWAY       T32   5-May-1980 $54,000 00012
48573 TRAINEE     SY          KELLER     T32   2-Aug-1981 $31,546 87289
83764 EXPERIENCED JIM         MEADER     T32   4-Apr-1980 $41,029 87289

DTR> FOR PERSONNEL WITH ID = 34456
CON> MODIFY ALL DEPT
Enter DEPT: F11
DTR> PRINT
No record selected, printing whole collection.

                    FIRST       LAST              START              SUP
 ID     STATUS      NAME        NAME     DEPT     DATE      SALARY   ID

34456 TRAINEE     HANK        MORRISON   F11   1-Mar-1982 $30,000 87289
38462 EXPERIENCED BILL        SWAY       F11   5-May-1980 $54,000 00012
48573 TRAINEE     SY          KELLER     F11   2-Aug-1981 $31,546 87289
83764 EXPERIENCED JIM         MEADER     F11   4-Apr-1980 $41,029 87289

DTR> ! YIPES!!!!
DTR>
```

To avoid this, the user could enter either:

```
SELECT 1; MODIFY DEPT
```

or

```
FOR PERSONNEL WITH ID = 34456 MODIFY DEPT
```

1.4 Using DATATRIEVE Prompts

There are two ways you can get DATATRIEVE to prompt you for values:

- Using forms of the MODIFY statement that do not require a USING clause
- Including a prompting value expression in the Assignment statements within the USING clause (for example, USING LAST_NAME = *."last name")

Having DATATRIEVE prompt you to enter values has the following advantages:

- You do not have to type in all the Assignment statements.
- If you enter an invalid value or one that is too large for the field, DATATRIEVE displays an error message and reprompts so you can try again.
- You do not have to enter nonnumeric values in quotation marks. In fact, DATATRIEVE treats quotation marks as part of the value, so you should not use them unless they are actually part of the field value.
- If you press TAB and then the RETURN key in response to a prompt for a field value, DATATRIEVE leaves the value of the field unchanged, regardless of any DEFAULT or MISSING values defined for the field. This can be useful if you are prompted to enter values for fields you decide not to change.
- If you respond with CTRL/Z to a prompt for a field value, DATATRIEVE does not change any field in the record you are currently changing. This is useful if you realize you made a mistake entering earlier values for that record.

  Remember, however, that entering CTRL/Z does not affect records you have finished modifying, only the one you are working with when you enter CTRL/Z. CTRL/Z also aborts the statement being executed. This means, for example, that if your statement is processing ten records and you enter CTRL/Z while modifying the fifth record, you do not get a chance to modify the remaining five records. You must reenter the statement to modify the fifth through tenth records.

Using prompting value expressions within the USING clause of a MODIFY statement is a very flexible method for assigning values to fields.

In the following example, the double asterisk prompts mean that the user is prompted to enter one field value that applies to all the records in the collection. The single asterisk prompts mean that the user is prompted to enter a field value for each record:

```
DTR> SET NO PROMPT
DTR> READY YACHTS MODIFY
DTR> FIND YACHTS WITH BEAM = 0
[5 records found]
DTR> FOR CURRENT MODIFY USING
CON>    BEGIN
CON>      PRINT SPECS
CON>      LOA = **.LOA
CON>      DISP = *.WEIGHT
CON>      BEAM = *.BEAM
CON>      PRICE = PRICE * 1.1
CON>    END

       LENGTH
        OVER
 RIG    ALL   WEIGHT BEAM  PRICE

SLOOP   32     9,500  00
Enter LOA: 33
Enter WEIGHT: 12000
Enter BEAM: 10
SLOOP   32    11,000  00  $29,500
Enter WEIGHT: <TAB><RET>
Enter BEAM: 11
SLOOP   31    13,600  00  $32,500
Enter WEIGHT: 15000
Enter BEAM: 12
SLOOP   35    23,200  00
Enter WEIGHT: <TAB><RET>
Enter BEAM: 13
SLOOP   32    14,900  00  $34,480
Enter WEIGHT: <TAB><RET>
Enter BEAM: 9
DTR> PRINT ALL

                                LENGTH
                                 OVER
MANUFACTURER    MODEL      RIG   ALL   WEIGHT BEAM  PRICE

 METALMAST    GALAXY     SLOOP   33    12,000  10
 O'DAY        32         SLOOP   33    11,000  11  $32,450
 RYDER        S. CROSS   SLOOP   33    15,000  12  $35,750
 TA CHIAO     FANTASIA   SLOOP   33    23,200  13
 WRIGHT       SEAWIND II SLOOP   33    14,900  09  $37,928

DTR>
```

You must respond to a prompt with a value rather than a value expression. For example, if you want to increase a price field by ten percent and let DATATRIEVE do the calculation, you must use direct assignment.

)ATATRIEVE will not let you enter PRICE * 1.1 in response to a prompt. If you re writing a procedure that needs this flexibility, you can prompt for part of the alue expression. For example, you can prompt for the price increase and include ıe arithmetic calculation of the new value for PRICE in your Assignment state- ıent (PRICE = PRICE * *."price increase").

.5 Ensuring Valid Values

)ATATRIEVE always checks the record definition that applies to the record you re changing to ensure that new field values have the correct length and data /pe. It also applies any VALID IF clauses in the record definition to the changed eld values. DATATRIEVE displays an error message and leaves the existing eld value untouched if a modify operation tries to enter a value that the record efinition does not allow.

'he VERIFY clause of the MODIFY statement lets you supplement the valida- on requirements in the record definition. It can also help you enforce security ıeasures for modification procedures.

'he format of the VERIFY clause is:

ERIFY [USING] validation-statement

'he validation statement can be a series of statements within a BEGIN-END lock.

'he VERIFY clause in the following example ensures that the first and last ames entered for an employee begin with a capital letter:

```
IR> FOR PERSONNEL WITH ID = *."ID number for record being changed"
)N>    MODIFY VERIFY USING
)N>      BEGIN
)N>        WHILE FIRST_NAME NOT BT "A" AND "Z"
)N>          BEGIN
)N>            PRINT SKIP, "Invalid first name"
)N>            FIRST_NAME = *."first name using CAPS"
)N>          END
)N>        WHILE LAST_NAME NOT BT "A" AND "Z"
)N>          BEGIN
)N>            PRINT SKIP, "Invalid last name"
)N>            LAST_NAME = *."last name using CAPS"
)N>          END

                          .
                          .
                          .

)N>      END
```

lote that DATATRIEVE does all verification only after all the data is entered for ıe record being modified.

Using View Domains 5

This chapter introduces the concept of view domains.

A view is a special type of domain that lets you select some (or all) fields in some or all) records from one or more domains. Using a view, you can refer to fields and field values in different domains without duplicating their records or data.

You define a view by creating a domain definition for it in the Common Data Dictionary (CDD). A view lets you read and modify selected field values. Because there is no data stored for a view, you cannot store or erase the records you retrieve with a view. Although you can combine records from various domains with the CROSS clause of the RSE, a view is the only type of domain that you can define in the CDD for working with data in more than one domain.

You define a view with the DEFINE DOMAIN command. The format of the command for defining a view is:

```
DEFINE DOMAIN view-path-name OF domain-path-name-1 [,...]  [BY | USING]

    level-number-1 field-name-1      OCCURS FOR rse-1 .

    level-number-2 field-name-2      {OCCURS FOR rse-n | FROM domain-path-name-n}

          .              .                  .
          .              .                  .
          .              .                  .

    [FORM [IS] form-name [IN] form-library]
```

After the keyword OF, you must list each domain that the view uses. You can specify the domains in any order, separating them with commas. You must end each field definition with a period and end the view definition with a semicolon.

You use two clauses to define the fields in a view:

- OCCURS FOR
- FROM

The top-level field must be defined with an OCCURS FOR clause. The record selection expression in the first OCCURS FOR clause determines the number of records in the view. Each subsequent OCCURS FOR clause creates a list within the view. Consequently, a view that contains more than one OCCURS FOR clause is always a hierarchy. (The first OCCURS FOR clause does *not* make the view a hierarchy. It only establishes the source record stream for the view.)

See Chapters 6 and 11 for a discussion of view domains that are hierarchies.

You establish the fields of data for the view with the FROM clause. It specifies the name of the field and the domain from which it derives. The domain must be the same domain named in the previous OCCURS FOR clause. The field name must be either a field name or a query name from that domain.

To ready a view, you must have the proper access privilege, and you must also have the same access privilege to the domain that the view uses. You ready a view domain directly; do not ready the domain that the view uses.

The next section shows how to define a view that contains a subset of records in the YACHTS domain. It uses that view to illustrate some general properties of views.

5.1 Views Using Subsets of Records

A view lets you work with a specific subset of records from another domain. For instance, you may want to work with the records for ketches only and no other rig type. The following example shows a view definition that allows you to work with four fields of the yachts that are ketches:

```
DTR> DEFINE DOMAIN KETCHES
DFN> OF YACHTS BY
DFN> 01 KETCH OCCURS FOR YACHTS WITH RIG EQ "KETCH".
DFN>    03 TYPE FROM YACHTS.
DFN>    03 LOA FROM YACHTS.
DFN>    03 PRICE FROM YACHTS.
DFN> ;
DTR> READY KETCHES
DTR> PRINT FIRST 4 KETCHES
```

```
                        LENGTH
                         OVER
NUFACTURER    MODEL      ALL     PRICE

LBERG       37 MK II     37     $36,951
HALLENGER   41           41     $51,228
ISHER       30           30
ISHER       37           37

R>
```

ıe view domain KETCHES, which is based on the single domain YACHTS, is ıt hierarchical because there is only one OCCURS FOR clause.

ɔu cannot store or erase records in a view. Otherwise, you can use a view just as ıu would any other domain. For example:

```
R> READY KETCHES MODIFY
R> FIND KETCHES WITH PRICE EQ 0
 records found]
R> PRINT ALL

                        LENGTH
                         OVER
NUFACTURER    MODEL      ALL     PRICE

ISHER       30           30
ISHER       37           37
EARSON      365          36
EARSON      419          42

R> FOR CURRENT PRINT THEN MODIFY PRICE

                        LENGTH
                         OVER
NUFACTURER    MODEL      ALL     PRICE

ISHER       30           30
ter PRICE: $30,000
ISHER       37           37
ter PRICE: 45,000
EARSON      365          36
ter PRICE: 32000
EARSON      419          42
ter PRICE: 54000
```

(continued on next page)

```
DTR> PRINT ALL

                          LENGTH
                           OVER
MANUFACTURER    MODEL      ALL     PRICE

 FISHER         30          30     $30,000
 FISHER         37          37     $45,000
 PEARSON        365         36     $32,000
 PEARSON        419         42     $54,000

DTR> FINISH
DTR>
```

Views using a subset of records are also useful with DBMS domains and Rdb rel tions and domains. See Chapters 14 and 15 for information about DBMS and R views.

5.2 Views Using Subsets of Fields

One type of view lets you refer to a subset of fields from the records of another domain. For example, the record definition for YACHTS contains seven elementary fields and three group fields:

```
DTR> SHOW YACHT
RECORD YACHT USING
01 BOAT.
  03 TYPE.
     06 MANUFACTURER PIC X(10)
        QUERY_NAME IS BUILDER.
     06 MODEL PIC X(10).
  03 SPECIFICATIONS
     QUERY_NAME SPECS.
     06 RIG PIC X(6)
        VALID IF RIG EQ "SLOOP","KETCH","MS","YAWL".
     06 LENGTH_OVER_ALL PIC XXX
        VALID IF LOA BETWEEN 15 AND 50
        QUERY_NAME IS LOA.
     06 DISPLACEMENT PIC 99999
        QUERY_HEADER IS "WEIGHT"
        EDIT_STRING IS ZZ,ZZ9
        QUERY_NAME IS DISP.
     06 BEAM PIC 99 MISSING VALUE IS 0.
     06 PRICE PIC 99999
        MISSING VALUE IS 0
        VALID IF PRICE>DISP*1.3 OR PRICE EQ 0
        EDIT_STRING IS $$$,$$$.
;
DTR>
```

If you want to work with only a few fields of the record, you can create a record definition for those fields and then create a domain and a data file containing on record for each record in YACHTS. The result is a data file that duplicates som field values in an existing data file (YACHT.DAT). Maintaining these two files s that they always contain the same field values would be difficult.

ou can define a view, however, that lets you look at just the fields in YACHTS
ıat you need, without duplicating field values. You also avoid the additional time
ıd overhead of creating another record definition and creating and updating two
ıta files:

```
'R> DEFINE DOMAIN MAKERS
'N> OF YACHTS BY
'N> 01 BOAT OCCURS FOR YACHTS.
'N>     03 TYPE FROM YACHTS.
'N>     03 RIG FROM YACHTS.
'N> ;
'R> READY MAKERS
'R> PRINT FIRST 6 MAKERS

NUFACTURER  MODEL     RIG

LBERG       37 MK II  KETCH
LBIN        79        SLOOP
LBIN        BALLAD    SLOOP
LBIN        VEGA      SLOOP
MERICAN     26        SLOOP
MERICAN     26-MS     MS
AYFIELD     30/32     SLOOP
LOCK I.     40        SLOOP
OMBAY       CLIPPER   SLOOP
UCCANEER    270       SLOOP

R>
```

3 Views Using More Than One Domain

ıe preceding sections showed how to use view domains to define a subset of
cords or fields from a single domain.

ews can also use more than one domain. There are two general ways different
mains can be combined in a view:

Combine record streams by using more than one OCCURS FOR clause. Each OCCURS FOR clause has its own RSE, and DATATRIEVE creates a hierarchical relationship between the record streams specified in each RSE.

For example, the sample domain SAILBOATS uses two OCCURS FOR clauses to create a hierarchical relationship between two record streams:

```
DTR> SHOW SAILBOATS
DOMAIN SAILBOATS OF YACHTS, OWNERS USING
01 SAILBOAT OCCURS FOR YACHTS.
        03 BOAT FROM YACHTS.
        03 SKIPPERS OCCURS FOR OWNERS WITH TYPE = BOAT.TYPE.
                05 NAME FROM OWNERS.
;

DTR>
```

Chapter 6 discusses creating hierarchies with view domains in more detail.

- Use a CROSS clause in the RSE of the OCCURS FOR clause to refer to more than one domain. The remainder of this section discusses using the CROSS clause in a view domain.

To illustrate how to use the CROSS clause in a view to combine records from more than one domain, look at the CROSS example in the chapter on using record selection expressions. Recall that this PRINT statement displays the NAME field from the OWNERS domain and the TYPE and PRICE fields from the corresponding records in the YACHTS domain:

```
DTR> PRINT NAME, YACHTS.TYPE, PRICE OF YACHTS CROSS OWNERS OVER TYPE

OWNER
NAME   MANUFACTURER    MODEL      PRICE

STEVE  ALBIN           VEGA       $18,600
HUGH   ALBIN           VEGA       $18,600
JIM    C&C             CORVETTE
ANN    C&C             CORVETTE
JIM    ISLANDER        BAHAMA      $6,500
ANN    ISLANDER        BAHAMA      $6,500
STEVE  ISLANDER        BAHAMA      $6,500
HARVE  ISLANDER        BAHAMA      $6,500
TOM    PEARSON         1OM
DICK   PEARSON         26
JOHN   RHODES          SWIFTSURE

DTR>
```

You can define a view domain, CROSS_SAILBOATS, to get the same results:

- Include the RSE YACHTS CROSS OWNERS OVER TYPE after the OCCURS FOR clause.
- Specify the fields you want to include in the domain--NAME, TYPE, and PRICE--in the FROM clauses of the view domain.

The following example shows the definition for CROSS_SAILBOATS. It shows how a simple PRINT statement produces the same results as the previous PRIN statement that used the CROSS clause.

```
DTR> SHOW CROSS_SAILBOATS
DOMAIN CROSS_SAILBOATS OF YACHTS, OWNERS USING
01 SAILBOAT OCCURS FOR YACHTS CROSS OWNERS OVER TYPE.
        03 NAME FROM OWNERS.
        03 TYPE FROM YACHTS.
        03 PRICE FROM YACHTS.
;

DTR> READY CROSS_SAILBOATS
```

```
t> PRINT CROSS_SAILBOATS

NER
ME  MANUFACTURER    MODEL       PRICE

EVE  ALBIN          VEGA        $18,600
SH   ALBIN          VEGA        $18,600
1    C&C            CORVETTE
N    C&C            CORVETTE
1    ISLANDER       BAHAMA       $6,500
N    ISLANDER       BAHAMA       $6,500
EVE  ISLANDER       BAHAMA       $6,500
RVE  ISLANDER       BAHAMA       $6,500
1    PEARSON        10M
CK   PEARSON        26
IN   RHODES         SWIFTSURE

t>
```

4 Advantages and Disadvantages of Using Views

ace you can define a view with any RSE that you might type interactively, view mains are convenient substitutes for typing complex record selection expres- ns you use often.

ie of the greatest advantages of a view is that you can use it to combine fields m an Rdb database with fields from RMS (file-structured) domains and DBMS mains. You can have a single view that brings together data from these differ- t types of databases.

view domain is also a convenient way to create a dynamic hierarchy. By using e OCCURS FOR clause, you can create temporary list fields. You then have the ility to display data in hierarchical form without being tied to hierarchical cords for other tasks.

u can also use a view to mask the data in certain fields from users who do not ed to see it. Select the fields you want the user to see from each underlying main and define a view that uses only those fields. However, because users ist also have access to the underlying domains, views cannot keep users from rieving sensitive data directly from those domains.

ie very important disadvantage of using views lies in the danger of modifying cords from multiple sources. You must be careful when you modify values in a w based on more than one domain. If the field you are changing is stored in re than one data file, you are updating only one of those files for each field lue you enter.

If the view refers to a second domain based on the value of a field in the first domain, a change to a field value in the first domain can cause DATATRIEVE t select an unexpected record from the second domain. When you use a form to modify such a view, the field value you see on the screen may not be the value y are actually modifying. See the chapter on forms in this book for an example of this problem in modifying a view.

Observe the following cautions and restrictions when you use views that refer to more than one domain:

- Try to avoid updating with a view.
- Set up view domains that minimize duplicate fields.
- Remember that when a view contains more than one OCCURS FOR rse ea OCCURS FOR rse after the first creates a list field. All the rules and restri tions for handling hierarchical data apply to those fields.
- Do not modify a field in a view that uses the FORM IS clause when that fie forms the basis for selecting records from a second domain. (See the chapte on forms in this book for an example of this restriction.)

Using Hierarchies 6

DATATRIEVE, the term *hierarchy* refers to a one-to-many relationship
:tween record sources.

ith hierarchies, you can nest record streams to see a single record from one
cord source displayed with a combination of records from another record source.
ıis nesting establishes a "parent-child" relationship between the two record
reams. For each record in the outer, "parent" record stream, you see all records
the inner, "child" record stream. Parent records are displayed even if there are
› corresponding child records in the inner record stream.

›me examples of how this can be useful are:

One team with several players

One project with several workers

One employee with several previous jobs

One library with many books

One computer with several users

›r instance, a hierarchy could nest record streams from the YACHTS and
WNERS domains. A parent-child relationship between YACHTS and OWNERS
uld link the YACHTS record with a given make and model, ALBIN VEGA, say,
th all OWNERS records that had the same make and model fields.

ere is how DATATRIEVE displays such a hierarchy. It shows each YACHTS
cord and a list of records from OWNERS of people who bought that make and

model boat. Note that the hierarchy includes the YACHTS record for a boat, eve if there are no corresponding records in the OWNERS domain:

MANUFACTURER	MODEL	RIG	LENGTH OVER ALL	WEIGHT	BEAM	PRICE	OWNER NAME	BOA
ALBERG	37 MK II	KETCH	37	20,000	12	$36,951		
ALBIN	79	SLOOP	26	4,200	10	$17,900		
ALBIN	BALLAD	SLOOP	30	7,276	10	$27,500		
ALBIN	VEGA	SLOOP	27	5,070	08	$18,600	STEVE	DELIVERA
							HUGH	IMPULSE
AMERICAN	26	SLOOP	26	4,000	08	$9,895		
AMERICAN	26-MS	MS	26	5,500	08	$18,895		
BAYFIELD	30/32	SLOOP	32	9,500	10	$32,875		
BLOCK I.	40	SLOOP	39	18,500	12			
BOMBAY	CLIPPER	SLOOP	31	9,400	11	$23,950		
BUCCANEER	270	SLOOP	27	5,000	08			
BUCCANEER	320	SLOOP	32	12,500	10			
C&C	CORVETTE	SLOOP	31	8,650	09		JIM	EGRET
							ANN	EGRET
CABOT	36	SLOOP	36	15,000	12			

The ability to nest record streams is a powerful feature of DATATRIEVE. By set ting up a parent-child relationship between record streams, you can see parent records whether or not there are any records in the child record stream. Using hierarchies is the only way to display records in this way. Joining YACHTS and OWNERS with a CROSS statement does not show the YACHTS records that d not have a corresponding records in the OWNERS domain:

```
DTR> PRINT BOAT, NAME, BOAT_NAME OF YACHTS CROSS OWNERS OVER TYPE
```

MANUFACTURER	MODEL	RIG	LENGTH OVER ALL	WEIGHT	BEAM	PRICE	OWNER NAME	BOA
ALBIN	VEGA	SLOOP	27	5,070	08	$18,600	STEVE	DELIVER
ALBIN	VEGA	SLOOP	27	5,070	08	$18,600	HUGH	IMPULSE
C&C	CORVETTE	SLOOP	31	8,650	09		JIM	EGRET
C&C	CORVETTE	SLOOP	31	8,650	09		ANN	EGRET
ISLANDER	BAHAMA	SLOOP	24	4,200	08	$6,500	JIM	POTEMKI
ISLANDER	BAHAMA	SLOOP	24	4,200	08	$6,500	ANN	POTEMKI
ISLANDER	BAHAMA	SLOOP	24	4,200	08	$6,500	STEVE	POTEMKI
ISLANDER	BAHAMA	SLOOP	24	4,200	08	$6,500	HARVE	MANANA
PEARSON	10M	SLOOP	33	12,441	11		TOM	LONE TR
PEARSON	26	SLOOP	26	5,400	08		DICK	PURSUIT
RHODES	SWIFTSURE	SLOOP	33	14,000	10		JOHN	STRIDER

```
DTR>
```

ou can create a hierarchical relationship in two ways:

Within a record definition by using the OCCURS clause to define a repeating field. DATATRIEVE sees repeating fields as an inner record within a record. Records with repeating fields are also called hierarchical records.

Between different types of records by nesting record streams:

- Hierarchical view domains
- Inner print lists
- Nested FOR statements

Nesting record streams creates a hierarchy between nonhierarchical records. (Any records without repeating fields are nonhierarchical. They are also called "flat" records.)

Note

Both methods let you show parent records with any and all child records. Creating hierarchies between flat records by nesting record streams is preferable because you can directly modify and retrieve data using the flat records. This avoids the complexities of accessing repeating fields to modify and retrieve data, but lets you create hierarchies when you need their advantages.

---

is chapter describes:

Defining records with repeating fields

Retrieving values from repeating fields

Modifying values stored in repeating fields

Creating hierarchies from flat records by nesting record streams

1 Defining Records with Repeating Fields

epeating fields in a DATATRIEVE record definition are similar to fields defined th the OCCURS clause in COBOL and to one-dimensional arrays in BASIC.

When a record definition contains a repeating field, it means that there can be multiple occurrences of each field subordinate to the repeating field. In DATATRIEVE syntax, repeating fields are also called *lists*. Fields subordinate tc the list are called *list items*.

Retrieving data from repeating fields is not as easy as retrieving data from other types of fields. For this reason, avoid using repeating fields when defining records. You can get the display advantages of repeating fields by nesting record streams from separate flat records.

However, you may have to use repeating fields to define a record for a data file that already exists. Or, you may have to use an existing record that contains repeating fields. (See the chapter entitled Designing Better Records for information on restructuring a record with repeating fields into several flat records.)

The examples on the following pages are based on the FAMILIES domain that is part of the VAX DATATRIEVE installation kit. The parent-child relationship common to all hierarchies is illustrated literally in the record definition for FAMILIES: for each record in FAMILIES, there is one set of parents to several children.

Records without repeating fields are often called flat records because the elementary fields in them are logically equivalent to each other. When you print a flat record, all the elementary fields are displayed in the order defined by the level numbers assigned to the fields. Figure 6-1 shows this logical equivalence imposec by the level numbers in the record definition YACHT.

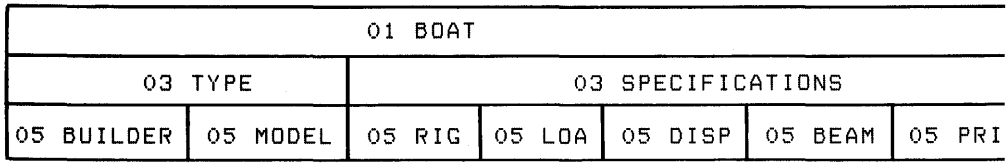


Figure 6-1: A Flat Record: YACHT

You can define a flat record for information about a family. Each record contains the names of the father and mother and the names and ages of the number of

ids you pick as the maximum number for the record. The following sample ecord definition works for families with up to two kids:

```
1 FAMILY.
  03 PARENTS.
     06 FATHER PIC X(10).
     06 MOTHER PIC X(10).
  03 KIDS.
     06 FIRST_KID.
        09 KID_NAME PIC X(10).
        09 AGE PIC 99 EDIT_STRING IS Z9.
     06 SECOND_KID.
        09 KID_NAME PIC X(10).
        09 AGE PIC 99 EDIT_STRING IS Z9.
```

When you print a record with this definition, DATATRIEVE prints the field values in the following order:

```
                          KID             KID
 FATHER      MOTHER       NAME     AGE    NAME     AGE

ERNIE       ANNE         SCOTT      2    BRIAN      0
```

The display generated by the flat record does not group the information about the kids in a way that suggests the parent-child relationship. In addition, the flat record lets you store information about two kids only. A repeating field provides a format that groups the information to convey the list items are subordinate to the other elementary fields in the record. It also lets you store information about more than two kids.

In hierarchical records, the repeating field is equivalent to the other elementary fields with the same level number, and the list items are subordinate to the list. Figure 6-2 shows this logical subordination of the list items to the list and the other elementary fields in the FAMILY record. The record stores the same type of information for each of the kids, and each field containing that information is described only once in the record definition. The record itself, however, can contain many fields of the same description -- one set for each kid.

Example 6-1 shows the actual record definition that corresponds to the logical structure illustrated in Figure 6-2.

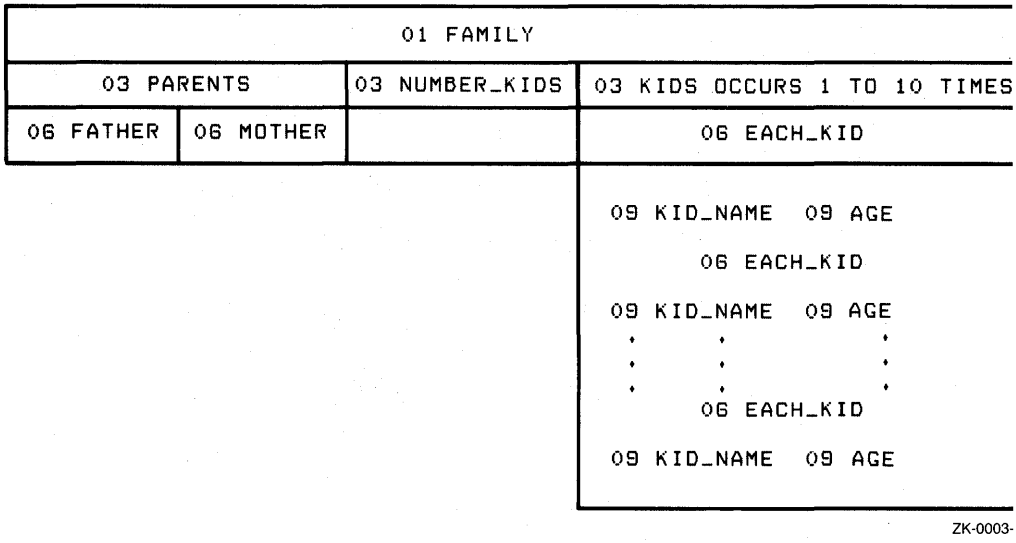


Figure 6-2: A Hierarchical Record: FAMILY_REC

The OCCURS clause in the record definition is the key to the hierarchical structure. Lists can be variable- or fixed-length, depending on the syntax of the OCCURS clause in the record definition. The list in the FAMILY record is a variable-length list: it repeats items a variable number of times according to a value stored in another record field (NUMBER_KIDS).

6.1.1 Defining Lists with a Fixed Number of Occurrences

If you define a hierarchical record with a list that occurs a fixed number of times, every record in the domain contains enough space to store the same number of list items. The OCCURS clause format for fixed-length lists is OCCURS n TIMES, where n is the number of occurrences.

You can use OCCURS n TIMES with an elementary or group field. A record definition can contain any number of OCCURS clauses in this format, any place in the record.

Using an OCCURS clause in a record definition eliminates the redundancy of defining the same fields for each kid and establishes the group field KIDS as a

st. This record definition uses a fixed-length list to provide a hierarchical structure for the information about families with two kids:

```
FIXED_LENGTH_FAMILY.
  03 PARENTS.
     06 FATHER PIC X(10).
     06 MOTHER PIC X(10).
  03 KIDS OCCURS 2 TIMES.
     06 KID_NAME PIC X(10).
     06 AGE PIC 99 EDIT_STRING IS Z9.
```

you define the record using OCCURS 2 TIMES, it is displayed in the following rmat:

```
                          KID
FATHER      MOTHER        NAME      AGE

tNIE        ANNE          SCOTT       2
                          BRIAN       0
```

his record definition causes the group field KIDS to repeat twice (OCCURS 2 IMES) in each record. Each elementary field subordinate to KIDS repeats twice.

.1.2 Defining Lists with a Variable Number of Occurrences

sing the OCCURS...DEPENDING clause in a field definition creates a hierarical record that allows a variable number of list items from one record to nother. This format lets you vary the number of list items in the records of a omain:

CCURS min TO max TIMES DEPENDING ON field-name

ou can have only one field in a record definition with an CCURS...DEPENDING clause in this format. It must appear at the end of the cord definition.

ach record in the sample FAMILIES domain contains the names of the parents, ie number of kids, and the name and age of each kid. The record definition for AMILIES uses the OCCURS...DEPENDING clause to define KIDS as a ariable-length list. The actual number of list items in a record depends on the alue of the NUMBER_KIDS field. If the value is 0, the record contains no data out kids. If the value of NUMBER_KIDS is 1, the record contains data about ie kid, and so on. Each occurrence of the KIDS field contains the group field ACH_KID. EACH_KID in turn contains two elementary fields: KID_NAME and GE. EACH_KID is a group field in the list. Like other group fields, it allows you refer to its subordinate fields with one name. (Note that DATATRIEVE does ot let you use the OCCURS field name as you would a group field name in your

statements. To DATATRIEVE, the OCCURS field name identifies what it sees a a record stream source within the record itself. For more information, see Sectior 6.2.)

Example 6-1 shows the record definition associated with FAMILIES.

Example 6-1: The FAMILY Record Definition

```
01 FAMILY.
   03 PARENTS.
      06 FATHER PIC X(10).
      06 MOTHER PIC X(10).
   03 NUMBER_KIDS PIC 99 EDIT_STRING IS Z9.
   03 KIDS OCCURS 0 TO 10 TIMES DEPENDING ON NUMBER_KIDS.
      06 EACH_KID.
         09 KID_NAME PIC X(10) QUERY_NAME IS KID.
         09 AGE PIC 99 EDIT_STRING IS Z9.
```

When you display the fields in FAMILIES, DATATRIEVE identifies the field KIDS as a list:

```
DTR> SHOW FIELDS FAMILIES
FAMILIES
   FAMILY
      PARENTS
         FATHER <Character string>
         MOTHER <Character string>
      NUMBER_KIDS       <Number>
      KIDS      <List>
         EACH_KID
            KID_NAME (KID)       <Character string>
            AGE <Number>

DTR>
```

The output of the PRINT command shows the relationship between NUMBER_KIDS and the fields KID_NAME and AGE. Example 6-2 shows all the records in the FAMILIES domain. The values of KID_NAME and AGE appear as a list in records with the number of kids greater than zero.

Example 6-2: The Hierarchical Records in FAMILIES

```
DTR> PRINT FAMILIES

                        NUMBER     KID
  FATHER      MOTHER     KIDS      NAME     AGE

JIM         ANN             2    URSULA      7
                                 RALPH       3
```

(continued on next page)

xample 6-2: The Hierarchical Records in FAMILIES (Cont.)

```
IM          LOUISE          5   ANNE        31
                                JIM         29
                                ELLEN       26
                                DAVID       24
                                ROBERT      16
)HN         JULIE           2   ANN         29
                                JEAN        26
)HN         ELLEN           1   CHRISTOPHR   0
RNIE        ANNE            2   SCOTT        2
                                BRIAN        0
HEARMAN     SARAH           1   DAVID        0
)M          ANNE            2   PATRICK      4
                                SUZIE        6
ASIL        MERIDETH        6   BEAU        28
                                BROOKS      26
                                ROBIN       24
                                JAY         22
                                WREN        17
                                JILL        20
)B          DIDI            0
EROME       RUTH            4   ERIC        32
                                CISSY       24
                                NANCY       22
                                MICHAEL     20
)M          BETTY           2   MARTHA      30
                                TOM         27
EORGE       LOIS            3   JEFF        23
                                FRED        26
                                LAURA       21
AROLD       SARAH           3   CHARLIE     31
                                HAROLD      35
                                SARAH       27
DWIN        TRINITA         2   ERIC        16
                                SCOTT       11

TR>
```

.1.3 Defining Sublists to Nest Lists Within Lists

lthough you can use only one OCCURS...DEPENDING clause in a record defi-ition, you can define any number of fixed-length lists within a variable-length st.

he sample record definition PET_REC is an extension of the FAMILY record hat illustrates sublists. The repeating field PET occurs twice for each kid, so ach kid in each family can record the data for two pets they own:

```
TR> SHOW PETS
OMAIN PETS USING PET_REC ON PET.DAT;
```

```
DTR> SHOW PET_REC
RECORD PET_REC
01 FAMILY.
   03 PARENTS.
      06 FATHER PIC X(10).
      06 MOTHER PIC X(10).
   03 NUMBER_KIDS PIC 99 EDIT_STRING IS Z9.
   03 KIDS OCCURS 0 TO 10 TIMES DEPENDING ON NUMBER_KIDS.
      06 EACH_KID.
         09 KID_NAME PIC X(10) QUERY_NAME IS KID.
         09 KID_AGE PIC 99 EDIT_STRING IS Z9.
         09 PET OCCURS 2 TIMES.
            13  PET_NAME  PIC X(10).
            13  PET_AGE   PIC 99.
;

DTR> READY PETS
DTR> PRINT FIRST 2 PETS

                        NUMBER     KID      KID     PET      PET
  FATHER      MOTHER     KIDS      NAME     AGE     NAME     AGE

JIM         LORAINE        2    GARY        24   POP         03
                                                  SODA        04
                                SUE         23   MOUSE       03
                                                  SHORTY      08
JIM         ANN            2    URSULA       7   SQUEEKY     03
                                                  FRANK       07
                                RALPH        3                00
                                                              00

DTR>
```

6.2 Retrieving Values from Repeating Fields

When you retrieve a value from a record containing a repeating field, you cannot always apply the same statements you do for other records. The following sequence of statements shows what can happen when you try to print the repeating field KIDS from the hierarchical record families:

```
DTR> READY FAMILIES
DTR> SHOW FAMILY_REC
RECORD FAMILY_REC
01 FAMILY.
   03 PARENTS.
      06 FATHER PIC X(10).
      06 MOTHER PIC X(10).
   03 NUMBER_KIDS PIC 99 EDIT_STRING IS Z9.
   03 KIDS OCCURS 0 TO 10 TIMES DEPENDING ON NUMBER_KIDS.
      06 EACH_KID.
         09 KID_NAME PIC X(10) QUERY_NAME IS KID.
         09 AGE PIC 99 EDIT_STRING IS Z9.
```

```
'R> PRINT FATHER OF FAMILIES

 FATHER

:M
:M



'R> PRINT MOTHER OF FAMILIES

 MOTHER

IN
)UISE



'R> PRINT KIDS OF FAMILIES
.INT KIDS OF FAMILIES

pected end of statement, encountered "OF".
'R>
```

ɔu can print the names of fathers and mothers successfully. But when you try to int the list field KIDS, you get an error message. If you form a collection, you n again print information on fathers and mothers but not kids:

```
'R> FIND FAMILIES
3 records found]
'R> PRINT ALL FATHER

FATHER

M
M


R> PRINT ALL MOTHER

MOTHER

N
UISE


R> PRINT ALL EACH_KID
ACH_KID" is undefined or used out of context
R> PRINT ALL KIDS
IDS" is undefined or used out of context
R> PRINT ALL KIDS OF FAMILIES
INT ALL KIDS OF FAMILIES

pected end of statement, encountered "OF".
```

In the first two examples, you get a message stating that the field name is undefined or used out of context. The third example results in the same message you got in the previous example. To retrieve the information, you can apply one o the following methods to set up a DATATRIEVE context:

- Use a FIND statement to establish a context for the list. Then use a SELECT statement to identify one record in the collection.
- Use nested FOR rse loops. The outer FOR loop forms a target stream of hierarchical records and the inner FOR loop forms a stream of list items within a hierarchical record.
- Use inner print lists (ALL print-list OF rse) to form a stream of list items within a record stream.

The following sections describe these methods for retrieving items from lists.

6.2.1 Retrieving Repeating Field Values with FIND and SELECT

You use the FIND statement to find all the records in the file that meet your specifications. Then you can use the SELECT statement to request any one of these records:

```
DTR> READY FAMILIES
DTR> FIND FAMILIES
[14 records found]
DTR> SELECT 3; PRINT

                        NUMBER     KID
  FATHER      MOTHER     KIDS      NAME      AGE

JOHN          JULIE        2     ANN          29
                                 JEAN         26
```

When you have selected a record that contains a list, you can treat the list as though it were a source of records like a domain or collection. You can continue a follows:

```
DTR> PRINT KIDS

   KID
   NAME      AGE

ANN           29
JEAN          26
```

You can also combine the FIND and SELECT statements to single out one list item. Then the context of the selected list item allows you to use the list item

ame by itself in a PRINT statement. Continue the previous example by forming collection of the KIDS list field and selecting a list item from the collection:

```
R> FIND KIDS
 records found]
R> SELECT 2; PRINT

  KID
  NAME     AGE

AN         26

R> PRINT AGE

E


R>
```

ou can use the same technique to get at nested repeating fields, such as the ET field in the hierarchical record PET_REC:

```
R> READY PETS
R> SHOW PET_REC ! Here's what the record for PETS looks like:
CORD PET_REC
 FAMILY.
 03 PARENTS.
    06 FATHER PIC X(10).
    06 MOTHER PIC X(10).
 03 NUMBER_KIDS PIC 99 EDIT_STRING IS Z9.
 03 KIDS OCCURS 0 TO 10 TIMES DEPENDING ON NUMBER_KIDS.
    06 EACH_KID.
       09 KID_NAME PIC X(10) QUERY_NAME IS KID.
       09 KID_AGE PIC 99 EDIT_STRING IS Z9.
       09 PET OCCURS 2 TIMES.
          13  PET_NAME  PIC X(10).
          13  PET_AGE   PIC 99.


R> ! First, form a collection of the records in the PETS domain:
R> FIND PETS
 records found]
R> SELECT 3; PRINT

                      NUMBER     KID      KID     PET      PET
FATHER      MOTHER     KIDS      NAME     AGE     NAME     AGE

M          LOUISE        5    ANNE        31   FRANK       14
                                                FRANK       14
                              JIM          29                00
                                                             00
                              ELLEN        26                00
                                                             00
                              DAVID        24                00
                                                             00
                              ROBERT       16                00
                                                             00
```

```
DTR> ! Second, form a collection of the "records" in the
DTR> ! KIDS repeating field:
DTR> FIND KIDS
[5 records found]
DTR> SELECT 1; PRINT

   KID      KID     PET      PET
   NAME     AGE     NAME     AGE

ANNE        31   FRANK       14
                 FRANK       14

DTR> ! Third, form a collection of the "records"
DTR> ! in the PET repeating field:
DTR> FIND PET
[2 records found]
DTR> ! Finally, you can print a field subordinate to
DTR> ! the nested repeating field PET:
DTR> SELECT 1; PRINT PET_AGE

PET
AGE

14
```

You cannot retrieve the value of repeating fields from more than one record using only FIND and SELECT statements.

6.2.2 Retrieving Repeating Field Values with Nested FOR Loops

To retrieve values from list items by nesting FOR loops, start from the top of the hierarchy and work toward the list items you want to retrieve. In the following example, the source for the RSE in the first or outer FOR loop is the hierarchical domain FAMILIES. The source in the second loop is the list item KIDS:

```
DTR> FOR FAMILIES
[Looking for statement]
CON> FOR KIDS WITH AGE < 10
[Looking for statement]
CON> PRINT KID_NAME

   KID
   NAME

URSULA
RALPH
CHRISTOPHR
SCOTT
BRIAN
DAVID
PATRICK
SUZIE

DTR>
```

ıe FOR statement preceding the PRINT statement in the following example ɔps through all the records in FAMILIES. For each of those records, the RSE the PRINT statement retrieves only the first kid whose age is less than 10:

```
R> FOR FAMILIES
ooking for statement]
N> PRINT KID_NAME OF FIRST 1 KIDS WITH AGE < 10

 KID
 NAME

SULA
RISTOPHR
ɔTT
VID
TRICK

R>
```

ıe OF rse clause in the PRINT statement serves the same purpose as a nested)R rse statement. The inner RSE (FIRST 1 KIDS WITH AGE < 10) identifies ms from the list field KIDS that are included within a FAMILIES record iden- ied by the outer FOR rse statement.

ıe equivalent statement using nested FOR rse statements is:

```
R FAMILIES FOR FIRST 1 KIDS WITH AGE < 10 PRINT KID_NAME
```

r nested repeating fields, use the same technique, but nest FOR statements ɔre than one level. The following example uses the hierarchical domain PETS as ɜ record source for the outer FOR loop. The repeating field KIDS is the source · the second FOR loop, and the nested repeating field PET is the source for the ıermost FOR loop. The example prints the MOTHER and KID_NAME fields to ɔw which PET record and KIDS occurrence the PET occurrence comes from:

```
t> FOR PETS WITH ANY KIDS
ı>     BEGIN
ı>     PRINT  MOTHER
ı>     FOR KIDS WITH ANY PET
ı>        BEGIN
ı>        PRINT COL 10, KID_NAME
ı>        FOR PET WITH PET_AGE GT 2
ı>             PRINT COL 20, PET_NAME, PET_AGE
ı>        END
ı>     END
```

(continued on next page)

```
  MOTHER

LORAINE

            KID
            NAME

          GARY

                              PET       PET
                              NAME      AGE

                          POP           03
                          SODA          04
          SUE
                          MOUSE         03
                          SHORTY        08
ANN
          URSULA
                          SQUEEKY       03
                          FRANK         07
          RALPH
LOUISE
          ANNE
                          FRANK         14
                          FRANK         14
          JIM
          ELLEN
          DAVID
          ROBERT

DTR>
```

6.2.3 Retrieving Repeating Field Values with Inner Print Lists

The simplest way to print a repeating field is to print the entire record containin the repeating field:

```
DTR> READY FAMILIES
DTR> PRINT FIRST 1 FAMILIES

                          NUMBER      KID
  FATHER       MOTHER      KIDS       NAME      AGE

JIM          ANN              2     URSULA       7
                                    RALPH        3

DTR>
```

To print selected fields from the record, you must specify a print list in the PRINT statement. (Print lists consist of field names or other value expressions and modifiers.) To specify a list item in a print list, you must use an *inner print list*, which has the format:

ALL print-list OF rse

ı the print-list clause of the inner print list, you include the list items you want
› display. The OF rse clause of the inner print list creates a context for the item
. the hierarchical list. Example 6-3 prints the name of the mother and informa-
on about her children for the first FAMILIES record.

ĸample 6-3: PRINT Statement with Inner Print List

```
'R> !
'R> !      ------------------------------
'R> !      |            print-list       |
'R> !      |          ------------------ |
'R> !      |        |    Inner Print List ||
'R> !      |        |     ------------   ||
'R> !      |        |    |print-list |    ||
'R> !      |        |    |           |    ||
'R> PRINT MOTHER, ALL KID_NAME, AGE OF KIDS OF FIRST 1 FAMILIES

             KID
MOTHER       NAME     AGE

IN           URSULA     7    !All kids from first family
             RALPH      3
```

ı this example, ALL KID_NAME, AGE OF KIDS is an inner print list. It is also
ı element of the outer print list that includes the field MOTHER as another ele-
ent. This outer print list is associated with the target record stream formed by
ıe OF FIRST 1 FAMILIES clause.

he syntax of the type of PRINT statement that includes an inner print list is:

RINT print-list, ALL print-list OF rse-1 [,print-list] OF rse-2

ı this syntax, ALL print-list OF rse-1 is the inner print list. The argument rse-1
eates a record stream from occurrences of a repeating field. That inner record
ream is itself within the record stream formed by rse-2.

the inner print list is the first element in the outer print list, you must precede
ıe inner print list with another mandatory keyword, ALL. The following example
similar to the previous one. However, it displays information about children in
.e first FAMILIES record first, then prints the mother's name:

```
'R> PRINT ALL ALL KID_NAME, AGE OF KIDS, MOTHER OF FIRST 1 FAMILIES

 KID
 NAME     AGE    MOTHER

.SULA       7   ANN
.LPH        4

'R>
```

The syntax of this type of PRINT statement that includes an inner print list as the first element of the outer print list is:

PRINT ALL ALL print-list OF rse-1 [,print-list] OF rse-2

There is only one difference between this syntax diagram and the previous one: you need an extra ALL when the first print list element in the outer print list is an inner print list.

There are two important points to remember when working with inner print lists.

- To DATATRIEVE, an inner print list is just another print-list element in th outer print list.
- An inner print list establishes context for items in a list.

While inner print lists can complicate statements, they allow you to control completely how DATATRIEVE displays repeating fields. By using the repeating field as the source for an RSE in an inner print list, you can specify which occurrences of the repeating field DATATRIEVE displays. The next example shows the results of limiting one or both of the RSEs in a PRINT statement with an inner print list:

```
DTR> ! Limit the RSE for the inner print list
DTR> ! to the first occurrence of KIDS from every family:
DTR> PRINT MOTHER,                  !Print list for rse-2
CON>     ALL KID_NAME, AGE -        !Print list for rse-1
CON>     OF FIRST 1 KIDS -          !rse-1, uses KIDS as record source
CON> OF FAMILIES                    !rse-2, uses FAMILIES as record source

                KID
  MOTHER        NAME      AGE

ANN             URSULA      7         !First kid from every family
LOUISE          ANNE       31
JULIE           ANN        29
ELLEN           CHRISTOPHR  0
ANNE            SCOTT       2
SARAH           DAVID       0
ANNE            PATRICK     4
MERIDETH        BEAU       28
DIDI
RUTH            ERIC       32
BETTY           MARTHA     30
LOIS            JEFF       23
SARAH           CHARLIE    31
TRINITA         ERIC       16
```

```
R> ! Limit both RSEs to print the first occurrence
R> ! of KIDS in the first FAMILIES record:
R> PRINT MOTHER,                    !Print list for rse-2
N>     ALL KID_NAME, AGE -          !Print list for rse-1
N>     OF FIRST 1 KIDS -            !rse-1, uses KIDS as record source
N> OF FIRST 1 FAMILIES              !rse-2, uses FAMILIES as record source

             KID
MOTHER       NAME     AGE

N         URSULA       7          !First kid of first family

R>
```

st as with FOR loops, you can nest inner print lists to retrieve desired informa-
ın from nested repeating fields.

ıe following example, like the previous example, retrieves only the information
ɔm the first occurrence of the repeating field KIDS from a single record. It uses
e PETS domain, however, and nests a third print list to display information
ɔm the nested repeating field PET:

```
R> PRINT MOTHER,                              !Print list for rse-3
N>     ALL KID_NAME, KID_AGE,                 !Print list for rse-2
N>             ALL PET_NAME, PET_AGE -        !Print list for rse-1
N>             OF FIRST 1 PET -               !rse-1, uses PET as record source
N>     OF FIRST 1 KIDS -                      !rse-2, uses KIDS as record source
N> OF PETS WITH MOTHER = "ANN"                !rse-3, uses PETS as record source

             KID      KID     PET      PET
MOTHER       NAME     AGE     NAME     AGE

N         URSULA       7   SQUEEKY     03    !First pet of first kid of
                                               !family whose mother is Ann


R>
```

ɜing nested inner print lists may require nesting the keyword ALL as well. If
e inner print list is the first element in the outermost print list, you must pre-
de it with as many ALL keywords as there are OF RSE phrases in the print
ɜtement.

The following example prints only the names of pets for the first two records in the PETS domain and requires three ALL keywords:

```
DTR> PRINT ALL -                    !Print list for rse-3
CON>             ALL                !Print list for rse-2
CON>               ALL PET_NAME     !Print list for rse-1
CON>               OF PET -         !rse-1, uses PET as record source
CON>             OF KIDS -          !rse-2, uses KIDS as record source
CON> OF FIRST 2 PETS                !rse-3, uses PETS as record source

   PET
   NAME

POP
SODA
MOUSE
SHORTY
SQUEEKY
FRANK


DTR>
```

6.2.4 Retrieving Repeating Field Values with the Context Searcher

You can save yourself the difficulty of typing complex inner print lists when deal ing with lists and sublists. The VAX DATATRIEVE Context Searcher helps you get access to list items. It constructs inner print lists for you once you establish a single record context for it to work on. When you use the name of a list or sublis item (even sublist items at the sixth level of a hierarchical record), it searches through the names of list items, constructing the inner print lists needed to retrieve the value.

You activate the Context Searcher with the SET SEARCH command. When you invoke DATATRIEVE, SET NO SEARCH is in effect unless you have a SET SEARCH command in your DTR$STARTUP file.

The following example shows how the Context Searcher simplifies some of the previous examples that used inner print lists:

```
DTR> SET SEARCH
DTR> READY FAMILIES
DTR> ! Compare with results from
DTR> ! PRINT MOTHER, ALL KID_NAME OF KIDS, FATHER OF FIRST 1 FAMILIES
DTR> PRINT MOTHER, KID_NAME, FATHER OF FIRST 1 FAMILIES
Not enough context. Some field names resolved by Context Searcher.

              KID
  MOTHER      NAME       FATHER

ANN         URSULA       JIM
            RALPH        JIM
```

```
[R> ! Compare with results from
[R> ! PRINT MOTHER, EACH_KID OF KIDS OF FIRST 1 FAMILIES
[R> PRINT MOTHER, EACH_KID OF FIRST 1 FAMILIES
›t enough context. Some field names resolved by Context Searcher.

                KID
 MOTHER         NAME     AGE

JN           URSULA       7
             RALPH        3

[R> ! Compare with results from
[R> ! PRINT ALL ALL EACH_KID OF KIDS OF
[R> ! FIRST 1 FAMILIES WITH NUMBER_KIDS = 3
[R> PRINT EACH_KID OF FIRST 1 FAMILIES WITH NUMBER_KIDS = 3
›t enough context. Some field names resolved by Context Searcher.

  KID
  NAME      AGE

:FF         23
ŁED         26
\URA        21

[R> ! Compare with results from
[R> ! PRINT ALL ALL ALL PET_NAME OF PET OF KIDS OF FIRST 2 PETS
[R> PRINT PET_NAME OF FIRST 2 PETS
›t enough context. Some field names resolved by Context Searcher.

  PET
  NAME

)P
)DA
)USE
IORTY
)UEEKY
ŁANK

[R>
```

2.5 Retrieving Repeating Field Values by Flattening Hierarchies

nother way to simplify retrieving values from repeating fields is to "flatten" the ierarchical structure of the record. To flatten a hierarchy means to repeat all elds in the record for each occurrence of the repeating field.

lattening the hierarchical domain FAMILIES would mean repeating the ATHER, MOTHER, NUMBER_KIDS, and the entire list within KIDS fields for ıch occurrence of the KIDS repeating field. The next two examples compare how ıe first two records of FAMILIES look when first displayed normally and then attened.

Normal display:

FATHER	MOTHER	NUMBER KIDS	KID NAME	AGE
JIM	ANN	2	URSULA	7
			RALPH	3
JIM	LOUISE	5	ANNE	31
			JIM	29
			ELLEN	26
			DAVID	24
			ROBERT	16

Flattened display:

FATHER	MOTHER	NUMBER KIDS	KID NAME	AGE	KID NAME	AGE
JIM	ANN	2	URSULA	7	URSULA	7
			RALPH	3		
JIM	ANN	2	URSULA	7	RALPH	3
			RALPH	3		
JIM	LOUISE	5	ANNE	31	ANNE	31
			JIM	29		
			ELLEN	26		
			DAVID	24		
			ROBERT	16		
JIM	LOUISE	5	ANNE	31	JIM	29
			JIM	29		
			ELLEN	26		
			DAVID	24		
			ROBERT	16		
JIM	LOUISE	5	ANNE	31	ELLEN	26
			JIM	29		
			ELLEN	26		
			DAVID	24		
			ROBERT	16		
JIM	LOUISE	5	ANNE	31	DAVID	24
			JIM	29		
			ELLEN	26		
			DAVID	24		
			ROBERT	16		
JIM	LOUISE	5	ANNE	31	ROBERT	16
			JIM	29		
			ELLEN	26		
			DAVID	24		
			ROBERT	16		

All the fields repeat, including the entire KIDS list, for each occurrence of the repeating fields KIDS. The repetition of the KIDS list in the flattened display makes it cumbersome and hard to read. For a more readable display, you can lim the fields to only those you want to see (see the next sections).

ou can flatten hierarchies in three different ways to achieve the same results:

With the CROSS clause

With inner print lists

With nested FOR loops

he next sections discuss these methods.

2.5.1 Using the CROSS Clause to Flatten Hierarchies -- To create the flat-ned display in the previous example, use a PRINT statement with the CROSS ause.

```
'R> PRINT FAMILIES CROSS KIDS

                        NUMBER     KID                KID
FATHER      MOTHER       KIDS      NAME      AGE      NAME      AGE

'M          ANN             2      URSULA      7      URSULA      7
                                   RALPH       3
'M          ANN             2      URSULA      7      RALPH       3
                                   RALPH       3
                                     .
                                     .
                                     .
)WIN        TRINITA         2      ERIC       16      ERIC       16
                                   SCOTT      11
)WIN        TRINITA         2      ERIC       16      SCOTT      11
                                   SCOTT      11
```

ATATRIEVE treats KIDS as a domain in this statement. For each "record" in e KIDS "domain," DATATRIEVE prints the corresponding record from the AMILIES domain (including the list field KIDS in those records) and the KIDS ecord."

ou can limit the flattened FAMILIES records displayed by the CROSS clause ' using the same techniques you use with two separate domains. DATATRIEVE ins the appropriate KIDS "records" with the corresponding FAMILIES record.

mit the display to joining FAMILIES to first two records of the KIDS lomain":

```
R> PRINT FIRST 2 FAMILIES CROSS KIDS  ! FIRST 2 in this
                                      ! statement refers to the KIDS
                                      ! "domain," not FAMILIES.

                        NUMBER     KID                KID
FATHER      MOTHER       KIDS      NAME      AGE      NAME      AGE

M           ANN             2      URSULA      7      URSULA      7
                                   RALPH       3
M           ANN             2      URSULA      7      RALPH       3
                                   RALPH       3
```

Limit the display to joining FAMILIES with the KIDS "record" containing "URSULA":

```
DTR> PRINT FAMILIES CROSS KIDS WITH KID_NAME CONTAINING "URSULA"

                        NUMBER     KID              KID
  FATHER       MOTHER    KIDS      NAME     AGE     NAME     AGE

JIM           ANN          2     URSULA       7   URSULA       7
                             2     RALPH        3

DTR>
```

The preceding displays included the KIDS repeating field and all the list items it contained. To keep from seeing the entire KIDS list for each KIDS "record" displayed, specify only the fields you want displayed in the PRINT statement.

```
DTR> PRINT FATHER, MOTHER, NUMBER_KIDS, KID_NAME, AGE OF FAMILIES CROSS

                        NUMBER     KID
  FATHER       MOTHER    KIDS      NAME     AGE

JIM           ANN          2     URSULA       7
JIM           ANN          2     RALPH        3
                           .
                           .
                           .
EDWIN         TRINITA      2     ERIC        16
EDWIN         TRINITA      2     SCOTT       11

DTR>
```

You can nest CROSS clauses to retrieve "records" from nested repeating fields. The following statement uses the PETS domain, which has the nested repeating field PET within the repeating field KIDS. It prints the first 4 "records" in the PET "domain" joined with KIDS "domain," which is itself joined with the PETS domain. The statement prints only the elementary fields of the flattened PETS record, omitting the list fields.

```
DTR> PRINT FATHER, MOTHER, NUMBER_KIDS, -
CON>       KID_NAME, KID_AGE, PET_NAME, PET_AGE
CON> OF FIRST 4 PETS CROSS KIDS CROSS PET

                        NUMBER     KID      KID     PET      PET
  FATHER       MOTHER    KIDS      NAME     AGE     NAME     AGE

JIM           LORAINE      2     GARY        24   POP         03
JIM           LORAINE      2     GARY        24   SODA        04
JIM           LORAINE      2     SUE         23   MOUSE       03
JIM           LORAINE      2     SUE         23   SHORTY      08

DTR>
```

: you often need to retrieve values in a repeating field of the same domain, you ın set up a view domain that contains the flattened records. For instance, you ɔuld define a view, FLAT_FAMILY_VIEW, that uses a CROSS clause to flatten ıe FAMILIES records:

```
ΓR> SHOW FLAT_FAMILY_VIEW
ɔMAIN FLAT_FAMILY_VIEW
        OF FAMILIES USING
1 FLAT_FAMILY OCCURS FOR FAMILIES CROSS KIDS.
        03 FATHER FROM FAMILIES.
        03 MOTHER FROM FAMILIES.
        03 NUMBER_KIDS FROM FAMILIES.
        03 KID_NAME FROM FAMILIES.
        03 AGE FROM FAMILIES.


ΓR>
```

ou can then use simple PRINT statements to retrieve the repeating field values ou need:

```
ΓR> READY FLAT_FAMILY_VIEW
ΓR> PRINT FIRST 2 FLAT_FAMILY_VIEW

                      NUMBER     KID
 FATHER      MOTHER    KIDS      NAME     AGE

IM          ANN           2    URSULA       7
IM          ANN           2    RALPH        3

ΓR> PRINT FLAT_FAMILY_VIEW WITH AGE GT 30

                      NUMBER     KID
 FATHER      MOTHER    KIDS      NAME     AGE

IM          LOUISE        5    ANNE        31
ERŌME       RUTH          4    ERIC        32
ARŌLD       SARAH         3    CHARLIE     31
ARŌLD       SARAH         3    HAROLD      35

ΓR>
```

.2.5.2 Using Inner Print Lists to Flatten Hierarchies -- For any PRINT state-ıent you use with the CROSS clause, there is an equivalent PRINT statement sing inner print lists that produces the same results. The following PRINT

statements show the inner print lists that duplicate the results of examples in the previous section:

```
DTR> ! Duplicate the PRINT FAMILIES CROSS KIDS statement:
DTR> PRINT ALL ALL FAMILY, EACH_KID OF KIDS OF FAMILIES

                          NUMBER     KID                KID
  FATHER       MOTHER     KIDS       NAME      AGE      NAME      AGE

JIM            ANN          2      URSULA        7    URSULA        7
                                   RALPH         3
JIM            ANN          2      URSULA        7    RALPH         3
                                   RALPH         3
                                     .
                                     .
                                     .
EDWIN          TRINITA      2      ERIC         16    ERIC         16
                                   SCOTT        11
EDWIN          TRINITA      2      ERIC         16    SCOTT        11
                                   SCOTT        11

DTR> ! Duplicate the PRINT FIRST 2 FAMILIES CROSS KIDS statement:
DTR> PRINT ALL ALL FAMILY, EACH_KID OF FIRST 2 KIDS OF FIRST 1 FAMILIES

                          NUMBER     KID                KID
  FATHER       MOTHER     KIDS       NAME      AGE      NAME      AGE

JIM            ANN          2      URSULA        7    URSULA        7
                                   RALPH         3
JIM            ANN          2      URSULA        7    RALPH         3
                                   RALPH         3

DTR> ! Duplicate the PRINT FAMILIES CROSS KIDS WITH
DTR> !   KID_NAME CONTAINING "URSULA" statement:
DTR> PRINT ALL ALL FAMILY, EACH_KID -
CON> OF KIDS WITH KID_NAME CONTAINING "URSULA" OF FAMILIES

                          NUMBER     KID                KID
  FATHER       MOTHER     KIDS       NAME      AGE      NAME      AGE

JIM            ANN          2      URSULA        7    URSULA        7
                                   RALPH         3
```

```
'R> ! Duplicate the PRINT FATHER, MOTHER, NUMBER_KIDS,
'R> !   KID_NAME, AGE OF FAMILIES CROSS KIDS statement:
'R> PRINT ALL ALL FATHER, MOTHER, NUMBER_KIDS, KID_NAME, AGE -
)N> OF KIDS OF FAMILIES

                      NUMBER     KID
FATHER      MOTHER     KIDS      NAME      AGE

M           ANN           2    URSULA       7
M           ANN           2    RALPH        3
                          .
                          .
                          .

)WIN        TRINITA       2    ERIC        16
)WIN        TRINITA       2    SCOTT       11

'R> ! Duplicate the PRINT FATHER, MOTHER, NUMBER_KIDS, KID_NAME, KID_AGE,
'R> !   PET_NAME, PET_AGE OF FIRST 4 PETS CROSS KIDS CROSS PET statement
'R> PRINT ALL ALL ALL FATHER, MOTHER, NUMBER_KIDS, KID_NAME, KID_AGE, -
)N> PET_NAME, PET_AGE OF FIRST 4 PET OF KIDS OF FIRST 1 PETS

                      NUMBER     KID      KID     PET      PET
FATHER      MOTHER     KIDS      NAME     AGE     NAME     AGE

M           LORAINE       2    GARY        24   POP         03
M           LORAINE       2    GARY        24   SODA        04
M           LORAINE       2    SUE         23   MOUSE       03
M           LORAINE       2    SUE         23   SHORTY      08

'R>
```

2.5.3 Using Nested FOR Statements to Flatten Hierarchies -- For any RINT statement you use with the CROSS clause, there are equivalent nested)R statements that produce the same results. The following nested FOR statements duplicate the results of CROSS statements in the previous section:

```
'R> ! Duplicate the PRINT FAMILIES CROSS KIDS statement:
'R> FOR FAMILIES FOR KIDS PRINT FAMILY, EACH_KID

                      NUMBER     KID                KID
FATHER      MOTHER     KIDS      NAME      AGE      NAME      AGE

M           ANN           2    URSULA       7    URSULA       7
                               RALPH        3
M           ANN           2    URSULA       7    RALPH        3
                               RALPH        3
                                 .
                                 .
                                 .

)WIN        TRINITA       2    ERIC        16    ERIC        16
                               SCOTT       11
)WIN        TRINITA       2    ERIC        16    SCOTT       11
                               SCOTT       11
```

```
DTR> ! Duplicate the PRINT FIRST 2 FAMILIES CROSS KIDS statement:
DTR> FOR FIRST 1 FAMILIES FOR FIRST 2 KIDS PRINT FAMILY, EACH_KID

                        NUMBER     KID              KID
  FATHER      MOTHER    KIDS       NAME     AGE     NAME     AGE

JIM           ANN         2        URSULA     7     URSULA     7
                           2        RALPH      3
JIM           ANN         2        URSULA     7     RALPH      3
                           2        RALPH      3

DTR> ! Duplicate the PRINT FAMILIES CROSS KIDS WITH
DTR> !                KID_NAME CONTAINING "URSULA" statement:
DTR> FOR FAMILIES FOR KIDS WITH KID_NAME CONTAINING "URSULA" -
CON> PRINT FAMILY, EACH_KID

                        NUMBER     KID              KID
  FATHER      MOTHER    KIDS       NAME     AGE     NAME     AGE

JIM           ANN         2        URSULA     7     URSULA     7
                           2        RALPH      3

DTR> ! Duplicate the PRINT FATHER, MOTHER, NUMBER_KIDS, KID_NAME, AGE
DTR> !                OF FAMILIES CROSS KIDS statement:
DTR> FOR FAMILIES FOR KIDS PRINT FATHER, MOTHER, NUMBER_KIDS, KID_NAME,

                        NUMBER     KID
  FATHER      MOTHER    KIDS       NAME     AGE

JIM           ANN         2        URSULA     7
JIM           ANN         2        RALPH      3
                          .
                          .
                          .
EDWIN         TRINITA     2        ERIC      16
EDWIN         TRINITA     2        SCOTT     11

DTR> ! Duplicate the
DTR> ! PRINT FATHER, MOTHER, NUMBER_KIDS,
DTR> !       KID_NAME, KID_AGE, PET_NAME, PET_AGE -
DTR> ! OF FIRST 4 PETS CROSS KIDS CROSS PET statement
DTR> FOR FIRST 1 PETS FOR KIDS FOR FIRST 4 PET -
CON> PRINT FATHER, MOTHER, NUMBER_KIDS,
CON>       KID_NAME, KID_AGE, PET_NAME, PET_AGE

                        NUMBER     KID      KID     PET      PET
  FATHER      MOTHER    KIDS       NAME     AGE     NAME     AGE

JIM           LORAINE     2        GARY      24     POP      03
JIM           LORAINE     2        GARY      24     SODA     04
JIM           LORAINE     2        SUE       23     MOUSE    03
JIM           LORAINE     2        SUE       23     SHORTY   08

DTR>
```

.3 Modifying Values Stored in Repeating Fields

he techniques used to retrieve data from repeating fields can be adapted for ıodifying data. This section shows two methods of modifying data stored in ɜpeating fields:

Use FIND and SELECT statements to establish context, and then use the MODIFY statement.

Use FOR statements in combination with the MODIFY statement to establish context with nested record streams.

his section also describes how to change the length of a variable-length list (a ɜpeating field defined with the OCCURS DEPENDING clause). For more infor-ıation about using the MODIFY statement, see Chapter 4 in this manual and ıe *VAX DATATRIEVE Handbook*.

.3.1 Modifying Repeating Field Values with FIND and SELECT

/hen you try to change the values stored in repeating fields, you encounter the ıme complications that occur when retrieving data from repeating fields.

or instance, you cannot directly modify a field subordinate to a repeating field. nce you have selected a record that contains a repeating field, follow these steps:

Use the FIND statement to create a collection of the occurrences of the repeating field.

Use the SELECT statement to single out one of those occurrences.

Use the MODIFY statement to change the value of the desired field of the occurrence you selected.

he following example uses this method. It modifies the AGE field in the repeat-ɡ field KIDS in the FAMILIES domain.

```
:R> ! Create a named collection from FAMILIES domain:
:R> FIND FIRST 1 FAM IN FAMILIES
. record found]
:R> PRINT ALL

                        NUMBER     KID
 FATHER      MOTHER      KIDS      NAME      AGE

:M          ANN             2    URSULA       8
                                 RALPH        3

:R> ! Select a record from the named collection:
:R> SELECT
```

```
DTR> MODIFY AGE  ! Won't work because AGE is subordinate to KIDS list fi
"AGE" is undefined or used out of context.
DTR> ! So, create another collection from the list field KIDS:
DTR> FIND KIDS
[2 records found]
DTR> ! Now select an occurrence of the list field,
DTR> ! in this case the second:
DTR> SELECT 2
DTR> PRINT

   KID
   NAME     AGE

RALPH         3

DTR> MODIFY AGE  ! Now we can modify the AGE field
Enter AGE: 4
DTR> ! Check to see that the field was really modified:
DTR> PRINT

   KID
   NAME     AGE

RALPH         4

DTR> RELEASE CURRENT
DTR> PRINT

                        NUMBER     KID
  FATHER      MOTHER     KIDS      NAME      AGE

JIM           ANN            2   URSULA        8
                                 RALPH         4

DTR>
```

Note that when you modify a selected record, the field you specify following MODIFY can never be the OCCURS field itself. In the preceding example, this means that you cannot enter MODIFY KIDS. If you want to modify all the fields in each occurrence of the repeating field, you can enter the name of a top-level group field subordinate to the OCCURS field (not all record definitions contain such a field), or you can specify all the elementary fields subordinate to the OCCURS field. In the context of the preceding example, this means that you can enter MODIFY EACH_KID (group field) or MODIFY KID_NAME, AGE (list of elementary fields) in order to enter a value for each elementary field in the list occurrence.

If you want to change the values of *all* occurrences of fields subordinate to a repeating field, you can add the keyword OF, followed by the name of the repeating field. Use this general format:

MODIFY [ALL] list-item OF list

)ATATRIEVE prompts you to enter a value for the field you specify following
ıe MODIFY statement or for each of its elementary items if you specify a group
eld.

lote that this format differs from the preceding one by including the OF list
.ause. When you include this clause, you modify all occurrences in the list at
nce. There are likely to be few times when you want to do that. This format can
e useful, however, when you want to modify items in a variable length list (one
ith an OCCURS...DEPENDING ON clause) and one of the following conditions
true:

There is more than one occurrence stored in the list and you want *all* occurrences to contain the same value for the field or fields you modify.

There is only one occurrence stored in the list.

he following examples illustrate each of these conditions.

```
[R> FIND FAMILIES WITH FATHER = "ARNIE"
l record found]
[R> PRINT
› record selected, printing whole collection.

                        NUMBER     KID
 FATHER      MOTHER      KIDS      NAME     AGE

ıNIE        ANNE            2    SCOTT        2
                                 BRIAN        0

:R> ! Oops... Scott and Brian are twenty-year-old twins!
:R> SELECT
:R> MODIFY AGE
ıGE" is undefined or used out of context.
:R> MODIFY AGE OF KIDS
ıter AGE: 20
:R> PRINT

                        NUMBER     KID
 FATHER      MOTHER      KIDS      NAME     AGE

:NIE        ANNE            2    SCOTT       20
                                 BRIAN       20

'R> FIND FAMILIES WITH FATHER = "JOHN"
! records found]
```

```
DTR> PRINT
No record selected, printing whole collection.

                        NUMBER     KID
  FATHER      MOTHER     KIDS      NAME      AGE

JOHN         JULIE          2    ANN          29
                                 JEAN         26
JOHN         ELLEN          1    CHRISTOPHR    0

DTR> SELECT 2
DTR> PRINT

                        NUMBER     KID
  FATHER      MOTHER     KIDS      NAME      AGE

JOHN         ELLEN          1    CHRISTOPHR    0

DTR> MODIFY AGE
"AGE" is undefined or used out of context.
DTR> MODIFY AGE OF KIDS
Enter AGE: 1
DTR> PRINT

                        NUMBER     KID
  FATHER      MOTHER     KIDS      NAME      AGE

JOHN         ELLEN          1    CHRISTOPHR    1

DTR>
```

6.3.2 Modifying Repeating Field Values with FOR and MODIFY Statements

You can modify values stored in repeating fields without using collections by nes ing record streams with the FOR and MODIFY statements.

Remember that you can treat the repeating field, or list, as a source of records like a domain or collection. Two formats for nesting record streams based on repeating fields have different results in modifying values:

FOR rse MODIFY list-rse USING assignment-statement

In this format, list-rse is a record selection expression that uses the repeating field as the record source. The first RSE specifies the record source and specific records to be modified. The list-rse specifies the repeating field and the particula occurrences in the list to be modified.

You supply only one value for each field you specify in the USING clause. If the list-rse specifies more than one occurrence in the list, each field value you supply applies to them all.

'he following example modifies the name of one child in the FAMILIES domain:

```
TR> ! Use SET NO PROMPT to turn off the "[Looking for...]" prompts
TR> SET NO PROMPT
TR> ! The outer RSE specifies a single record
TR> ! from the FAMILIES domain:
TR> FOR FAMILIES WITH FATHER = "TOM" AND MOTHER = "ANNE"
)N>     ! The inner RSE within the MODIFY statement uses the
)N>     ! repeating field KIDS as a record source and
)N>     ! specifies a single occurrence of KIDS.  Had it
)N>     ! specified more occurrences, they all would be
)N>     ! modified with the value specified in the USING clause:
)N>     MODIFY KIDS WITH KID_NAME = "PATRICIA" USING
)N>       BEGIN
)N>         ! Print the occurrence of KIDS specified:
)N>         PRINT
)N>         ! Change the value of the subordinate field KID_NAME:
)N>         KID_NAME = "PATRICK"
)N>         ! Print the modified occurence of KIDS:
)N>         PRINT
)N>       END

  KID
  NAME     AGE

\TRICIA     4

  KID
  NAME     AGE

\TRICK      4

TR>
```

'ith this format, you can modify only a single occurrence of a repeating field or ve all occurrences specified in the list-rse the same value. The next format ıows how to process independently more than one occurrence in the same :atement.

)R rse FOR list-rse MODIFY [field-name [,...]]

se this format to independently process more than one occurrence of a repeating :ld in the same statement. When you use this format, DATATRIEVE prompts ıu to enter field values for as many times as there are occurrences of the repeat-g field.

you do not specify field names, DATATRIEVE prompts you to enter values for l fields subordinate to the repeating field.

The following example uses this format to change the value of all the occurrences of the KIDS repeating field in the first FAMILIES record:

```
DTR> FOR FIRST 1 FAMILIES          ! The RSE in the outer FOR statement
CON>                               ! specifies a single record in
CON>                               ! FAMILIES.  If it specified more,
CON>                               ! DATATRIEVE would prompt for
CON>                               ! values for repeating fields in each
CON>                               ! record.
CON>
CON>    FOR KIDS                   ! The inner FOR statement specifies
CON>                               ! all occurrences of KIDS in the
CON>                               ! record or records in the outer
CON>                               ! FOR statement.  It could have
CON>                               ! limited the RSE to a single
CON>                               ! occurrence of the repeating field.
CON>          BEGIN
CON>            PRINT
CON>            MODIFY AGE         ! The MODIFY statement specifes that
CON>                               ! only the AGE field subordinate to
CON>                               ! the KIDS repeating field will be
CON>                               ! changed.
CON>            PRINT
CON>          END

   KID
   NAME     AGE

URSULA       7
Enter AGE: 8

   KID
   NAME     AGE

URSULA       8
RALPH        3
Enter AGE: 4
RALPH        4

DTR>
```

6.3.3 Changing the Length of a Variable-Length List

If you define a repeating field with the OCCURS DEPENDING clause, you may be able to change the number of list items (the number of times a repeating field repeats), depending on how you define the data file for the domain:

- The most restrictive case is a data file that you define without the MAX or KEY clauses. This creates a sequential file with variable-length records. In such a file, you can change only the number of list items up to the value you first store in the field referred to in the OCCURS DEPENDING clause. You cannot exceed that number because DATATRIEVE determines the length of each record when you first store it.

For FAMILIES, which uses a sequential file with variable-length records, this means you cannot increase the value specified for NUMBER_KIDS above that entered when the record was first stored:

```
DTR> FIND FIRST 1 FAMILIES; SELECT
DTR> PRINT

                        NUMBER     KID
  FATHER      MOTHER     KIDS      NAME     AGE

JIM          ANN           2     URSULA      8
                                 RALPH       4

DTR> MODIFY NUMBER_KIDS
Enter NUMBER_KIDS: 3
Error using RMS file "DTR$LIBRARY:FAMILY.DAT".
%RMS-F-RSZ, invalid record size
DTR>
```

Because you cannot increase NUMBER_KIDS, you cannot add information on new children to a FAMILIES record.

If you specify the MAX clause when defining the data file (whether the file is indexed or sequential) you create a file with fixed-length records. In a domain based on such a file, you can change the number of list items only up to the maximum value specified in the OCCURS DEPENDING clause. You cannot exceed that value, since the MAX clause in the file definition causes DATATRIEVE to create a fixed-length RMS file based on the maximum value in the OCCURS DEPENDING clause.

The least restrictive case is a data file you define using the KEY clause but not the MAX clause. This creates an indexed file with variable-length records. In such a file, you can change the number of list items to any number you want. The following example shows how to increase the number of list items for a domain based on an indexed file with variable-length records.

```
DTR> READY INDEXED_FAMILIES WRITE
DTR> FIND FIRST 1 INDEXED_FAMILIES
[1 Record found]
DTR> PRINT
No record selected, printing whole collection

                        NUMBER     KID
  FATHER      MOTHER     KIDS      NAME     AGE

JIM          ANN           2     URSULA      7
                                 RALPH       3
```

```
DTR> SELECT
DTR> MODIFY NUMBER_KIDS
Enter NUMBER_KIDS: 4
DTR> FIND KIDS
[4 records found]
DTR> SELECT 3
DTR> MODIFY
Enter KID_NAME: NICKY
Enter AGE: 2
DTR> SELECT 4
DTR> MODIFY
Enter KID_NAME: TAM
Enter AGE: 1
DTR> PRINT FIRST 1 INDEXED_FAMILIES

                        NUMBER     KID
  FATHER      MOTHER     KIDS      NAME     AGE

JIM          ANN           4    URSULA       7
                                RALPH        3
                                NICKY        2
                                TAM          1
DTR>
```

6.4 Creating Hierarchies with Multiple RSEs

The complications that occur when you have to retrieve or modify data stored in repeating fields make it a good idea to avoid using hierarchical records.

However, you can have the benefits of hierarchical records without the disadvantages by creating hierarchies from flat records. There are several advantages to hierarchies based on flat records.

- Because they are based on flat records, you avoid the complications of retrieving and modifying data stored in records with repeating fields. You can simply print or modify fields directly in domains based on the flat records.
- Like records with repeating fields, they let you display a parent-child relationship between data when you want to.
- They offer more flexibility because the parent-child relationship is not imposed by the record definition.
- There is no limit to the number of occurrences of the "child" record stream In records with repeating fields, the OCCURS clause limits how many valu a repeating field can store.

'his section describes three techniques for combining record streams to form ierarchies:

View domains

Inner print lists

Nested FOR statements

'ach of the techniques creates a hiearchical relationship without using repeating elds in a record definition. Instead, they nest record streams from separate omains to create the one-to-many relationship characteristic of a hierarchy.

.4.1 Creating Hierarchies with View Domains

'iew domains do not form hierarchies unless they nest more than one record tream. For instance, view domains that use only fields from a single domain are ot hierarchical.

'iews that combine data from two record sources with the CROSS clause are not ierarchical either, because the record streams are joined (a one-to-one relationhip between the record streams) instead of nested (a one-to-many relationship etween the record streams). The view described in the section on flattening ierarchies with the cross clause was an example of this kind of view.

Iowever, any view domain definition that uses more than one OCCURS FOR lause creates a hierarchy. It creates a parent-child relationship between the RSE pecified in the first OCCURS FOR clause and the RSE specified in the second.

'o access a hierarchical view, you need to use the techniques to retrieve values :om repeating fields that are shown in this chapter. (You access a nonhierarchical iew as you would any other flat domain. See the chapter on Using View Domains)r details.) A hierarchical view domain contains more than one OCCURS FOR lause. DATATRIEVE considers the second and following OCCURS FOR clauses s lists, or repeating fields. For example, the SAILBOATS domain is a hierachical iew based on two domains, YACHTS and OWNERS.

```
TR> SHOW SAILBOATS
OMAIN SAILBOATS
OF YACHTS, OWNERS BY
1 SAILBOAT OCCURS FOR YACHTS.
  03 BOAT FROM YACHTS.
  03 SKIPPERS OCCURS FOR OWNERS WITH TYPE EQ BOAT.TYPE.
     05 NAME FROM OWNERS.
```

Iote that the second OCCURS FOR clause in SAILBOATS refers to the field ame TYPE twice, since the OWNERS and YACHTS domains both have a field f that name. You can qualify a field name by adding a prefix to it to differentiate

it from the other field with the same name. In this case, BOAT.TYPE specifies the TYPE field in the YACHTS domain and distinguishes it from the TYPE field in the OWNERS domain. See Appendix A for more information about qualified field names.

After you ready SAILBOATS, the SHOW FIELDS command shows that the sec ond OCCURS FOR clause creates a hierarchy:

```
DTR> READY SAILBOATS
DTR> SHOW FIELDS SAILBOATS
SAILBOATS
   SAILBOAT
      BOAT
         TYPE   <Indexed field>
            MANUFACTURER (BUILDER)      <Character string, indexed key>
            MODEL       <Character string, indexed key>
         SPECIFICATIONS (SPECS)
            RIG <Character string>
            LENGTH_OVER_ALL (LOA)       <Character string>
            DISPLACEMENT (DISP) <Number>
            BEAM        <Number>
            PRICE       <Number>
      SKIPPERS  <List>
         NAME   <Character string>

DTR>
```

The SKIPPERS field is a list field that repeats for each YACHTS record in the outer stream. To refer to field values contained in the list, you must use one of the methods in this chapter. For example, use multiple FIND and SELECT state ments to print then modify the names of owners of a certain TYPE of yacht:

```
DTR> READY SAILBOATS WRITE
DTR> FIND OWNED IN SAILBOATS WITH ANY SKIPPERS
[6 records found]
DTR> PRINT ALL

                                LENGTH
                                 OVER                          OWNER
MANUFACTURER    MODEL     RIG    ALL   WEIGHT BEAM  PRICE      NAME

 ALBIN          VEGA      SLOOP  27     5,070  08   $18,600   STEVE
                                                               HUGH
 C&C            CORVETTE  SLOOP  31     8,650  09             JIM
                                                               ANN
 ISLANDER       BAHAMA    SLOOP  24     4,200  08    $6,500   JIM
                                                               ANN
                                                               STEVE
                                                               HARVE
 PEARSON        10M       SLOOP  33    12,441  11             TOM
 PEARSON        26        SLOOP  26     5,400  08             DICK
 RHODES         SWIFTSURE SLOOP  33    14,000  10             JOHN

DTR> SELECT 3
DTR> FIND SKIPPERS
[4 records found]
```

```
R> PRINT ALL

NER
ME

M
N
EVE
RVE

R> SELECT 2
R> MODIFY NAME
ter NAME: ANNE
R> PRINT BOAT, ALL SKIPPERS SORTED BY NAME OF OWNED

                                LENGTH
                                 OVER                                OWNER
NUFACTURER    MODEL      RIG     ALL    WEIGHT BEAM   PRICE          NAME

LBIN          VEGA       SLOOP   27      5,070  08   $18,600 HUGH
                                                               STEVE
&C            CORVETTE   SLOOP   31      8,650  09           ANN
                                                               JIM
SLANDER       BAHAMA     SLOOP   24      4,200  08    $6,500 ANNE
                                                               HARVE
                                                               JIM
                                                               STEVE
EARSON        10M        SLOOP   33     12,441  11           TOM
EARSON        26         SLOOP   26      5,400  08           DICK
HODES         SWIFTSURE  SLOOP   33     14,000  10           JOHN

R>
```

ou can use also inner print lists to display the owners of ISLANDER BAHAMA ichts directly:

```
R> PRINT ALL -
N>            ALL NAME OF SKIPPERS SORTED BY NAME -
N>       OF SAILBOATS WITH ANY SKIPPERS AND
N>                          MANUFACTURER = "ISLANDER"

NER
ME

NE
RVE
M
EVE

R>
```

4.2 Using Inner Print Lists to Create Dynamic Hierarchies

ne preceding examples used the view domain SAILBOATS, based on the ACHTS and OWNERS domains. To create this view, you needed to use a view omain definition. But you can create the same effect dynamically by using an

inner print list. The inner print list can include an RSE that references one domain, while the outer RSE refers to a second domain. The following example produces the same display as printing the SAILBOATS view domain:

```
DTR> PRINT BOAT,                              !Print list for rse-2
CON>     ALL NAME                             !Print list for rse-1
CON>     OF OWNERS WITH TYPE = BOAT.TYPE -    !rse-1, uses OWNERS
CON> OF YACHTS                                !rse-2, uses YACHTS

                                  LENGTH
                                   OVER                                  OWNER
MANUFACTURER    MODEL      RIG     ALL    WEIGHT BEAM   PRICE            NAME

 ALBERG         37 MK II   KETCH   37     20,000  12    $36,951
 ALBIN          79         SLOOP   26      4,200  10    $17,900
 ALBIN          BALLAD     SLOOP   30      7,276  10    $27,500
 ALBIN          VEGA       SLOOP   27      5,070  08    $18,600          STEVE
                                                                         HUGH
 AMERICAN       26         SLOOP   26      4,000  08     $9,895
 AMERICAN       26-MS      MS      26      5,500  08    $18,895
                                     .
                                     .
                                     .

DTR>
```

Here, the inner print list is ALL NAME OF OWNERS WITH TYPE = BOAT.TYPE. The inner print list serves to relate each owner of a particular mak and model with that make and model boat in the outer stream, formed from the YACHTS domain.

A second example generates a display of yacht and owner information for any yacht that has an owner. This query adds a restriction on the outer RSE using th ANY rse Boolean expression:

```
DTR> PRINT BOAT,                              !Print list for rse-2
CON>     ALL NAME                             !Print list for rse-1
CON>     OF OWNERS WITH TYPE = BOAT.TYPE -    !rse-1, uses OWNERS
CON> OF YACHTS WITH ANY                       !rse-2, uses YACHTS
CON>       OWNERS WITH TYPE = BOAT.TYPE       !rse for ANY clause of rse-2

                                  LENGTH
                                   OVER                                  OWNER
MANUFACTURER    MODEL      RIG     ALL    WEIGHT BEAM   PRICE            NAME

 ALBIN          VEGA       SLOOP   27      5,070  08    $18,600          STEVE
                                                                         HUGH
 C&C            CORVETTE   SLOOP   31      8,650  09                     JIM
                                                                         ANN
 ISLANDER       BAHAMA     SLOOP   24      4,200  08     $6,500          JIM
                                                                         ANN
                                                                         STEVE
                                                                         HARVE
 PEARSON        10M        SLOOP   33     12,441  11                     TOM
 PEARSON        26         SLOOP   26      5,400  08                     DICK
 RHODES         SWIFTSURE  SLOOP   33     14,000  10                     JOHN

DTR>
```

.4.3 Using Nested FOR Statements to Create Dynamic Hierarchies

ou can also create dynamic hierarchies by nesting FOR statements. Although ested FOR statements are logically equivalent to inner print lists or view omains with nested OCCURS clauses, DATATRIEVE displays the data differ- ıtly.

he following example uses nested FOR statements to retrieve the same informa- on that printing the SAILBOATS view domain retrieves:

```
TR> FOR YACHTS
ON>    BEGIN
ON>    PRINT BOAT
ON>    FOR OWNERS WITH TYPE = BOAT.TYPE
ON>       PRINT NAME
ON>    END

                                LENGTH
                                 OVER
ANUFACTURER   MODEL      RIG     ALL    WEIGHT BEAM  PRICE

ALBERG        37 MK II   KETCH   37     20,000  12  $36,951
ALBIN         79         SLOOP   26      4,200  10  $17,900
ALBIN         BALLAD     SLOOP   30      7,276  10  $27,500
ALBIN         VEGA       SLOOP   27      5,070  08  $18,600

WNER
AME

TEVE
UGH
AMERICAN      26         SLOOP   26      4,000  08   $9,895
AMERICAN      26-MS      MS      26      5,500  08  $18,895
                                 .
                                 .
                                 .

TR>
```

'ou can control the printing format to make the display similar to that produced y printing the SAILBOATS view domain:

```
TR> FOR YACHTS
ON>    BEGIN
ON>    PRINT BOAT
ON>    FOR OWNERS WITH TYPE = BOAT.TYPE
ON>       PRINT COL 60, NAME (-)
ON>    END
```

MANUFACTURER	MODEL	RIG	LENGTH OVER ALL	WEIGHT	BEAM	PRICE	
ALBERG	37 MK II	KETCH	37	20,000	12	$36,951	
ALBIN	79	SLOOP	26	4,200	10	$17,900	
ALBIN	BALLAD	SLOOP	30	7,276	10	$27,500	
ALBIN	VEGA	SLOOP	27	5,070	08	$18,600	STEVE HUGH
AMERICAN	26	SLOOP	26	4,000	08	$9,895	
AMERICAN	26-MS	MS	26	5,500	08	$18,895	

DTR>

Part 3
Programming in DATATRIEVE

Using DATATRIEVE Procedures 7

ften you want to execute the same series of DATATRIEVE commands and atements over and over again, and you may want to have other users execute ıose same commands and statements. You have to retype the input each time, ıless you put the commands and statements in a procedure. By using proceıres, you can develop the series of steps once and then simply invoke the proceıre each time you want to do the same thing again. A *procedure* is a fixed :quence of DATATRIEVE commands and statements you create, name, and ore in the Common Data Dictionary.

.1 Defining a Procedure

ɔr almost any series of commands and statements you use repeatedly, you can ıve yourself time by defining a procedure. Perhaps there is a simple query you ıve to enter frequently using a particular database. You might, for instance, ish to know all the manufacturers of large yachts:

```
'R> READY YACHTS
'R> FIND BIGGIES IN YACHTS WITH LOA GT 40 SORTED BY BUILDER
 records found]
'R> PRINT ALL

                                LENGTH
                                 OVER
NUFACTURER    MODEL      RIG     ALL    WEIGHT BEAM  PRICE

HALLENGER     41         KETCH   41     26,700  13   $51,228
OLUMBIA       41         SLOOP   41     20,700  11   $48,490
ULFSTAR       41         KETCH   41     22,000  12   $41,350
SLANDER       FREEPORT   KETCH   41     22,000  13   $54,970
AUTOR         SWAN 41    SLOOP   41     17,750  12
EWPORT        41 S       SLOOP   41     18,000  11
LYMPIC        ADVENTURE  KETCH   42     24,250  13   $80,500
EARSON        419        KETCH   42     21,000  13

R>
```

Although this sequence is short, putting it in a procedure and invoking the procedure is useful if you need to produce the display frequently.

To define a procedure, enter the DEFINE PROCEDURE command at DATATRIEVE command level:

DEFINE PROCEDURE procedure-name

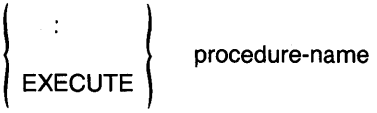
DATATRIEVE then prompts with DFN > to indicate that it expects a procedure definition. Enter the commands or statements that form the procedure definition. DATATRIEVE continues to prompt with DFN > until you enter the keyword END PROCEDURE on a line by itself.

```
DTR> DEFINE PROCEDURE BIG_YACHTS
DFN> READY YACHTS
DFN> FIND BIGGIES IN YACHTS WITH LOA GT 40 SORTED BY BUILDER
DFN> PRINT ALL
DFN> END_PROCEDURE
DTR>
```

As soon as you enter END_PROCEDURE, DATATRIEVE stores the procedure definition in your default dictionary directory. DATATRIEVE does not check for syntax errors when you enter the procedure definition but does check when you invoke the procedure.

7.2 Invoking a Procedure

You invoke a procedure by preceding its name with the keyword EXECUTE or with a colon (:).



To invoke a procedure, you must have P (PASS_THRU), S (SEE), and E (EXECUTE_EXTEND) access to it.

The content of a procedure determines where you can invoke it. In general, you can invoke a procedure anywhere you can use the commands or statements contained in the procedure. For example, if the procedure contains DATATRIEVE commands and statements, you can invoke it at the DATATRIEVE command level:

```
DTR> :BIG_YACHTS
```

'ou cannot invoke a procedure during an ADT, EDIT, or Guide Mode session.
'ou cannot include a procedure in a domain, record, or table definition.

)ATATRIEVE does not display the contents of procedures as they execute, even
: you issue the SET VERIFY command.

'ou do not have to enter DATATRIEVE to invoke a procedure. You can invoke a
rocedure from the VMS command level. For example, if DTR32 is your DCL
ymbol for invoking DATATRIEVE, you can invoke BIG_YACHTS with this
ommand line at the system prompt:

```
DTR32 EXECUTE BIG_YACHTS
```

.fter DATATRIEVE executes the last command or statement in the file, you are
utomatically returned to the system prompt.

'ou can use the colon to execute a procedure from VMS level, but you must pre-
ede it with a semicolon (;):

```
DTR32; :BIG_YACHTS
```

or DATATRIEVE procedures that are run often from VMS level, you can also
efine a DCL symbol for the entire command line shown in the last example:

```
BIG_YACHTS :== "''DTR32'; :BIG_YACHTS"
```

sers can run DATATRIEVE procedures this way simply by entering at VMS
)mmand level the symbol you define.

.3 Contents of a Procedure

procedure can contain any number of the following DATATRIEVE elements:

- Full DATATRIEVE commands and statements
- Command and statement clauses and arguments
- Comments

3.1 Commands and Statements

hen you execute BIG_YACHTS, the results are the same as entering the
EADY command and the FIND and PRINT statements at command level.

```
DTR> :BIG_YACHTS

                                LENGTH
                                 OVER
MANUFACTURER    MODEL      RIG   ALL   WEIGHT BEAM  PRICE

 CHALLENGER  41          KETCH   41    26,700  13  $51,228
 COLUMBIA    41          SLOOP   41    20,700  11  $48,490
 GULFSTAR    41          KETCH   41    22,000  12  $41,350
 ISLANDER    FREEPORT    KETCH   41    22,000  13  $54,970
 NAUTOR      SWAN 41     SLOOP   41    17,750  12
 NEWPORT     41 S        SLOOP   41    18,000  11
 OLYMPIC     ADVENTURE   KETCH   42    24,250  13  $80,500
 PEARSON     419         KETCH   42    21,000  13

DTR>
```

7.3.2 Arguments and Clauses

Besides full commands and statements, a procedure can contain simply an argument or clause from a command or statement. For example, it can contain a record selection expression:

```
DTR> DEFINE PROCEDURE BIG_YACHTS_RSE
DFN> BIGGIES IN YACHTS WITH LOA GT 40 SORTED BY BUILDER
DFN> END_PROCEDURE
DTR>
```

Having separated the FIND statement from the record selection expression, you can invoke the procedure to complete a FIND command:

```
DTR> FIND :BIG_YACHTS_RSE
[8 records found]
DTR>
```

In fact, you can use this procedure in any command or statement containing an RSE argument, such as the PRINT statement:

```
DTR> PRINT ALL :BIG_YACHTS_RSE

                                LENGTH
                                 OVER
MANUFACTURER    MODEL      RIG   ALL   WEIGHT BEAM  PRICE

 CHALLENGER  41          KETCH   41    26,700  13  $51,228
 COLUMBIA    41          SLOOP   41    20,700  11  $48,490
 GULFSTAR    41          KETCH   41    22,000  12  $41,350
 ISLANDER    FREEPORT    KETCH   41    22,000  13  $54,970
 NAUTOR      SWAN 41     SLOOP   41    17,750  12
 NEWPORT     41 S        SLOOP   41    18,000  11
 OLYMPIC     ADVENTURE   KETCH   42    24,250  13  $80,500
 PEARSON     419         KETCH   42    21,000  13
```

.3.3 Comments

Vhen you define a procedure, you can include comments, which DATATRIEVE tores in the CDD.

: you want to display comments on the terminal when you (or another user) xecute your procedure, you can use the PRINT command, as shown in the fol-)wing example:

```
TR> DEFINE PROCEDURE YACHTS_REPORT
?N> PRINT "THIS REPORT REQUIRES AN ESTABLISHED COLLECTION"
?N> PRINT "SORTED BY BUILDER"
?N>          .
?N>          .
?N>          .
```

/hen you execute the procedure, the first two lines print a message on the :rminal:

```
TR> :YACHTS_REPORT
IIS REPORT REQUIRES AN ESTABLISHED COLLECTION
)RTED BY BUILDER
```

nless you use PRINT statements to display comments and messages in a proce- ıre, DATATRIEVE does not display any of its contents. This is true whether or)t SET VERIFY is in effect. You can, however, include comments that are not splayed during execution by placing an exclamation point (!) before each com- ent line:

```
'R> DEFINE PROCEDURE YACHTS_REPORT
'N> ! LATEST VERSION 01-Apr-84
'N> PRINT "THIS REPORT REQUIRES AN ESTABLISHED COLLECTION"
'N> PRINT "SORTED BY BUILDER"
'N>          .
'N>          .
'N>          .
'R> :YACHTS_REPORT
 IS REPORT REQUIRES AN ESTABLISHED COLLECTION
 RTED BY BUILDER
```

)u can use comments in a procedure to explain the purpose of its parts and, :nce, make it easy for you and others to maintain.

4 Editing a Procedure

hen you invoke a procedure, DATATRIEVE executes each command or state- ent in the procedure as if it were entered directly at DATATRIEVE command /el. Some errors may occur during execution of the procedure. A typing error, r instance, can result in a syntax error in an otherwise correctly formatted com- and. If an error occurs during execution, DATATRIEVE prints an error mes- ge and terminates the procedure. You can correct the error by using the ATATRIEVE Editor.

Invoke the Editor with the following command:

```
EDIT procedure-name
```

When you find the error, use the appropriate Editor commands to correct it.

7.5 Troubleshooting Procedures

You can put an entire series of statements into a procedure and then tell DATATRIEVE to execute the procedure. If you have made any errors, DATATRIEVE stops executing the procedure when it finds the first error and sends you an appropriate error message. The following example contains two errors:

```
DTR> DEFINE PROCEDURE WAGE_REPORT
DFN> REPORT WAGES
DFN> SET REPORT_NAME = WEEKLY WAGE REPORT
DFN> SET COLUMNS-PAGE = 70
DFN> PRINT LAST_NAME, GROSS_PAY, FICA,
DFN> FEDERAL_TAX, STATE_TAX,
DFN> GROSS_PAY - (FICA + FEDERAL_TAX + STATE_TAX)-
DFN>              ("NET PAY") USING $$,$$$.99
DFN> AT BOTTOM OF REPORT PRINT SKIP 2, COL 1, "TOTAL:",
DFN> TOTAL GROSS_PAY USING $$$,$$$.99,
DFN> TOTAL FICA USING $$$,$$$.99,
DFN> TOTAL FEDERAL_TAX USING $$$,$$$.99,
DFN> TOTAL STATE_TAX USING $$$,$$$.99,
DFN> TOTAL (GROSS_PAY - (FICA + FEDERAL_TAX +
DFN>       STATE_TAX)) USING $$$,$$$.99
DFN> END_REPORT
DFN> END_PROCEDURE
DTR> :WAGE_REPORT
REPORT WAGES
SET REPORT_NAME = WEEKLY WAGE REPORT
                  ^
Expected header segment, encountered "WEEKLY".
```

If you would like to refer to the record definition for the WAGES domain, see th record definition appendix in the *VAX DATATRIEVE Guide to Writing Reports*

To correct the error in the SET REPORT_NAME command, use the Editor and place quotation marks (" ") around WEEKLY WAGE REPORT. Then invoke th procedure again:

```
DTR> :WAGE_REPORT
"WAGES" is not a readied source, collection, or list
DTR> EDIT WAGE_REPORT
```

Using the Editor, place the READY WAGES command before the report statement and invoke the procedure again.

```
R> :WAGE_REPORT

                        WEEKLY WAGE REPORT              9-Aug-1984
                                                        Page 1

 LAST        GROSS                      FEDERAL      STATE
 NAME         PAY          FICA           TAX         TAX        NET PAY

AKE          $1,000.00     $103.86      $204.77        $.01      $691.36
NN           $1,500.00     $145.87      $297.98      $54.32    $1,001.83
LL             $500.00      $52.93       $79.75      $32.98      $334.34
NES            $999.99     $103.85      $204.76      $57.90      $633.48
ONY          $1,900.98     $145.87      $375.98      $75.90    $1,303.23
ARK          $9,500.00     $145.87      $999.84     $106.90    $8,247.39

TAL:        $15,400.97     $698.25    $2,163.08     $328.01   $12,211.63
```

6 Aborting Procedures

ou can abort a procedure by including an ABORT statement in it. If the abort nditions arise and SET ABORT is in effect, DATATRIEVE aborts the proce- ire and prints a message on your terminal. If SET NO ABORT is in effect, ATATRIEVE aborts the statement that contains the ABORT but continues to ecute the other commands and statements in the procedure.

ie default setting in DATATRIEVE is SET ABORT. You can ensure that SET BORT is in effect by including that statement in the procedure definition. For ample:

```
R> DEFINE PROCEDURE BIG_YACHTS_QUERY
N> SET ABORT
N> DECLARE LENGTH PIC 99
N>   VALID IF LENGTH GT 35.
N> LENGTH = *."MIN LOA"
N> IF LENGTH GT 42
N>   THEN ABORT "NO BOATS THAT BIG"
N> FIND BIGGIES IN YACHTS WITH LOA GE LENGTH
N>   SORTED BY BUILDER
N> PRINT BUILDER, RIG, LOA, PRICE OF BIGGIES
N> END_PROCEDURE
R>
```

you invoke BIG_YACHTS_QUERY and supply a length of 35 or smaller, ATATRIEVE reprompts you for a valid length. If you supply a length greater an 42, the procedure aborts, prints the specified abort message, and returns you the DATATRIEVE command level.

```
DTR> :BIG_YACHTS_QUERY
Enter MIN LOA: 35
Validation error for LENGTH
Re-enter MIN LOA: 43
ABORT: NO BOATS THAT BIG
DTR>
```

If you assign a value between 36 and 42 to LENGTH, DATATRIEVE prints the appropriate collection:

```
DTR> :BIG_YACHTS_QUERY
Enter MIN LOA: 39

                        LENGTH
                         OVER
MANUFACTURER   RIG       ALL     PRICE

 BLOCK I.      SLOOP     39
 CHALLENGER    KETCH     41      $51,228
 COLUMBIA      SLOOP     41      $48,490
 GULFSTAR      KETCH     41      $41,350
 ISLANDER      KETCH     41      $54,970
 LINDSEY       MS        39      $35,900
 NAUTOR        SLOOP     41
 NEWPORT       SLOOP     41
 OLYMPIC       KETCH     42      $80,500
 PEARSON       SLOOP     39
 PEARSON       KETCH     42
```

7.7 Sample Procedures

With the information from previous chapters on commands, statements, and pro cedures, you have enough techniques available to design numerous procedures. Later sections of this chapter describe:

- Including procedures within other procedures (nesting)
- Using procedures in compound statements
- Aborting procedures
- Generalizing procedures so they work on more than one domain
- Maintaining procedures to accommodate changes you want to make in then

Use the following examples as models for procedures you create yourself. Example 7-1 shows you how to create a procedure that uses the Report Writer t write a summary report of yacht data.

:xample 7-1: Sample Procedure Using the Report Writer

```
TR> DEFINE PROCEDURE YACHT_SUMMARY
FN> SET ABORT
FN> PRINT "THIS REPORT REQUIRES AN ESTABLISHED COLLECTION,"
FN> PRINT "SORTED BY LOA AND BEAM."
FN> PRINT "HAVE YOU ESTABLISHED A COLLECTION?"      ---------(1)
FN> IF *."YES OR NO" CONTAINING "N" THEN
FN>   ABORT "SORRY, NO  COLLECTION."      ----------------------(2)
FN> REPORT ON *."OUTPUT DEVICE OR FILE"       ----------------(3)
FN> SET REPORT_NAME="EXAMPLE: REPORT FROM A PROCEDURE"
FN> SET LINES_PAGE=55, COLUMNS_PAGE=60
FN> PRINT BUILDER, MODEL, LOA, BEAM, PRICE
FN> AT BOTTOM OF LOA PRINT SKIP, "AVERAGE PRICE =", AVERAGE PRICE, SKIP
FN> AT BOTTOM OF REPORT PRINT COL 17,"NUMBER OF BOATS = ", COL 42, COUNT,
FN> SKIP, "AVERAGE PRICE OF ALL BOATS =", AVERAGE PRICE
FN> END_REPORT
FN> END_PROCEDURE
TR>
```

he example illustrates some statements that are particularly useful in rocedures:

1. Use the PRINT statement to display a message to whoever invokes the procedure.

2. The prompting value expression *."YES OR NO" requires the user to respond to the question: HAVE YOU ESTABLISHED A COLLECTION? The Boolean expression CONTAINING checks the user's response to the question.

   If you invoke YACHT_SUMMARY and answer NO to the first prompt, the procedure aborts:

   ```
   DTR> :YACHT_SUMMARY
   THIS REPORT REQUIRES AN ESTABLISHED COLLECTION,
   SORTED BY LOA AND BEAM.
   HAVE YOU ESTABLISHED A COLLECTION?
   Enter YES OR NO: NO
   ABORT: SORRY, NO  COLLECTION.
   DTR>
   ```

   If you answer YES to the first prompt, but, in fact, you do not have a current collection, the Report Writer aborts the procedure and prints an error message:

   ```
   DTR> :YACHT_SUMMARY
   THIS REPORT REQUIRES AN ESTABLISHED COLLECTION,
   SORTED BY LOA AND BEAM.
   ```

(continued on next page)

```
HAVE YOU ESTABLISHED A COLLECTION?
Enter YES OR NO: YES
A current collection has not been established.
DTR>
```

3. The prompting value expression *."OUTPUT DEVICE OR FILE" allows the user to select the device or file to contain the report when DATATRIEVE executes the procedure.

   If you make a collection of YACHTS with LOA between 36 and 37 and pric not equal to zero, the following report results:

```
DTR> READY YACHTS
DTR> FIND YACHTS WITH LOA BETWEEN 36 37 AND PRICE NE 0
[5 records found]
DTR> SORT CURRENT BY LOA, BEAM
DTR> :YACHT_SUMMARY
THIS REPORT REQUIRES AN ESTABLISHED COLLECTION,
SORTED BY LOA AND BEAM.
HAVE YOU ESTABLISHED A COLLECTION?
Enter YES OR NO: YES
Enter OUTPUT DEVICE OR FILE: TT:


                    REPORT FROM A PROCEDURE   01-Apr-1984
                                              Page 1


                                   LENGTH
                                    OVER
MANUFACTURER          MODEL         ALL       BEAM      PRICE


  ISLANDER            36            36          11      $31,730
  I. TRADER           37            36          12      $39,500

AVERAGE PRICE =                                         $35,615

  IRWIN               37 MARK II    37          11      $36,950
  NORTHERN            37            37          11      $50,000
  ALBERG              37 MK II      37          12      $36,951

AVERAGE PRICE =                                         $41,300

                NUMBER OF BOATS =              5
                AVERAGE PRICE OF ALL BOATS =            $39,026
```

7.8 How to Nest Procedures Within Procedures

A nested procedure is a procedure within another procedure. You can use this technique to create one procedure that can be used by several other procedures.

he following procedure calculates the price per pound of a boat and assigns a colnn header and edit-string for that value expression. You cannot invoke this produre by itself, but you can invoke the PRICE_PER_POUND procedure in nother procedure that prints the builder, model, and price per pound of all boats the CURRENT collection. These commands define the two procedures:

```
R> DEFINE PROCEDURE PRICE_PER_POUND
N> PRICE/DISPLACEMENT ("PRICE"/"PER"/"POUND") USING
N>  $$9.99
N> END_PROCEDURE
R>


R> DEFINE PROCEDURE PRICE_REPORT
N> PRINT ALL BUILDER, MODEL, :PRICE_PER_POUND
N> END_PROCEDURE
R>
```

hen you invoke the procedure PRICE_REPORT, DATATRIEVE displays the ree fields on the terminal. The following example uses the BIG_YACHTS produre to establish the CURRENT collection and PRICE_REPORT to print a ort report:

```
R> :BIG_YACHTS; :PRICE_REPORT

                                LENGTH
                                 OVER
NUFACTURER    MODEL      RIG     ALL    WEIGHT BEAM  PRICE

HALLENGER  41          KETCH    41     26,700  13  $51,228
OLUMBIA    41          SLOOP    41     20,700  11  $48,490
ULFSTAR    41          KETCH    41     22,000  12  $41,350
SLANDER    FREEPORT    KETCH    41     22,000  13  $54,970
AUTOR      SWAN 41     SLOOP    41     17,750  12
EWPORT     41 S        SLOOP    41     18,000  11
LYMPIC     ADVENTURE   KETCH    42     24,250  13  $80,500
EARSON     419         KETCH    42     21,000  13

                         PRICE
                          PER
NUFACTURER    MODEL      POUND

HALLENGER  41           $1.92
OLUMBIA    41           $2.34
ULFSTAR    41           $1.88
SLANDER    FREEPORT     $2.50
AUTOR      SWAN 41      $0.00
EWPORT     41 S         $0.00
LYMPIC     ADVENTURE    $3.32
EARSON     419          $0.00

R>
```

When nesting procedures, do not allow a procedure to invoke itself or you create an infinite loop.

7.9 Using a Procedure in a Compound Statement

You can invoke a procedure in a REPEAT statement to execute it a number of times or in a FOR statement to apply it to a record stream. You must, however, use care when invoking a procedure in these statements. For example, the following statement is syntactically correct, but produces results you may not expect:

```
REPEAT 5 :procedure-name
```

This statement *does not* execute the procedure five times. When DATATRIEVE encounters the first complete statement in the procedure, it assumes that the REPEAT statement is also complete. Therefore, it executes the *first* statement in the procedure five times. DATATRIEVE then executes the remaining statements in the procedure once each.

To repeat the entire procedure, enclose the procedure call or the procedure definition in a BEGIN-END block. For example, the following sequence of statements puts a procedure call in a BEGIN-END block and repeats the procedure five times:

```
DTR> REPEAT 5 BEGIN
[Looking for statement]
CON> :HEAVY_SLOOP
CON> END
DTR>
```

The following example includes a FOR statement and a BEGIN-END block in a procedure definition and invokes the procedure in a REPEAT statement:

```
DTR> SHOW HEAVY_SLOOP
PROCEDURE HEAVY_SLOOP
FOR YACHTS WITH BUILDER = *."MANUFACTURER"
   BEGIN
     IF RIG = "SLOOP" AND DISP GE 10000
     PRINT BOAT
   END
END_PROCEDURE
```

```
TR> REPEAT 3 :HEAVY_SLOOP
nter MANUFACTURER: CAL

                                    LENGTH
                                     OVER
ANUFACTURER    MODEL       RIG      ALL    WEIGHT BEAM  PRICE

CAL          3-30        SLOOP     30     10,500  10
CAL          35          SLOOP     35     15,000  11
nter MANUFACTURER: PEARSON
PEARSON      10M         SLOOP     33     12,441  11
PEARSON      35          SLOOP     35     13,000  10
PEARSON      36          SLOOP     37     13,500  11
PEARSON      39          SLOOP     39     17,000  12
nter MANUFACTURER: NAUTOR
NAUTOR       SWAN 41     SLOOP     41     17,750  12

TR>
```

f you invoke a procedure in a FOR statement, you must use the same technique: nclose the call or the procedure definition in a BEGIN-END block. For instance:

```
OR rse BEGIN :procedure-name END
```

emember, if you use a procedure in a loop, do not include any commands or a IND, SELECT, DROP, SORT, or REDUCE statement in that procedure. ATATRIEVE does not accept commands or any of these statements in BEGIN-ND blocks or other compound statements.

.10 Generalizing Procedures

ou can generalize procedures so that they operate on numerous domains. Be ıre that the generalized procedures meet the following two conditions:

- The corresponding fields in the various domains have the same name.
- The procedure refers to an alias rather than to the domain name.

or example, you might want to keep separate domains for boats from different eographical areas, perhaps one each for boats from the east, west, and south oasts. If the domains WEST_YACHTS, EAST_YACHTS, and OUTH_YACHTS have the same record definition and data file format, you can se one procedure on all three.

1. When you ready each domain, rename it with an alias, using the AS clause in the READY command. To create the alias ALL_YACHTS for the domain WEST_YACHTS, respond to the DTR> prompt with this READY command:

```
DTR> READY WEST_YACHTS AS ALL_YACHTS
DTR>
```

   You have established an alias for the WEST_YACHTS domain. During your current session, DATATRIEVE recognizes all references to the alias ALL_YACHTS as references to WEST_YACHTS.

2. When you define a procedure, refer to the alias rather than to a domain name. In procedures you have already defined, you can use the EDIT procedure-name command to change domain names to an alias. Thus, you can generalize the procedure BIG_YACHTS_QUERY by changing the domain name YACHTS in the FIND statement to ALL_YACHTS:

```
DFN> FIND BIGGIES IN ALL_YACHTS WITH LOA GE LENGTH -
DFN> SORTED BY BUILDER
```

3. When you invoke the BIG_YACHTS_QUERY procedure, DATATRIEVE applies it to the domain readied with the alias ALL_YACHTS. After you have executed the BIG_YACHTS_QUERY procedure, you can use it on another domain. You must first use the FINISH command to remove the readied domain WEST_YACHTS under the alias ALL_YACHTS from your workspace. Then you can ready the next domain you want the procedure BIG_YACHTS_QUERY to work on:

```
DTR> READY WEST_YACHTS AS ALL_YACHTS
DTR> SHOW READY
Ready sources:
   ALL_YACHTS:  Domain, RMS indexed, protected read
          <CDD$TOP.DTR$LIB.DEMO.WEST_YACHTS;1>
No loaded tables.

DTR> :BIG_YACHTS_QUERY
Enter min LOA: 39

                      LENGTH
                       OVER
MANUFACTURER  RIG      ALL     PRICE

 GOOBER       MS       40     $42,000
 HEILE        SLOOP    41     $61,400
 INVEIGH      KETCH    41     $48,950
```

```
DTR> FINISH
DTR> SHOW READY
No ready sources.
No loaded tables.
DTR> READY EAST_YACHTS AS ALL_YACHTS
DTR> SHOW READY
Ready sources:
   ALL_YACHTS:  Domain, RMS indexed, protected read
         <CDD$TOP.DTR$LIB.DEMO.EAST_YACHTS;1>
No loaded tables.

DTR>
```

11 Maintaining Procedures

ɔu can maintain the procedures stored in your default dictionary directory with e SHOW, EDIT, and DELETE commands.

11.1 Displaying Procedure Names

ɔu can list the names of all procedures in your default directory with the SHOW mmand:

```
R> SHOW PROCEDURES
ocedures:
  BIG_YACHTS;1     BIG_YACHTS_QUERY;1          CHEAP;1
  MS_SEARCH;1      PHONE_REP;1        TEST;1   YACHT_SUMMARY;1

R>
```

11.2 Displaying Procedures

you want to display a procedure on your terminal, you can use the SHOW com- ınd and specify the name of the procedure to be displayed. You must have P ASS_THRU), S (SEE), and R (READ) access privilege to the procedure.

```
R> SHOW MS_SEARCH
OCEDURE MS_SEARCH
ADY YACHTS
ND YACHTS WITH RIG = "MS"
R CURRENT PRINT BUILDER,
JILDER VIA COMPANY_TABLE) ("ADDRESS")
)_PROCEDURE

ɪ>
```

7.11.3 Deleting Procedures

You can delete a procedure from your default dictionary directory with the DELETE command. You must have P (PASS_THRU) and X (EXTEND) access privileges to the parent directory and P (PASS_THRU) and either D (LOCAL_DELETE) or G (GLOBAL_DELETE) access to the procedure.

```
DTR> SHOW PROCEDURES
Procedures:
        BIG_YACHTS      BIG_YACHTS_QUERY                CHEAP
        MS_SEARCH       PHONE_REP       YACHT_SUMMARY

DTR> DELETE BIG_YACHTS;
DTR> SHOW PROCEDURES
Procedures:
        BIG_YACHTS_QUERY                CHEAP           MS_SEARCH
        PHONE_REP       YACHT_SUMMARY

DTR>
```

Note that the DELETE command must end with a semicolon (;).

To be able to recover your procedure if deleted, you should maintain a backup copy (especially if it is a long procedure). Use the DATATRIEVE EXTRACT command to copy your procedure to a command file for backup.

7.12 Protecting Procedures

When you define a procedure, DATATRIEVE stores the procedure definition in your default dictionary directory and creates an access control list for the procedure. DATATRIEVE automatically stores one access control list entry that specifies your username as the only valid identification and grants you C (CONTROL) D (LOCAL_DELETE), E (EXTEND/EXECUTE), H (HISTORY), M (MODIFY), R (READ), S (SEE), U (UPDATE), and W (WRITE) access privileges.

You can modify the access control list to give various types of access privilege to other users. To execute the procedure, a user must have P (PASS_THRU), S (SEE), and E (EXTEND/EXECUTE) privileges.

Using Command Files 8

'his chapter discusses two types of command files:

DATATRIEVE command files that contain *only* DATATRIEVE commands and statements.

DCL command files that contain a list of DCL commands. A DCL command file can also invoke DATATRIEVE and contain DATATRIEVE commands and statements.

'his chapter describes how to use DATATRIEVE command files and provides ome points to keep in mind when using DCL command files with DATATRIEVE nd the CDD.

.1 Using DATATRIEVE Command Files

)ATATRIEVE command files are similar to procedures. Both contain fixed equences of DATATRIEVE commands and statements, and both allow you to xecute frequently used operations. There are, however, some differences between ommand files and procedures:

You invoke a command file using an at sign (@) before its name, and you invoke a procedure using a colon (:) or EXECUTE before its name.

Command files reside outside DATATRIEVE in a VMS directory, and procedures are stored in the Common Data Dictionary. Because of this, command files have the added security of VMS file protection and access control lists, while procedures can take advantage of CDD history and access control lists.

- You can display the commands and statements in a command file as they execute by issuing the SET VERIFY command. SET VERIFY does not work with procedures.
- You cannot invoke command files inside a compound statement, such as a FOR statement or a BEGIN-END block. You can execute procedures inside compound statements.

Use command files for the following purposes:

- You can create a DTR$STARTUP command file (see Chapter 1) that contains any DATATRIEVE commands and statements you want executed each time you use DATATRIEVE.
- You can create and then invoke command files to add definitions of dictionary objects to the Common Data Dictionary. You can change the definitions of these dictionary objects by editing them with the DATATRIEVE Editor.
- You can easily move DATATRIEVE procedures and data definitions around the CDD from within DATATRIEVE with command files. Use the EXTRACT command to copy a dictionary object to a command file. Use the SET DICTIONARY command to move to the desired CDD dictionary and invoke the command file. Use the same technique with EXTRACT ALL to copy all dictionary objects from one dictionary to another.
- You can use command files to aid in debugging DATATRIEVE procedures. By initially creating procedures as command files and using SET VERIFY, you can see statements and commands as they are processed. You can also convert existing procedures to command files for the same purpose. See the chapter on using procedures and compound statements in the *VAX DATATRIEVE Handbook* for details.
- You can use the EXTRACT command to create command files as backup files to maintain the integrity of the CDD. You can use your backup files of domain, record, file, and procedure definitions if something happens to corrupt the CDD and you need to restore the definitions it previously contained

1.1.1 Creating a DATATRIEVE Command File

You create a command file with a text editor. You can edit command files with either the DATATRIEVE Editor or any of the various VMS editors.

To create a new command file or edit an existing command file from within DATATRIEVE, follow these steps:

1. Invoke the DATATRIEVE Editor with the EDIT command.
2. Delete the contents of the editing buffer, which will be the last statement or command DATATRIEVE executed.
3. To create a new command file, enter the DATATRIEVE commands and statements just as you would at the DATATRIEVE prompt. (Do not include the prompts DTR>, CON>, DFN>, or RW>.) To edit an existing command file, use the INCLUDE file-spec editing command to copy the command file into the editing buffer.
4. To store the new or changed command file, use the WRITE file-spec editing command to copy the contents of the buffer to a VMS file.
5. Finally, either exit or quit from the DATATRIEVE Editor. If you use EXIT, DATATRIEVE will execute the command file without requiring an explicit invocation command.

To use any of the VMS editors, you must exit from DATATRIEVE and create or edit the command file as you would any other file.

Note that by default DATATRIEVE expects a file type of either .COM or .DTR. If you use either file type, you do not have to supply it when you invoke the command file.

1.1.1 ADT, EDIT, and SET GUIDE in Command Files -- If you put an ADT, EDIT, or SET GUIDE command in a command file, DATATRIEVE puts you into the requested mode and waits for you to respond to the prompt. When you exit from the Editor, Guide Mode, or ADT, DATATRIEVE executes the next line in the command file. If that line is a valid response to the prompts of the Editor, Guide Mode, or ADT, but not a valid DATATRIEVE command or statement, DATATRIEVE ignores it, displays an error message on your terminal, and returns you to command level.

1.1.2 Comments in Command Files -- You can include comments in a command file by placing an exclamation point (!) before each comment line. If you issue the SET VERIFY command, the comments appear on your terminal when you invoke the file, along with all the commands and statements in the file.

8.1.2 Invoking a Command File

When you invoke a command file, DATATRIEVE executes each command or statement as if you had entered it directly from your keyboard. If you issue a SET VERIFY command, or if the command file contains a SET VERIFY command, DATATRIEVE displays each command and statement as it executes on your terminal screen. If an error occurs, DATATRIEVE prints an error message and terminates the execution of the command file.

If SET ABORT is in effect, DATATRIEVE returns to command level without executing the rest of the procedure or command file.

If SET NO ABORT is in effect, DATATRIEVE aborts the current statement and then processes any statements and commands remaining in the procedure or command file.

8.1.2.1 Invoking a Command File from Within DATATRIEVE -- From within DATATRIEVE, you invoke a command file stored in a VMS directory by preceding the file specification with an at sign (@). To invoke a command file, you must enter it on a line by itself, for example:

```
DTR> @BIGBOAT.COM
```

If the file type is .COM or .DTR and the file is in your default VMS directory, you need enter only the file name:

```
DTR> @PRT
```

If the command file is in another user's directory, you invoke it by specifying all the necessary information in the following format:

device:[username]filename.type;version

For example:

```
DTR> @DBA2:[WEAVER]BIGBOAT.COM;3
```

To invoke a command file in another user's VMS directory, you must have R (READ) access to that directory.

You cannot invoke command files while you are in ADT or Guide Mode.

You can invoke a command file in response to the RW> prompt of the Report Writer. The file must begin with valid report statements. After you complete the

eport specification in the file with an END_REPORT statement, you can follow he specification with other valid DATATRIEVE commands or statements.

.1.2.2 Invoking a Command File Outside of DATATRIEVE -- You need not nter DATATRIEVE to invoke a command file. You can invoke a command file rom the VMS command level. For example, if DTR32 is your symbol for invok- ıg DATATRIEVE, invoke PRT.COM in your default VMS directory with this ommand line at the system level prompt:

```
DTR32 @PRT
```

ıfter DATATRIEVE executes the last command or statement in the file, you are utomatically returned to the system prompt.

'or DATATRIEVE command files that are run often from VMS level, you can lso define a DCL symbol for the entire command line. For example, the com- ıand line in the previous example can be defined as follows:

```
PRT :== "''DTR32' @PRT"
```

'sers can run DATATRIEVE command files at the VMS command level simply y entering the symbol you define.

.1.3 Sample DATATRIEVE Command File

xample 8-1 shows a command file similar to the YACHT_SUMMARY proce- ıre defined in the chapter on using procedures. This example illustrates both the milarities and the differences between procedures and command files. The com- ıand file here is called YSUM.COM. You invoke it with the command file invoca- on command (@).

ı contrast to the sample procedure, if SET VERIFY is in effect, the sample com- ıand file prints each comment, statement, and command in the file as ATATRIEVE encounters it.

'hen DATATRIEVE processes the statement with the *."YES OR NO" ·ompting value expression, it pauses in the execution of the command file to ait for the user's response to the question, "HAVE YOU ESTABLISHED A OLLECTION?" As in the sample procedure YACHT_SUMMARY, the Boolean :pression CONTAINING checks your response to the question, and, if the sponse contains a letter N anywhere, the command file aborts.

'hen DATATRIEVE encounters the *."OUTPUT DEVICE OR FILE" prompt, pauses again for you to select the device or file for output of the report.

ote that except for the report name, the report produced by the command file is e same as the one produced by the procedure YACHT_SUMMARY.

Example 8-1: Sample Command File Using the Report Writer

```
!
!THIS COMMAND FILE, YSUM.COM, PRODUCES A SUMMARY REPORT OF YACHT DATA
SET ABORT
!THIS REPORT REQUIRES AN ESTABLISHED COLLECTION,
!SORTED BY LOA AND BEAM.
!
!HAVE YOU ESTABLISHED A COLLECTION?
IF *."YES OR NO" CONTAINING "N" THEN
        ABORT "SORRY, NO  COLLECTION."
REPORT ON *."OUTPUT DEVICE OR FILE"
SET REPORT_NAME="SAMPLE REPORT"/"FROM A COMMAND FILE"
SET LINES_PAGE=55, COLUMNS_PAGE=60
PRINT BUILDER, MODEL, LOA, BEAM, PRICE
AT BOTTOM OF LOA PRINT SKIP, "AVERAGE PRICE =",
  AVERAGE (PRICE), SKIP
AT BOTTOM OF REPORT PRINT COL 13,"NUMBER OF BOATS = ",
  COL 33, COUNT, SKIP, "AVERAGE PRICE OF ALL BOATS =",-
  AVERAGE (PRICE)
END_REPORT
```

When you have readied the domain and established the appropriate collection, issue the SET VERIFY command and invoke the command file with an at sign (@) and without the .COM file type. DATATRIEVE prints each command and statement as it is executed.

```
DTR> READY YACHTS
DTR> FIND FIRST 5 YACHTS WITH LOA BETWEEN 36 37 AND PRICE NE 0
[5 records found]
DTR> SORT BY LOA, BEAM
DTR> SET VERIFY
DTR> @YSUM
SET ABORT
!THIS REPORT REQUIRES AN ESTABLISHED COLLECTION,
!SORTED BY LOA AND BEAM.
!
!HAVE YOU ESTABLISHED A COLLECTION?
IF *."YES OR NO" CONTAINING "N" THEN
       ABORT "SORRY, NO  COLLECTION."
Enter YES OR NO: YES
REPORT ON *."OUTPUT DEVICE OR FILE"
SET REPORT_NAME="SAMPLE REPORT"/"FROM A COMMAND FILE"
SET LINES_PAGE=55, COLUMNS_PAGE=60
PRINT BUILDER, MODEL, LOA, BEAM, PRICE
AT BOTTOM OF LOA PRINT SKIP, "AVERAGE PRICE =",
  AVERAGE (PRICE), SKIP
AT BOTTOM OF REPORT PRINT COL 17,"NUMBER OF BOATS = ",
  COL 40, COUNT, SKIP, "AVERAGE PRICE OF ALL BOATS =",-
  AVERAGE (PRICE)
END_REPORT
Enter OUTPUT DEVICE OR FILE: TT:
```

```
                    SAMPLE REPORT                    1-Aug-1985
                FROM A COMMAND FILE                  Page 1

                                          LENGTH
                                           OVER
  MANUFACTURER          MODEL             ALL      BEAM     PRICE

  ISLANDER          36                     36        11     $31,730
  I. TRADER         37                     36        12     $39,500

AVERAGE PRICE =                                             $35,615

  NORTHERN          37                     37        11     $50,000
  IRWIN             37 MARK II             37        11     $36,950
  ALBERG            37 MK II               37        12     $36,951

AVERAGE PRICE =                                             $41,300

                NUMBER OF BOATS =                5
                AVERAGE PRICE OF ALL BOATS =                $39,026
```

.1.4 Invoking a Command File from a Procedure

ou can invoke a command file from a procedure you define with the DEFINE ROCEDURE command.

or example, suppose you create a procedure PICKBOATS to form a collection of oats that cost over $10,000. Within the procedure you can invoke the sample ommand file from the previous section, YSUM, to generate the report. Since ICKBOATS starts off with the SET VERIFY command, the contents of YSUM ut not PICKBOATS) appear automatically on the screen when you execute ICKBOATS.

he procedure PICKBOATS contains some PRINT statements after the YSUM command to illustrate that DATATRIEVE processes all statements in a rocedure before those in a nested command file.

```
R> SHOW PICKBOATS
OCEDURE PICKBOATS
T VERIFY
ADY YACHTS
ND YACHTS WITH PRICE GT 10000 SORTED BY LOA, BEAM
SUM
INT "These lines come AFTER the @YSUM command"
INT "in the PICKBOATS procedure."
INT "Notice that DATATRIEVE processes them"
INT "BEFORE the commands in YSUM."
D_PROCEDURE
```

(continued on next page)

```
DTR> :PICKBOATS
These lines come AFTER the @YSUM command
in the PICKBOATS procedure.
Notice that DATATRIEVE processes them
BEFORE the commands in YSUM.

SET ABORT
!THIS REPORT REQUIRES AN ESTABLISHED COLLECTION,
!SORTED BY LOA AND BEAM.
!
!HAVE YOU ESTABLISHED A COLLECTION?
IF *."YES OR NO" CONTAINING "N" THEN ABORT "SORRY, NO COLLECTION."
Enter YES OR NO: Y
REPORT ON *."OUTPUT DEVICE OR FILE"
SET REPORT-NAME="SAMPLE REPORT"/"FROM A COMMAND FILE"
SET LINES-PAGE=55, COLUMNS-PAGE=60
PRINT BUILDER, MODEL, LOA, BEAM, PRICE
AT BOTTOM OF LOA PRINT SKIP, "AVERAGE PRICE =",
AVERAGE PRICE, SKIP
AT BOTTOM OF REPORT PRINT COL 17, "NUMBER OF BOATS = ",
COL 40, COUNT, SKIP, "AVERAGE PRICE OF ALL BOATS =", AVERAGE PRICE
END_REPORT
Enter OUTPUT DEVICE OR FILE: TT:


                    SAMPLE REPORT              1-Aug-1985
                FROM A COMMAND FILE            Page 1


                                    LENGTH
                                     OVER
  MANUFACTURER          MODEL        ALL      BEAM     PRICE

  EASTWARD              HO           24       09     $15,900

AVERAGE PRICE =                                       $15,900

  IRWIN                 25           25       12     $10,950

AVERAGE PRICE =                                       $10,950

  GRAMPIAN              26           26       08     $11,495
  AMERICAN              26-MS        26       08     $18,895
  WESTERLY              CENTAUR      26       08     $15,245
  TANZER                26           26       09     $11,750
  ALBIN                 79           26       10     $17,900
                          .
                          .
                          .

DTR>
```

t is important to remember that a command file within a procedure is always executed after all the other statements in the procedure.

8.1.5 Invoking a Command File from Another Command File

You can invoke procedures and command files from within a command file. For example, the command file MSMOD readies the YACHTS domain then executes the command file MOD. MOD contains a loop with a FOR statement and a BEGIN-END block of statements that allow the user to modify prices interactively:

```
TR> SET VERIFY
TR> @MSMOD
 MSMOD contains only the next two lines:
EADY YACHTS WRITE
MOD

 MOD contains the following lines:

OR YACHTS WITH RIG = "MS"
  BEGIN
    PRINT
    IF *."Y TO MODIFY, N TO SKIP" CONTAINING "Y"
    THEN MODIFY PRICE ELSE
    PRINT "NO CHANGE"
    IF *."Y TO CONTINUE, N TO ABORT" CONTAINING "N"
    ABORT "END OF PRICE CHANGES"
  END

                                LENGTH
                                 OVER
ANUFACTURER    MODEL      RIG    ALL   WEIGHT BEAM  PRICE

AMERICAN     26-MS        MS      26     5,500  08  $18,950
nter Y TO MODIFY, N TO SKIP: Y
nter PRICE: 19350
nter Y TO CONTINUE, N TO ABORT: Y
EASTWARD     HO           MS      24     7,000  09  $15,900
nter Y TO MODIFY, N TO SKIP: N
) CHANGE
nter Y TO CONTINUE, N TO ABORT: N
3ORT: END OF PRICE CHANGES
TR> FIND YACHTS WITH RIG = "MS"
5 records found]
TR> SELECT
TR> PRINT

                                LENGTH
                                 OVER
NUFACTURER     MODEL      RIG    ALL   WEIGHT BEAM  PRICE

MERICAN      26-MS        MS      26     5,500  08  $19,350

TR>
```

Recall that you cannot use commands in compound statements. Because the at sign (@) is a command (the Invoke Command File command), you cannot use the at sign within a compound statement. In other words, you cannot invoke command files from within compound statements.

When embedding command files within other command files (also called nesting), do not allow a command file to invoke itself, either directly or indirectly, doing so may create an infinite loop.

8.1.6 Aborting Command Files

To abort a command file that contains an error, include an ABORT statement in the file. If the responses meet the abort conditions and SET ABORT is in effect, DATATRIEVE aborts the command file and prints the message specified for the ABORT statement. If SET NO ABORT is in effect, DATATRIEVE aborts the command or statement that contains the ABORT but continues to execute the remaining commands and statements in the file.

8.1.7 Maintaining Command Files

VMS directories, not the CDD, store the command files. If you adopt the convention of using .COM as the file type for command files, you can display the names of your command files on your terminal by requesting a directory listing of *.COM at the DCL command level. You can adopt any other convention you wisl and use the wildcard in the same manner.

```
$ DIRECTORY *.COM

YSUM.COM;1      WIDTH.COM;1      SAMPLE.COM;1
```

You can display the contents of a command file with the DCL TYPE command:

```
$ TYPE SAMPLE.COM
```

You can delete a command file from your directory with the DCL DELETE command:

```
$ DELETE SAMPLE.COM;*
```

8.1.8 Protecting Command Files

Command files have more protection from unauthorized use than DATATRIEVI procedures do. Each procedure does have its own access control list that DATATRIEVE checks when someone invokes the procedure, but whoever has access to the CDD has read and write access to procedures. Command files, on the other hand, have VMS protection. To prevent people from using your command files, you can change the VMS protection to deny R (READ) access to others.

2 Using DCL Command Files

ATATRIEVE lets you execute commands, statements, procedures, and com-
and files from DCL command level. You can do this by defining a symbol to
voke DATATRIEVE and following the symbol with any legal DATATRIEVE
mmand. (See Chapter 1 for details.)

ou can embed such a DCL command line in a DCL command file. When you do
is, you have to keep in mind some special precautions. This section describes
ose precautions.

2.1 Reassigning SYS$INPUT in Command Files That Require Interactive Input

r default, DCL command procedures expect any data or other input to come
om the command procedure itself. The DCL command file will not work if it
vokes DATATRIEVE and DATATRIEVE requires input from the terminal.
cause of this, you must reassign the logical name SYS$INPUT.
ATATRIEVE requires input from the terminal when it:

Executes any command, statement, procedure, or command file that prompts the user for input

Uses a forms product

reassign the logical name SYS$INPUT so the DCL command procedure
pects input from the terminal, precede the command that invokes
ATATRIEVE with this ASSIGN command:

```
SIGN/USER_MODE SYS$COMMAND SYS$INPUT
```

the symbol to invoke DATATRIEVE is DTR32, and the procedure
OMPTER requires terminal input, that means the DCL command file must
ntain the following lines:

```
ASSIGN/USER_MODE SYS$COMMAND SYS$INPUT
DTR32 EXECUTE PROMPTER
```

2.2 Command Files with an Invalid CDD$DEFAULT Can Damage the CDD

a DCL command file uses an equivalence to the logical name CDD$DEFAULT
at is not a valid CDD path name, subsequent commands in the file could delete
otherwise damage CDD directories.

There are several ways to assign an invalid CDD$DEFAULT:

- You might misspell the name of a dictionary directory in assigning CDD$DEFAULT; for example, you might type CDD$TOP.SOLES instead of CDD$TOP.SALES.
- You might delete the directory you specified as the CDD$DEFAULT and forget to specify another CDD$DEFAULT.
- You might assign a logical name to CDD$DEFAULT. CDD$DEFAULT is already a logical name. The CDD translates only one logical name, so it assumes that the logical name you assigned to CDD$DEFAULT is a full or relative path name, not another logical name.

When the DMU utility cannot use the CDD$DEFAULT you have defined, it sets your default dictionary directory to CDD$TOP, the only node certain to exist in every CDD. This action can cause a problem if it occurs during execution of a command file. For example, consider what might happen when the following com mand file executes:

```
$ ON WARNING THEN EXIT
$ RUN SYS$SYSTEM:DMU
DELETE/ALL
EXIT
```

Before DMU can execute the DELETE/ALL command, it must find the default directory. If it finds an invalid default directory, it issues DMU and CDD error messages and sets the default directory to CDD$TOP. The DCL command ON WARNING THEN EXIT does not stop execution of the command file because DMU continues to execute in spite of the error. Instead of deleting every direc- tory and object under the default directory, DMU deletes everything under CDD$TOP. Similar, but less drastic results can occur in any command file that has an invalid CDD$DEFAULT and attempts to alter the dictionary in any way.

If a command file invokes DATATRIEVE and CDD$DEFAULT is incorrectly defined, no DATATRIEVE commands can execute. You cannot run DATATRIEVE when CDD$DEFAULT is incorrectly defined. However, you can run DATATRIEVE if CDD$DEFAULT is not defined at all or if you type DEASSIGN CDD$DEFAULT. In this case, DATATRIEVE sets your default dic tionary directory to CDD$TOP and you run the risk of unintentionally altering CDD$TOP.

There are several safeguards you can implement to help protect the CDD against this type of error:

- Check the translation of CDD$DEFAULT before executing a command file that alters the dictionary.

Limit everyone except the system manager or data administrator to PASS_THRU privilege at CDD$TOP. Then, if users unintentionally set their default directories to CDD$TOP, they are unlikely to have sufficient privileges to alter the dictionary.

Always define CDD$DEFAULT as a path name, not a logical name.

Use full path names rather than relative path names in command files that alter the dictionary.

Avoid the use of powerful commands, such as DELETE/ALL, in command files.

Using DATATRIEVE Variables 9

variable is a symbol whose value can change as you execute a program. You can ıe the letter A as a variable, for instance. The name of the variable stays the ıme, but its value can change as DATATRIEVE acts upon it. You use variables DATATRIEVE:

To assign values to fields in STORE and MODIFY statements

As counters in FOR, REPEAT, and WHILE loops

As conditional values in Boolean expressions

To specify field names that would otherwise be ambiguous

1 Declaring Variables

ɔu declare a variable with a statement in this form:

ECLARE variable-name variable-definition.

ıe variable name is the name you give to the variable. The variable definition nsists of field definition clauses. When you declare a variable, you can use any the DATATRIEVE definition clauses except OCCURS and REDEFINES. You ıst include at least one PIC, COMPUTED BY or USAGE clause. You can also e the QUERY-HEADER, EDIT-STRING, SIGN, MISSING VALUE, and EFAULT VALUE clauses.

declare the variable A to be a three-digit numeric value with an initial value of ·o, you use this variable name and variable definition:

```
ι> DECLARE A PIC 999.
ι> A = 0
```

If you print the variable, it looks like this:

```
DTR> PRINT A

 A

000
```

You can define two kinds of variables:

- Local variables
- Global variables

You use the DECLARE statement to define both local and global variables. A variable you define within a BEGIN-END statement is a local variable, and you can use it only within that statement. A variable you define at DATATRIEVE command level is a global variable. It remains in your workspace until you release it or exit from DATATRIEVE. Use the assignment statement (variable = value) to set the variable to a particular value. For example:

```
DTR> DECLARE BIG PIC X(5).
DTR> BIG = "YES "
DTR> PRINT BIG

   BIG

    YES
```

The initial value for variables in numeric fields is 0. In alphanumeric strings, it is spaces. These are the default values if you do not specify a different default value or missing value.

You can also use a date field as a variable, as in this example:

```
DTR> DECLARE Y USAGE DATE EDIT_STRING DD-MMM-YY.
DTR> Y = "TODAY"
DTR> PRINT Y

    Y

19-May-84
```

.2 Local Variables

ou define local variables with DECLARE statements entered in BEGIN-END nd THEN statements. The local variable has an effect only within the clause or tatement in which you declare it.

ı the following example, the local variable declared in the inner statement super- edes one with the same name declared in the outer statement. Notice that the ifferent value or different data type assigned to the inner variable has no effect ı the value of the variable in the outer statement. Note also that neither local ariable exists when DATATRIEVE finishes executing the compound statements ontaining them both:

```
TR> SET NO PROMPT
TR> BEGIN
)N>   DECLARE X PIC XXX.
)N>   X = "TOP"
)N>   PRINT X
)N>     BEGIN
)N>       DECLARE X PIC 9.99.
)N>       X = 1.23
)N>       PRINT X
)N>     END
)N>   PRINT X
)N> END

:

)P

.

23


P

R>
```

3 Global Variables

ıppose you want to assign to each boat in YACHTS a new price that is two- irds of the present price. Using a COMPUTED BY clause in a global variable, u can apply a single formula to every yacht, as in the next example. Use the ECLARE statement to create the variable. Use a COMPUTED BY clause with

a value expression to calculate the changed values:

```
DTR> READY YACHTS MODIFY
DTR> DECLARE FIRE_PRICE COMPUTED BY PRICE/1.5
CON> EDIT_STRING IS $99,999.99.
DTR> FIRE_PRICE = 0
DTR> FOR FIRST 5 YACHTS PRINT BOAT, FIRE_PRICE

                                LENGTH
                                 OVER                          FIRE
MANUFACTURER    MODEL     RIG    ALL   WEIGHT BEAM  PRICE      PRICE

 ALBERG         37 MK II  KETCH  37    20,000  12   $36,951  $24,634.00
 ALBIN          79        SLOOP  26     4,200  10   $17,900  $11,933.33
 ALBIN          BALLAD    SLOOP  30     7,276  10   $27,500  $18,333.33
 ALBIN          VEGA      SLOOP  27     5,070  08   $18,600  $12,400.00
 AMERICAN       26        SLOOP  26     4,000  08    $9,895  $06,596.67
DTR>
```

The variable FIRE_PRICE declared at DATATRIEVE command level remains in the workspace throughout the session. It changes its value whenever the value of PRICE changes. (See Appendix A for a discussion of context changes.) The variable remains in your workspace until you release the variable with a RELEASE statement or declare another global variable with the same name.

9.4 Using Variables to Assign Values to Fields

You can use variables to assign values to fields in the USING clauses of STORE and MODIFY statements. You cannot, however, use a variable to respond to a prompt for a field value, whether the prompt is the result of the STORE or MODIFY statement or of a prompting value expression in an assignment statement.

In the USING clause of the STORE and MODIFY statements, you can supply values for fields by using value expressions on the right side of assignment statements. In some circumstances, you can use variables in those assignments to control the uniformity of input data.

In this example. WORK is a domain you want to contain uniform names. The domain is indexed on WHO and allows duplicates:

```
DTR> SHOW WORK_REC
RECORD WORK_REC
 USING
01 TOP.
        03 JOB PIC X(15).
        03 RESPONSIBLE_PERSON PIC X(4)
           QUERY_NAME WHO.
```

NAME_TABLE translates the varying inputs into uniform values to store in the vork domain:

```
TR> SHOW NAME_TABLE
ABLE NAME_TABLE
DIT_STRING IS X(16)
                :       ED
D               :       ED
M               :       ED
                :       ED
                :       FRED
H               :       FRED
RED             :       FRED
                :       FRED
                :       RICK
                :       RICK
BL              :       RICK
ICK             :       RICK
L               :       RICK
LSE "NOT A VALID NAME"
ND_TABLE
```

n the following STORE statement, the USING clause uses the variable 'ERSON with a prompting value expression for the responsible person. The table ranslates the value supplied to that prompt and stores the uniform results in the eld WHO:

```
TR> SET NO PROMPT
TR> DECLARE PERSON PIC X(16).
TR> READY WORK WRITE
TR> REPEAT 3 STORE WORK USING
)N> BEGIN
)N>     JOB = *.JOB
)N>     PERSON = *.WHO
)N>     WHO = PERSON VIA NAME_TABLE
)N> END
nter JOB: CLEANING
nter WHO: Σ
nter JOB: DRYING
nter WHO: FR
nter JOB: SELLING
nter WHO: R
TR> PRINT WORK

                      RESPONSIBLE
     JOB                PERSON

.EANING             ED
:YING               NOT A VALID NAME
:LLING              RICK

'R>
```

9.5 Changing the Value of a Variable

You can change the value of a variable with an assignment statement, using any DATATRIEVE value expression on the right side of the statement. You can also use a prompting value expression to change the value of a variable. This example declares a variable and changes that value first with an assignment statement and then with a prompting value expression:

```
DTR> DECLARE X PIC XXX.
DTR> X = 0
DTR> PRINT X

 X

0

DTR> X = "ABC"; PRINT X

 X

ABC

DTR> X = *."VALUE FOR X"
Enter VALUE FOR X: LIP
DTR> PRINT X
 X

LIP

DTR>
```

9.6 Using Context Variables

DATATRIEVE provides a different kind of variable from the ones previously dis cussed in this chapter. This is called a **context variable.** Instead of storing values context variables serve as labels that identify a record stream to DATATRIEVE You assign context variables to be temporary names of particular record stream: In this way you can make clear the domain from which a record stream originate or you can create two different record streams based on the same domain.

In most cases. DATATRIEVE will know to what record stream a field name applies without needing context variables. For example, DATATRIEVE does no need context variables in the following store statement:

```
DTR> ! First, define a domain that will hold a subset
DTR> ! of YACHTS records, namely, those that
DTR> ! cost more than $20,000.
DTR> DEFINE DOMAIN RITZY_ONES USING YACHT ON RITZY;
DTR> DEFINE FILE FOR RITZY_ONES
DTR> READY RITZY_ONES WRITE
```

```
)TR> ! The FOR statement includes a STORE USING statement
)TR> ! to store the desired records in RITZY_ONES.
)TR> ! Note that you don't need context variables.
)TR> FOR YACHTS WITH PRICE > "$20,000"
:ON> STORE RITZY_ONES USING
:ON>   BEGIN
:ON>       TYPE = TYPE
:ON>       PRICE = PRICE
:ON>   END
```

n this example, DATATRIEVE assigns the values of TYPE and PRICE of all oats in YACHTS that cost more than $20,000 records in to a new domain called ITZY_ONES.

hould you so choose, however, you can use context variables to identify record treams and to qualify field names. This makes your statements and procedures ss ambiguous and easier to maintain. For example, you can perform the preceding store operation as follows:

```
TR> FOR Y IN YACHTS WITH PRICE > "$20,000"
ON> STORE R IN RITZY_ONES USING
ON>   BEGIN
ON>       R.TYPE = Y.TYPE
ON>       R.PRICE = Y.PRICE
ON>   END
```

certain cases, however, you must use context variables to identify a record tream explicitly. When you need to access the same domain two or more times in ne statement, or when you need to compare record streams from the same omain, you must use context variables. In all other cases, you can use the omain name for qualifying field names, or else DATATRIEVE resolves the context automatically (as in the first example above).

ut when you must establish two record streams from the same domain, or when ou cross a domain over itself, you use context variables to label different record reams. By qualifying each field name with the context variable and a period (.), ou indicate clearly to DATATRIEVE how to evaluate field references. In the ove example, DATATRIEVE looks to the R stream to evaluate R.TYPE and to e Y stream to evaluate Y.TYPE. Thus DATATRIEVE allows you to create two r more) record streams from the same domain without confusing or mixing cords.

or example, assume that you are interested in finding the average payroll of ch department in the PERSONNEL domain. You need to access all the records PERSONNEL twice, once to group them by department and again to compute e average salary.

Using context variables allows you to distinguish the references to PERSONNEL so that you can perform both operations within the same statement:

```
DTR> FOR D IN PERSONNEL REDUCED TO DEPT
CON> PRINT D.DEPT, AVERAGE SALARY OF S IN PERSONNEL WITH -
CON> S.DEPT = D.DEPT

      AVERAGE
DEPT  SALARY

C82   $40,493
D98   $42,244
E46   $39,658
F11   $37,892
G20   $39,185
T32   $39,144
TOP   $75,892
```

DATATRIEVE reduces the PERSONNEL domain to unique occurrences of the DEPT field. This record stream is identified by the context variable D. Then, for each value of DEPT in the D stream, DATATRIEVE calculates the average salary of all records in the S stream that have matching values for DEPT.

For another example, assume that you wish to find how much of the company's workforce each department employs. You can perform the query in the following manner:

```
DTR> FOR D IN PERSONNEL REDUCED TO DEPT
CON> PRINT D.DEPT, 100 * ((COUNT OF PER IN PERSONNEL -
CON> WITH PER.DEPT = D.DEPT)/ COUNT OF PERSONNEL) ("PCT") USING Z9.9%

DEPT   PCT

C82   21.7%
D98   17.4%
E46    8.7%
F11   17.4%
G20   13.0%
T32   17.4%
TOP    4.3%
```

This statement sets up an inner loop of records from the PERSONNEL domain identified by the context variable PER and an outer loop of records from the san domain identified by the label D. For each group of records with the same depar ment, DATATRIEVE counts the records and divides it by the total number of al records in PERSONNEL. The context variables D and PER allow you to refer t the same records in the PERSONNEL domain twice. Thus you can print out all the department names and count all their members in the same statement.

ection 2.4 contains an example illustrating the use of context variables to com-
are record streams from the same domain. Appendix A contains more examples
context variables and presents detailed information about how DATATRIEVE
solves context.

Part 4
Optimizing DATATRIEVE

Restructuring Data 10

ıis chapter describes how to create new domains with data from existing ones. ɔu might do this to:

- Add new fields to the record definition associated with the domain
- Change field definitions to affect the values stored in the data file
- Rearrange the fields in the record definition
- Combine data from two or more domains
- Create a copy of a domain for testing
- Change the file organization
- Change the index structure (key fields)
- Create a domain that contains a subset of records contained in another domain

ɔw you create the new domain depends on whether you want to keep the old main. If you want to keep the old domain, follow these three steps when creat-ɾ the new domain:

1. Define the new domain, its record, and its data file.
2. Ready the new domain for WRITE access and the old domain for READ access.
3. Use the DATATRIEVE Restructure statement to transfer field values from the old data file to the new one.

ɾou want to use any old procedures on the new domain, you must edit them hey refer to fields not included in the new domain. Follow your standard

DATATRIEVE editing procedure to make the necessary changes, using:

```
EDIT procedure-name
```

If the old procedures refer only to fields included in the new domain, you need not change the procedures. You can ready the new domain with the old domain name as an alias (READY NEW AS OLD) and execute the old procedures.

If you do not want to keep the old domain, you can still use the old procedures if you follow these steps:

1. Define the new domain (NEW), record (NEW_REC), and file (NEW.DAT).
2. Use the Restructure command to transfer the data from the old domain (OLD) to the new one (NEW).
3. Delete the definition of the old domain (OLD).
4. Enter another domain definition that uses the old domain name (OLD), the new record definition (NEW_REC), and the new data file (NEW.DAT):

   ```
   DTR> DEFINE DOMAIN OLD USING NEW_REC ON NEW.DAT;
   DTR>
   ```

5. Check the old procedures for any references to field names not included in the new record definition, and edit where necessary.

Note

You cannot use DATATRIEVE to restructure DBMS or Rdb domains. You must use RDO to restructure Rdb domains. You must use the VAX DBMS DDL compilers and DBO utility to restructure databases. However, you can store data from DBMS or Rdb domains in RMS domains.

10.1 A Sample Domain

PROJECTS is a sample domain you can create to practice restructuring:

```
DTR> SHOW PROJECTS, PROJECT_REC
DOMAIN PROJECTS USING PROJECT_REC ON PROJECT;
RECORD PROJECT_REC
01 PROJECT_REC.
   03  PROJ_CODE    PIC 9(3)  QUERY_NAME IS CODE.
   03  PROJ_NAME    PIC X(10) QUERY_NAME IS NAME.
   03  MANAGER_NUM  PIC 9(5)  QUERY_NAME IS NUM.
;
```

he data file PROJECT.DAT is a sequential file and contains these records:

```
TR> PRINT PROJECTS

OJ     PROJ      MANAGER
DE      NAME        NUM

2  GROUNDS       00006
5  BUILDING 2    00003
8  SHED          00002
8  RESEARCH      00006
7  PUB REL       00008
3  MATERIALS     00002

R>
```

0.2 Adding Fields to a Record Definition

o create a new domain with two fields added to PROJECT_REC, you follow ese steps:

. Define a new domain:

```
DTR> DEFINE DOMAIN NEW_PROJECTS
DFN>  USING NEW_PROJECT_REC ON NEWPROJ;
```

. Edit the record definition to change the name of the record and add the desired field definitions. Note that with EDIT_BACKUP in effect, DATATRIEVE does not delete the original record definition when you exit the edit buffer.

```
DTR> EDIT PROJECT_REC
1        REDEFINE RECORD PROJECT_REC USING
* c
REDEFINE RECORD NEW_PROJECT_REC USING
01 NEW_PROJECT_REC.
   03  PROJ_CODE        PIC 9(3) QUERY_NAME IS NUM.
   03  PROJ_NAME        PIC X(10) QUERY_NAME IS NAME.
   03  PROJ_COST        PIC 9(6)V99 EDIT_STRING IS $$$,$$9.99.
   03  MANAGER_NUM      PIC 9(5).
   03  MGR_NAME         PIC X(15).
;
* exit
[Record NEW_PROJECT_REC is 40 bytes long]

DTR>
```

. Define a new data file for NEW_PROJECTS. This example creates an indexed file to replace the sequential file associated with PROJECTS:

```
DTR> DEFINE FILE FOR NEW_PROJECTS KEY=PROJ_CODE
DTR>
```

You are now ready to transfer the data from the old domain to the new one.

10.3 Entering Data in the New File

To transfer data from the old domain to the new one, you must first ready both domains. Ready the new domain for WRITE or EXTEND access, and ready the old one for READ access. Then use the Restructure statement to transfer the data:

```
DTR> READY NEW_PROJECTS WRITE
DTR> READY PROJECTS
DTR> NEW_PROJECTS = PROJECTS
DTR>
```

For each field name in NEW_PROJECT_REC that matches a field name in PROJECTS_REC, the Restructure statement transfers field values from each record in PROJECTS to a record in NEW_PROJECTS. For a field in the new record definition that does not match a field in the old one, DATATRIEVE initializes the field according to its data type and its field definition.

If the field has a DEFAULT VALUE clause, DATATRIEVE initializes the field with the default value. If the field has a MISSING VALUE clause and no DEFAULT VALUE clause, DATATRIEVE initializes the field with the missing value. If the field has neither a DEFAULT VALUE clause nor a MISSING VALUE clause, DATATRIEVE initializes a numeric field as 0 and an alphabetic or alphanumeric field as spaces.

The data file associated with your new domain now has records in it. When you display the contents of the new domain on your terminal, you can see the two new fields and the same values contained in the PROJECTS domain:

```
DTR> PRINT NEW_PROJECTS

PROJ    PROJ        PROJ     MANAGER      MGR
NUM     NAME        COST       NUM        NAME

002  GROUNDS          $0.00  00006
005  BUILDING 2       $0.00  00003
008  SHED             $0.00  00002
018  RESEARCH         $0.00  00006
037  PUB REL          $0.00  00008
073  MATERIALS        $0.00  00002

DTR>
```

0.4 Creating Record Subsets

ou can create the new domain from a subset of the old domain's records. You ecify the limiting conditions in the RSE of the Restructure statement. For ample, you can limit a domain to the projects of two managers:

```
R> READY NEW_PROJECTS WRITE
R> READY PROJECTS
R> NEW_PROJECTS = PROJECTS WITH MANAGER_NUM EQ 2, 6
R> PRINT NEW_PROJECTS

OJ     PROJ          PROJ      MANAGER       MGR
M      NAME          COST        NUM         NAME

2   GROUNDS          $0.00   00006
8   SHED             $0.00   00002
8   RESEARCH         $0.00   00006
3   MATERIALS        $0.00   00002

R>
```

ote that the Restructure statement relies on record definitions having the same ld names. If you want to change field names, you can either edit the record finition after the restructure operation or use a STORE USING statement stead of the Restructure statement. The chapter on Designing Better Records ntains an example of STORE USING to restructure data.

0.5 Combining Data from Two or More Domains

nother reason for creating a new domain is to combine the data from two or ore existing domains. If you frequently use the same CROSS clause to form cord streams and you cannot use a view domain because you need to store cords in the domain, you can define a new domain to meet your needs.

r example, when you enter data in the file of NEW_PROJECTS, you can also clude the names of the managers from another domain, MANAGERS:

```
t> SHOW MANAGERS, MANAGER_REC
IAIN MANAGERS USING MANAGER_REC ON MGR;
CORD MANAGER_REC USING
 MANAGER.
 03  MANAGER_NUM          PIC 9(5).
 03  MGR_NAME             PIC X(8).

t>
```

splaying the records from MANAGERS shows that values in the field ANAGER_NUM correspond to the values in the MANAGER_NUM field in the

domain PROJECTS:

```
DTR> READY MANAGERS; PRINT MANAGERS

MANAGER       MGR
  NUM         NAME

 00002  BLOUNT
 00003  GERBLE
 00005  GORFF
 00006  PUFFNER
 00008  FEBNELL

DTR>
```

Using a CROSS clause in the RSE of the Restructure statement, you can match MANAGERS records with the corresponding PROJECTS records. The OVER clause allows you to match those records with matching values in the MANAGER_NUM fields. You must ready all three domains to transfer the data from PROJECTS and MANAGERS to NEW_PROJECTS:

```
DTR> READY NEW_PROJECTS WRITE
DTR> READY PROJECTS
DTR> READY MANAGERS
DTR> NEW_PROJECTS = PROJECTS CROSS MANAGERS OVER MANAGER_NUM
DTR>
```

Displaying the records in NEW_PROJECTS shows the result of the Restructure with a CROSS clause in the RSE. Notice that the value of PROJ_COST in each record is 0; the field did not exist in either of the source domains:

```
DTR> PRINT NEW_PROJECTS

PROJ    PROJ          PROJ      MANAGER       MGR
NUM     NAME          COST        NUM         NAME

002  GROUNDS          $0.00   00006  PUFFNER
005  BUILDING 2       $0.00   00003  GERBLE
008  SHED             $0.00   00002  BLOUNT
018  RESEARCH         $0.00   00006  PUFFNER
037  PUB REL          $0.00   00008  FEBNELL
073  MATERIALS        $0.00   00002  BLOUNT

DTR>
```

10.6 Using the Alias Clause to Restructure a Domain

You can use the Alias clause to restructure a domain. When you use this method you can make use of the difference between the record definition in the CDD and

:he record definition controlling a readied domain in your workspace. For exam-
ɔle, when you ready YACHTS as OLD_YACHTS, the record definition YACHT is
ıssociated with the data file YACHT.DAT. If you then edit and redefine the for-
nat of the record definition YACHT, this change to the record is not associated
vith the readied domain (OLD_YACHTS). It is only associated with YACHTS the
ıext time you ready the domain.

This method lets you make use of the difference between the record definition
n the CDD and the record definition controlling a readied domain in your
vorkspace. The change in the record definition does not take effect until you use
he FINISH command to finish the domain and the READY command to ready it
gain. Simply readying the domain again does not activate the new record
efinition.

You can make use of this fact if you want to change a record definition or change
he type of file organization of a domain's data file. Follow these steps to change
he record definition or file type without redefining the domain. In both cases, you
efine a new data file and transfer the data with the Restructure statement:

1. Ready the domain as an alias:

```
DTR> READY YACHTS AS OLD_YACHTS
DTR> SHOW READY
Ready sources:
   OLD_YACHTS:  Domain, RMS sequential, protected read
          <CDD$TOP.INVENTORY.YACHTS;1>
No loaded tables.

DTR>
```

2. Change the record definition with the EDIT record-path-name command, creating a later version of the same record definition.

3. Define a new data file for the domain. This creates a new version of the file associated with the readied domain but does not interfere with the link between the domain you already readied and the original version of the data file. Do not use the SUPERSEDE option of the DEFINE FILE command:

```
DTR> DEFINE FILE FOR YACHTS KEY = TYPE
DTR>
```

4. Ready the domain as a different alias and specify the WRITE access mode. This READY command uses the new version of the record definition and

opens the new data file created by the DEFINE FILE command:

```
DTR> READY YACHTS AS NEW_YACHTS WRITE
DTR> SHOW READY
Ready sources:
   NEW_YACHTS:  Domain, RMS indexed, protected write
           <CDD$TOP.INVENTORY.YACHTS;1>
   OLD_YACHTS:  Domain, RMS sequential, protected read
           <CDD$TOP.INVENTORY.YACHTS;1>
No loaded tables.

DTR>
```

5. Now use the Restructure statement to move the data from the original data file to the new one. DATATRIEVE transfers data from fields in the original data file into fields with the same names in the new data file.

```
DTR> NEW_YACHTS = OLD_YACHTS
DTR>
```

10.7 Changing the Organization of a Data File

You can also use the Alias clause of the READY command to change the organization of a data file associated with a domain. This example replaces the indexed data file associated with YACHTS with a sequential data file:

```
DTR> READY YACHTS AS OLD
DTR> DEFINE FILE FOR YACHTS
DTR> READY YACHTS AS NEW WRITE
DTR> NEW = OLD
DTR> FIND NEW
[113 records found]
DTR>
```

10.8 Further Examples of Restructuring Domains

This example defines an indexed file for a new domain based on FAMILY_REC. The records of the source domain, FAMILIES, are in a sequential file. The complex Boolean expression in the RSE of the Restructure statement limits the number of records transfered to the new domain. The new domain contains the records of only those families who have no kids younger than 15:

```
DTR> READY FAMILIES
DTR> DEFINE DOMAIN NEWFAMS USING FAMILY_REC ON FAMS;
DTR> DEFINE FILE FOR NEWFAMS MAX, KEY = MOTHER (DUP)
DTR> READY NEWFAMS WRITE
DTR> NEWFAMS = FAMILIES WITH NOT ANY KIDS WITH AGE LE 15
DTR> FIND NEWFAMS
[8 records found]
DTR> FIND FAMILIES
[16 records found]
DTR>
```

The next example defines a new domain called YACHTS_PRICE_LIST, which contains only the fields TYPE and PRICE from the old YACHT record definition. The DEFINE RECORD command shortens the length of the MODEL field from ten to eight bytes and provides a MISSING VALUE edit string for the PRICE field. The FIND statements following the Restructure statement check the number of records transferred and the accuracy of the transfer (with the CROSS clause in the second FIND). The PRINT statement shows the effect of the new MISSING VALUE edit string:

```
TR> DEFINE DOMAIN YACHTS_PRICE_LIST USING YPL_REC ON YPL.DAT;
TR> DEFINE RECORD YPL_REC USING
FN> 01 BOAT.
FN>    03 TYPE.
FN>       05 BUILDER PIC X(10).
FN>       05 MODEL PIC X(8).
FN>    03 PRICE PIC 9(5) MISSING VALUE IS 0
FN>       EDIT_STRING $$$,$$$?"NOT LISTED".
FN> ;
Record is 23 bytes long.]
TR> DEFINE FILE FOR YACHTS_PRICE_LIST KEY = TYPE
TR> READY YACHTS_PRICE_LIST AS YPL WRITE
TR> READY YACHTS
TR> SHOW READY
eady sources:
  YACHTS:  Domain, RMS indexed, protected read
           <CDD$TOP.DTR32.WAJ.YACHTS;1>
  YPL:  Domain, RMS indexed, protected write
           <CDD$TOP.DTR32.WAJ.YACHTS_PRICE_LIST;1>
 loaded tables.

R> YPL = YACHTS WITH LOA GT 35
R> FIND YACHTS WITH LOA GT 35
3 records found]
R> FIND YPL
3 records found]
R> FIND A IN YPL CROSS B IN YACHTS OVER
ooking for field name]
N> TYPE WITH A.PRICE NE B.PRICE
 records found]
R> FIND YPL WITH PRICE MISSING
2 records found]
R> PRINT FIRST 3 CURRENT

UILDER     MODEL      PRICE

OCK I.     40         NOT LISTED
BOT        36         NOT LISTED
WN EAST    38         NOT LISTED

R>
```

10.9 Better Data Organization

DATATRIEVE provides you with many alternative strategies for record design. You can save design time and processing time by choosing the best strategy for your data. The chapter on designing better records offers suggestions for organizing data efficiently at the beginning of the data design process and for reorganizing existing data in the most efficient ways. It contains examples of reorganizing data with both the Restructure and STORE...USING statements.

Designing Better Records 11

'AX DATATRIEVE offers great flexibility in defining records, tables, and ariables. There are usually several alternative strategies for organizing your data ɔ make data access easier and faster. Before deciding how to define your data, ·y to determine the types of queries and reports that you want to produce. This hapter offers some suggestions for organizing your data in the most efficient 'ays.

'ou may also want to change the organization of existing data. This chapter pro- ides examples using DATATRIEVE's Restructure and STORE USING state- ıents to reorganize the data.

eep the following points in mind when organizing or reorganizing data:

- Consider using flat records rather than hierarchical records for greater ease of accessing data.

  Define several related records, rather than one large record containing every field.

  Choose keys carefully to optimize performance for accessing data and joining related records.

  Use tables to validate data, to make data entry easier, and to use as little storage space as possible.

  Use virtual (COMPUTED BY) fields wherever possible to save storage and make data entry easier.

1.1 Flat Records and Hierarchical Records

hen defining a record, you can choose to use lists (hierarchies) or to break off ch list item into separate records (flat files). It is usually easier to access data in ıt files than in hierarchical files. This point can be illustrated with the sample ımain FAMILIES.

The data stored in FAMILIES could be organized in flat records or in hierarchical records. FAMILIES happens to use a hierarchical record organization, a record containing the repeating list field KIDS. Figure 11-1 illustrates the structure of the record FAMILY.

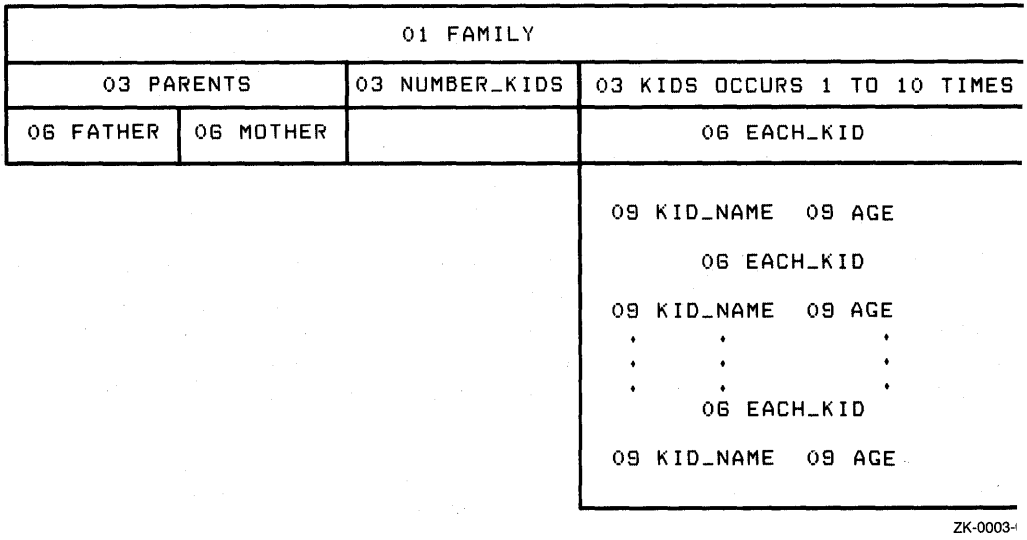


Figure 11-1: Structure of a Hierarchical Record

VAX DATATRIEVE supports lists or hierarchies created using the OCCURS clause in record definitions. You can consider the list field to be a small domain within each record of the large domain. For example, you can view each record in FAMILIES as containing several KIDS "records." To access one of the KIDS "records," you must do two things:

- Identify a specific record in FAMILIES.
- Identify the KIDS "record" within that FAMILIES record.

In the following example, two FOR loops are required to modify ELLEN's age:

```
DTR> FOR FAMILIES WITH FATHER = "JIM" AND MOTHER = "LOUISE"
CON> FOR KIDS WITH KID_NAME = "ELLEN"
CON> MODIFY AGE
Enter AGE: 26
DTR>
```

Sometimes nested FOR loops are not sufficient to access data stored in a list. If you want to sort all the records in FAMILIES by the age of the children, you

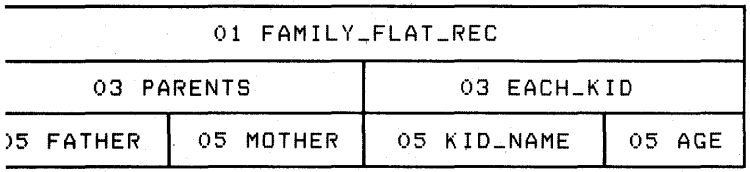
ust first flatten the records in FAMILIES with the CROSS clause:

```
R> FOR FAMILIES CROSS KIDS SORTED BY AGE
N> PRINT PARENTS, KID_NAME, AGE

                          KID
FATHER      MOTHER        NAME      AGE

EARMAN    SARAH         DAVID         0
HN        ELLEN         CHRISTOPHR    0
NIE       ANNE          BRIAN         0
NIE       ANNE          SCOTT         2
M         ANN           RALPH         3
M         ANNE          PATRICK       4
                .
                .
                .
R>
```

ı alternative to this complex syntax and high performance overhead is to orga-
ze the records in a flat file to begin with, as Figure 11-2 shows.



MK-01594-00

gure 11-2: The Structure of a Flat Record

ıis is the complete record definition of FAMILY_FLAT_REC:

```
R> SHOW FAMILY_FLAT_REC
CORD FAMILY_FLAT_REC USING
 FAMILY_REC.
 03 PARENTS.
    05 FATHER PIC X(10).
    05 MOTHER PIC X(10).
 03 EACH_KID.
    05 KID_NAME PIC X(10).
    05 AGE PIC 99
        EDIT_STRING IS Z9.


R>
```

Each record of FAMILY_FLAT has elementary fields for FATHER, MOTHER, KID_NAME, and AGE. This simplifies the task of modifying a child's age. For example, to modify Ellen's age:

```
DTR> MODIFY AGE OF FAMILY_FLAT WITH FATHER = "JIM" AND
CON> KID_NAME = "ELLEN"
```

To sort by the age of children, you can enter:

```
DTR>PRINT FAMILY_FLAT SORTED BY AGE
```

Because FAMILY_FLAT does not have hierarchical records like FAMILIES, VAX DATATRIEVE does not have to flatten records before sorting them. This gives you better performance along with easier access to data. There are additional costs, however, for storing the same parent information with each child in the family. This issue is discussed in Chapter 12.

11.1.1 Restructuring a Hierarchical File to a Flat File

You can use a Restructure statement to convert the records in FAMILIES to FAMILY_FLAT. After defining the domain, record, and file for FAMILY_FLAT, enter the following statements:

```
DTR> READY FAMILIES
DTR> READY FAMILY_FLAT WRITE
DTR> FAMILY_FLAT = FAMILIES CROSS KIDS
```

The Restructure statement contains a CROSS clause so that each child is in a separate record, paralleling the structure of FAMILY_FLAT. A PRINT statement displays the records of FAMILY_FLAT:

```
DTR> PRINT FAMILY_FLAT

                          KID
  FATHER      MOTHER      NAME     AGE

JIM           ANN         URSULA     7
JIM           ANN         RALPH      3
JIM           LOUISE      ANNE      31
JIM           LOUISE      JIM       29
JIM           LOUISE      ELLEN     26
JIM           LOUISE      DAVID     24
JIM           LOUISE      ROBERT    16
JOHN          JULIE       ANN       29
```

(continued on next page)

```
AROLD      SARAH       HAROLD      35
AROLD      SARAH       SARAH       27
DWIN       TRINITA     ERIC        16
DWIN       TRINITA     SCOTT       11

TR>
```

low all of the data for parents and their children has been stored in AMILY_FLAT, but one problem remains. In joining FAMILIES on the st field KIDS, you leave out any records of FAMILIES with parents but no hildren. In fact, there is one such record in FAMILIES:

```
TR> PRINT FAMILIES WITH NOT ANY KIDS

                        NUMBER     KID
 FATHER      MOTHER      KIDS      NAME      AGE

)B          DIDI           0
```

ut this record from FAMILIES is not included in the record stream formed by AMILIES CROSS KIDS, because the KIDS list is empty. As a result, the estructure statement does not store the data about ROB and DIDI in AMILY_FLAT:

```
TR> FIND FAMILY_FLAT WITH FATHER = "ROB"
) records found]
TR>
```

o include records of parents without children in FAMILY_FLAT, you need a eparate storing operation:

```
:R> FOR A IN FAMILIES WITH NOT ANY KIDS
.ooking for statement]
)N> STORE FAMILY_FLAT USING PARENTS = A.PARENTS
```

ow the transfer of data from FAMILIES to FAMILY_FLAT is complete:

```
'R> PRINT FAMILY_FLAT WITH FATHER = "ROB"

                         KID
 FATHER      MOTHER      NAME      AGE

)B          DIDI

'R>
```

11.1.2 Defining Several Smaller Related Records

Though VAX DATATRIEVE lets you define very large records, you may be better off dividing a large record into several smaller related records. If you include all the fields in one large record, you can access any portion of the data by readying only one domain. However, if you need information from only one field, DATATRIEVE still must read through the large record.

Another problem with large, all-inclusive records is that several records can duplicate the same information. Not only is this expensive to store, but you may have problems when updating data if you do not change the information in all the relevant records.

This problem could occur with the FAMILY_FLAT records discussed in the previous section. Parent information is stored in each child's record. If the marital status of the parents should change, each of the children's records would have to be updated. You can avoid this problem by storing parent data in one domain (FOLKS) and children's data in a second domain (CHILDREN).

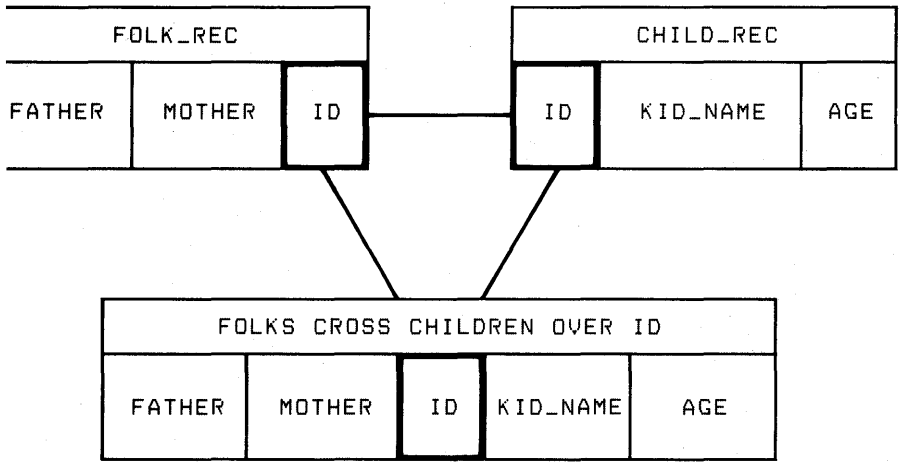
The two domains could each have an ID field, representing an ID assigned to eac set of parents. In the FOLKS domain, you store the ID along with the parents' names. In the CHILDREN domain, you store the parent ID along with the children's names. The record definitions of FOLK_REC and CHILD_REC follow:

```
DTR> SHOW FOLK_REC
RECORD FOLK_REC USING
01 FOLK_REC.
   03 ID PIC 99
         EDIT_STRING IS Z9.
   03 PARENTS.
      05 FATHER PIC X(10).
      05 MOTHER PIC X(10).
;

DTR> SHOW CHILD_REC
RECORD CHILD_REC USING
01 CHILD_REC.
   03 ID PIC 99
         EDIT_STRING IS Z9.
   03 KID_NAME PIC X(10).
   03 AGE PIC 99
         EDIT_STRING IS Z9.
;

DTR>
```

When you need information about both parents and children, you can join the FOLKS records with the CHILDREN records over the common ID field. Figure 11-3 illustrates the result of this relational join. The boldface lines enclose the suggested key fields.



MK-01595-00

ɟure 11-3: Joining FOLKS and CHILDREN with CROSS

.1.3 Restructuring a Large Record into Several Smaller Records

ʎTATRIEVE simplifies the conversion of large records to several smaller ːords. This point is illustrated by converting the larger records of ʎMILY_FLAT to the smaller records of FOLKS and CHILDREN.

ɹth FOLKS and CHILDREN have an ID field that indicates a unique set of parts. Because FAMILY_FLAT has duplicate occurrences for sets of parents (one · each of their children), you need to determine the unique sets of parents in the ːords of FAMILY_FLAT before assigning ID values. Use the REDUCED TO ɹuse in the record selection expression to find the unique values. Then use a 'ORE USING statement to store values in FOLKS, assigning values for ID ɪh RUNNING COUNT. The following DATATRIEVE session uses these ɪtements:

```
ɾ> READY FAMILY_FLAT
ɾ> READY FOLKS WRITE
ɾ> FOR A IN FAMILY_FLAT REDUCED TO PARENTS
ɲ>   STORE FOLKS USING
ɲ>     BEGIN
ɲ>       ID = RUNNING COUNT
ɲ>       FATHER = A.FATHER
ɲ>       MOTHER = A.MOTHER
ɲ>     END
```

this example shows, the STORE USING statement is another way to ɪtructure a domain. A PRINT statement displays the records in the new

domain FOLKS:

```
DTR> PRINT FOLKS

ID   FATHER      MOTHER

 1 ARNIE       ANNE
 2 BASIL       MERIDETH
 3 EDWIN       TRINITA
 4 GEORGE      LOIS
 5 HAROLD      SARAH
 6 JEROME      RUTH
 7 JIM         ANN
 8 JIM         LOUISE
 9 JOHN        ELLEN
10 JOHN        JULIE
11 ROB         DIDI
12 SHEARMAN    SARAH
13 TOM         ANNE
14 TOM         BETTY

DTR>
```

To store records in the related CHILDREN domain, you need the ID and parent data from FOLKS and the children data from FAMILY_FLAT. The record selection expression FAMILY_FLAT CROSS FOLKS OVER PARENTS gives you all the necessary information. You can use this RSE as the right-hand part of a Restructure statement for the CHILDREN domain:

```
DTR> READY FAMILY_FLAT, FOLKS
DTR> READY CHILDREN WRITE
DTR> CHILDREN = FAMILY_FLAT CROSS FOLKS OVER PARENTS
```

A PRINT statement displays the records in the new CHILDREN domain:

```
DTR> PRINT CHILDREN

      KID
ID    NAME     AGE

 1 SCOTT         2
 1 BRIAN         0
 2 BEAU         28
 2 BROOKS       26
 2 ROBIN        24
             .
             .
             .
11               0
12 DAVID         0
13 PATRICK       4
13 SUZIE         6
14 MARTHA       30
14 TOM          27

DTR>
```

lote that for ID number 11, a record was stored without a child's name. This is ıe record for ROB and DIDI, the only couple in the database without children. ecause this record is stored in CHILDREN, DATATRIEVE is able to match a OLKS record and a CHILDREN record for ROB and DIDI. As a result, ATATRIEVE includes information about ROB and DIDI when the FOLKS and HILDREN domains are joined over the ID field:

```
TR> FOR FOLKS CROSS CHILDREN OVER ID
DN>    PRINT FATHER, MOTHER, ID, KID_NAME, AGE

                                KID
 FATHER      MOTHER     ID     NAME     AGE

RNIE        ANNE         1  SCOTT        2
RNIE        ANNE         1  BRIAN        0
ASIL        MERIDETH     2  BEAU        28
                    .
                    .
                    .

DB          DIDI        11               0
                    .
                    .
                    .

TR>
```

1.1.4 Creating a Hierarchical View of Flat Records

ou can also use a view domain to display data in the FOLKS and CHILDREN omains. By using two OCCURS FOR clauses in the view domain definition, you eate a hierarchical relationship between FOLKS and CHILDREN. Printing the ecords in the view domain gives a display similar to the original FAMILIES omain.

he following example shows a view domain, FAMILY_VIEW, that simulates the ructure of the original hierarchical domain FAMILIES using the flat domains OLKS and CHILDREN:

```
TR> SHOW FAMILY_VIEW
DMAIN FAMILY_VIEW OF FOLKS, CHILDREN USING
 PARENTS OCCURS FOR FOLKS.
       03 FATHER FROM FOLKS.
       03 MOTHER FROM FOLKS.
       03 KIDS OCCURS FOR CHILDREN WITH ID = FOLK_REC.ID.
          05 KID_NAME FROM CHILDREN.
          05 AGE FROM CHILDREN.
```

(continued on next page)

```
DTR> PRINT FAMILY_VIEW

                          KID
  FATHER      MOTHER      NAME     AGE

ARNIE         ANNE        SCOTT      2
                          BRIAN      0
BASIL         MERIDETH    BEAU      28
                          BROOKS    26
                          ROBIN     24
                          JAY       22
                          WREN      17
                          JILL      20
EDWIN         TRINITA     ERIC      16
                          SCOTT     11
                    .
                    .
                    .
ROB           DIDI                   0
SHEARMAN      SARAH       DAVID      0
TOM           ANNE        PATRICK    4
                          SUZIE      6
TOM           BETTY       MARTHA    30
                          TOM       27

DTR>
```

11.2 Choose Keys for Optimization

DATATRIEVE performs best if you choose key fields for indexed records wisely. This is especially important when you use the CROSS clause. Chapter 12 in this manual has more information on key optimization.

11.3 Using Tables

Tables are useful in record definitions both for validation and for saving storage space. The record for PERSONNEL can be improved by using tables. The current definition of PERSONNEL_REC contains this field definition:

```
05 EMPLOYEE_STATUS      PIC IS X(11)
                        QUERY_NAME IS STATUS
                        QUERY_HEADER IS "STATUS"
                        VALID IF STATUS EQ "TRAINEE","EXPERIENCED".
```

EMPLOYEE_STATUS is an 11-byte field that takes only two values: TRAINEE or EXPERIENCED. Rather than storing the 11 bytes for each record, you could use a table to translate the value for a 1-byte status code. This device saves 10 bytes of storage per record and reduces time for data entry.

[ere is a definition for the dictionary table STATUS_TABLE:

```
[R> DEFINE TABLE STATUS_TABLE
?N>     E       :       EXPERIENCED
?N>     T       :       TRAINEE
?N> END_TABLE
```

ou can now edit PERSONNEL_REC, deleting the EMPLOYEE_STATUS field ıd adding two new fields that reference the table. EMP_STATUS_CODE vali- ıtes entries for the status code by checking the table. EMP_STATUS, a virtual eld, translates these code entries to either "EXPERIENCED" or "TRAINEE":

```
   05 EMP_STATUS_CODE     PIC X
                            QUERY_NAME IS S_CODE
                            VALID IF EMP_STATUS_CODE IN STATUS_TABLE.

   05 EMP_STATUS           COMPUTED BY
                                EMP_STATUS_CODE VIA STATUS_TABLE.
```

ecause the new definition defines a record with 10 fewer bytes, you need to ɛfine a new file for PERSONNEL. The following procedure illustrates how to ɛfine a new file for PERSONNEL and restructure the data to match the new ɛcord definition with the STORE USING statement:

```
:R> SHOW RESTRUCTURE_PERSONNEL
ŁOCEDURE RESTRUCTURE_PERSONNEL
:ADY PERSONNEL AS OLD
:FINE FILE FOR PERSONNEL KEY = ID
:ADY PERSONNEL AS NEW WRITE
)R O IN OLD STORE N IN NEW USING
      BEGIN
              N.ID = O.ID
              CHOICE
                      O.EMPLOYEE_STATUS = "EXPERIENCED" THEN
                      N.EMP_STATUS_CODE = "E"
                      O.STATUS = "TRAINEE" THEN
                      N.EMP_STATUS_CODE = "T"
              END_CHOICE
              N.FIRST_NAME = O.FIRST_NAME
              N.LAST_NAME  = O.LAST_NAME
              N.DEPT  = O.DEPT
              N.START_DATE = O.START_DATE
              N.SALARY = O.SALARY
              N.SUP_ID = O.SUP_ID
      END
)D_PROCEDURE

'R>
```

1.4 Using COMPUTED BY Fields

ıe revised definition PERSONNEL_REC uses a COMPUTED BY field to trans- te values by using a table. COMPUTED BY fields save storage space and can ɛlp ensure accurate data.

11.4.1 Computing Age

You can use COMPUTED BY fields to perform date calculations. Instead of entering a value for a person's age, you can enter a value for the birth date. Then you can define a COMPUTED BY field to calculate the age.

For example, FAMILIES and FAMILY_FLAT contain an AGE field. To keep the value of AGE accurate, you need to update the field continually. Instead, you could substitute the following two fields:

```
05 BIRTH_DATE  USAGE IS DATE.

05 AGE    COMPUTED BY ("TODAY" - BIRTH_DATE)/365.25
                      EDIT_STRING IS ZZ9.
```

TODAY is a date value expression that always takes the value of the current system date. Note that the factor 365.25 takes account of leap years.

11.4.2 Quarterly Summaries

You can also use COMPUTED BY fields to compute the fiscal quarter from a date value. The following examples use the domain CURRENT_SALES. The record includes a field for the salesperson's ID, the date of the sale, and the amount of the sale. Figure 11-4 illustrates the structure of the record.

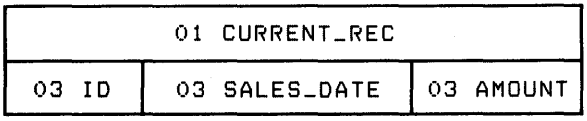


Figure 11-4: Structure of CURRENT_REC

The record definition is:

```
DTR> SHOW CURRENT_REC
RECORD CURRENT_REC USING
01 CURRENT_REC.
   03 ID    PIC IS 9(5).
   03 SALES_DATE    USAGE DATE.
   03 AMOUNT     PIC IS 9(5)V99
                EDIT_STRING IS $$$,$$$.99.
;

DTR>
```

You can add several COMPUTED BY fields to the record definition to calculate the fiscal quarter, quarterly sales, and yearly sales. The QTR field calculates the fiscal quarter from the date field SALES_DATE through a dictionary table: For example:

```
05 SALES_DATE USAGE IS DATE.
05 QTR COMPUTED BY
          (FORMAT SALES_DATE USING NN) VIA QTR_TABLE
          EDIT_STRING IS "Q"9.
```

The FORMAT value expression in QTR returns the numerical value for the month of SALES_DATE. (For more information on FORMAT value expressions see the chapter on defining and calculating values in the *VAX DATATRIEVE Handbook* and the chapter on value expressions in the *VAX DATATRIEVE Reference Manual.*) DATATRIEVE evaluates the COMPUTED BY clause, looking up this value in a table and finding the numerical value for the fiscal quarter. DATATRIEVE then displays this value, preceding the quarter number with the letter Q. The table QTR_TABLE is defined as:

```
TR> SHOW QTR_TABLE
ABLE QTR_TABLE
    QUERY_HEADER IS "QTR"
    EDIT_STRING IS 9
      1  :  3
      2  :  3
      3  :  3
      4  :  4
      5  :  4
      6  :  4
      7  :  1
      8  :  1
      9  :  1
     10  :  2
     11  :  2
     12  :  2
ND_TABLE

TR>
```

The preceding table assumes that the first quarter begins on July 1, the second on September 1, and so on. Different tables may be appropriate depending on an organization's official calendar. If quarter breaks do not occur on the first of the month, you may need a table that associates a quarter number with each of the 365 days of the year. When you look up the value in such a table, use (FORMAT TART_DATE USING JJJ) as the code field.

The CHOICE or IF-THEN-ELSE value expressions increase the flexibility of OMPUTED BY fields because you can assign values based on conditional tests. ou might want to display the sales amounts for each quarter in a separate column. You could define the following four virtual fields for the sales of different

quarters:

```
05 Q1_SALES COMPUTED BY IF QTR EQ 1 THEN AMOUNT ELSE 0.

05 Q2_SALES COMPUTED BY IF QTR EQ 2 THEN AMOUNT ELSE 0.

05 Q3_SALES COMPUTED BY IF QTR EQ 3 THEN AMOUNT ELSE 0.

05 Q4_SALES COMPUTED BY IF QTR EQ 4 THEN AMOUNT ELSE 0.
```

The values of the virtual fields for quarterly sales are either 0 or the sales amount, depending on the value for QTR.

You can also include a COMPUTED BY field in the record to calculate total sales:

```
05 TOTAL_SALES COMPUTED BY
      (Q1_SALES + Q2_SALES + Q3_SALES + Q4_SALES).
```

Now you can produce the desired output by entering a SUM statement:

```
DTR> SHOW SUMMING
PROCEDURE SUMMING
READY CURRENT_SALES
FIND CURRENT_SALES
SUM Q1_SALES ("Q1") USING $$$$,$$$.$$ ,
    Q2_SALES ("Q2") USING $$$$,$$$.$$ ,
    Q3_SALES ("Q3") USING $$$$,$$$.$$ ,
    Q4_SALES ("Q4") USING $$$$,$$$.$$ ,
    TOTAL_SALES ("TOTAL") USING $$$$,$$$.$$ BY ID
END_PROCEDURE

DTR> :SUMMING

  ID        Q1           Q2           Q3           Q4          TOTAL

11111    $2,150.91    $2,807.11    $2,748.39    $2,389.90   $10,096.31
12345    $7,805.69    $3,801.44    $9,973.94    $8,672.99   $30,254.06
22222    $5,693.29    $3,836.24    $7,274.76    $6,325.88   $23,130.17
23456   $10,311.18    $1,447.40   $13,175.40   $11,456.87   $36,390.85
33333    $7,679.00    $6,854.45    $9,812.05    $8,532.22   $32,877.72
34567    $2,338.91   $14,294.89    $2,988.61    $2,598.79   $22,221.20
44444    $8,868.17   $10,890.45   $11,331.55    $9,853.52   $40,943.69
45678    $8,999.99   $11,339.01   $11,499.99    $9,999.99   $41,838.98
55555   $23,288.42    $1,979.92   $29,757.42   $25,876.02   $80,901.78
56789   $11,111.06   $14,197.04   $14,197.46   $12,345.62   $51,851.18
66666    $9,000.01   $21,832.99   $11,500.01   $10,000.01   $52,333.02
77777    $6,593.10   $30,463.98    $8,424.52    $7,325.67   $52,807.27
88888    $4,500.00   $38,694.00    $5,750.00    $5,000.00   $53,944.00
99999    $4,499.99   $44,249.51    $5,749.99    $4,999.99   $59,499.48
       $112,839.72 $206,688.43 $144,184.09 $125,377.47 $589,089.71

DTR>
```

Note that this procedure uses the SUM statement to generate totals across each row for the different ID numbers.

Improving DATATRIEVE Performance 12

)ATATRIEVE performance depends on many factors. Among them are file rganization, selection of keys, and forming queries that take advantage of ey optimization. Here are some helpful hints and techniques that can reduce)ATATRIEVE's response time.

2.1 Choosing a File Organization

)ATATRIEVE allows you to define indexed or sequential files for your data. equential files require less storage, but DATATRIEVE must search records one y one according to their physical order in the file. This organization may be opti- ıal in certain cases. For example, a domain's records may contain a field for the ırrent date, and so records are physically arranged in the order in which they ere stored. If you access groups of records in chronological order, you may find ıis organization efficient.

ı other cases, your access needs may not be suited to this type of physical orga- zation. You may need to access a group of records with some common charac- ristic that is distributed throughout the file. If the records are stored in a quential file, DATATRIEVE may have to read all the records to find the ones ıat are requested. You also may want to erase records from the file. Although ou can modify all the fields in a record from a sequential file to contain spaces or ros, you cannot use DATATRIEVE to erase a record from a sequential file.

ı these cases, an indexed file is probably a better choice. Although indexed files quire more storage, they provide DATATRIEVE with an index based on one or ore key fields. To retrieve records, DATATRIEVE may be able to perform a fast arch through the key-based index, rather than an exhaustive search through e records.

your data is stored in an indexed file, you should structure queries to take lvantage of keyed access. This enables DATATRIEVE to use the indexes to the e. The next sections provide guidelines to help you do this.

12.1.1 Choosing the Primary and Alternate Keys

When defining data, try to decide which field of the record is likely to be named most often in queries. Make that field the primary key of an indexed file for the domain. For example, if you are setting up a PERSONNEL domain, you might predict that most users will seek information on an employee based on his or her ID. In that case, make the ID field the primary key.

It is also desirable that the primary key be unique. That is, the primary key should be sufficient to identify the record. This is the default for primary keys in DATATRIEVE. Though it is legal in DATATRIEVE to have duplicate values for the primary key, too many duplicates slow performance so that a search of such a file may even take longer than a search of a sequential file. Allowing duplicate values can also create very large files.

DATATRIEVE does not allow you to change the value of a primary key field. Therefore, do not select as a primary key any field you expect to update. (If you must change the value in a primary key, you have to erase the record and store it again.)

If the best field for the primary key does not uniquely identify the record, find another field such that the two fields jointly can determine the record. You can then designate a group field, encompassing the two fields, as the primary key. For example, in the YACHTS domain, the group field TYPE (consisting of BUILDER and MODEL) is the primary key, uniquely determining each record in YACHTS.

If you use a group field as a primary key, keep in mind that DATATRIEVE can perform keyed access only on the first elementary field within the group field. For example, DATATRIEVE optimizes on the BUILDER field of YACHTS, the first elementary field of TYPE. But DATATRIEVE cannot do keyed access on the second elementary field, MODEL. Any queries based on MODEL require DATATRIEVE to do an exhaustive search through the records in YACHTS.

In addition, group field keys should not contain numeric items. DATATRIEVE cannot do keyed access on a numeric elementary field when it is part of a multiple field key.

If there are other fields that will be used often in queries, you can define them as alternate keys. Alternate keys can have duplicate values. However, if you expect to have many duplicate values for a field, do not define it as a key; access can be faster and the file is smaller when the field is not a key.

Try to keep your index structure as simple as you can. Defining many alternate key fields that you expect to update frequently can cause performance problems later on. RMS must allocate disk space for your file's index just as it allocates storage for your file's data. Frequent updates to key fields mean that your index becomes fragmented, stored in many small pieces on different parts of the disk. When this happens, DATATRIEVE takes longer to read the index than it did when you first created the file. You can correct this problem using RMS utilities.

2.2 Designing Files

: a file is large and randomly accesssed, the DEFINE FILE command in)ATATRIEVE may not result in optimized file design.

)ATATRIEVE, for example, uses a default for an RMS file with the following arameters:

A bucket size of 2 (512-byte) disk blocks

A contiguous allocation of 3 blocks

Global buffer count of 0

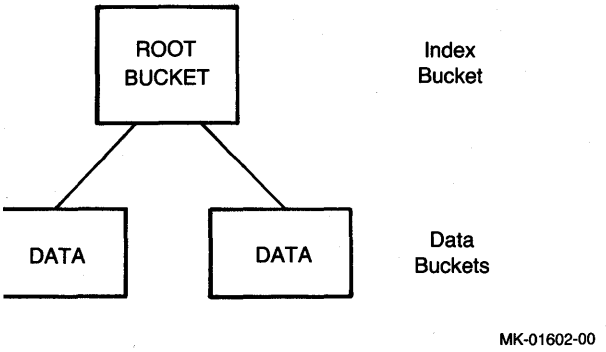
File extent of 0 blocks

he RMS defaults for the DEFINE FILE syntax can cause data in a large ıdexed file to become scattered over your disk, requiring time-consuming I/O ıput/output) operations.

wo important considerations are bucket size and index structure. These two file :tributes are related; that is, the smaller the bucket size, typically the deeper the ıdex structure. Frequently, a major problem is that bucket size is too large, and ıe resulting index structure is too flat.

bucket is the unit of transfer between storage devices and I/O buffers in mem-'y. A bucket size can be from 1 to 32 blocks (each block containing 512 bytes). s a general rule, a small bucket size is optimal for randomly accessed records. Iowever, buckets must contain enough room for record insertion or bucket split-ng occurs. Bucket splits fragment your data and increase I/O overhead and me.)

flat file has an index structure with only one level. If your file contains more ıan a few hundred records, a single level index bucket (also called a root bucket) ın be very large. A large root bucket results in slow access time for a particular ıta record. Figure 12-1 shows the structure of a flat file.



gure 12-1: Flat File Structure

Flat file structure and nonoptimal bucket size may be the most significant reasons for slow access time. A file with two or three levels of index structure and smaller bucket size allows you to access data much more quickly. Figure 12-2 shows a file with two index levels.

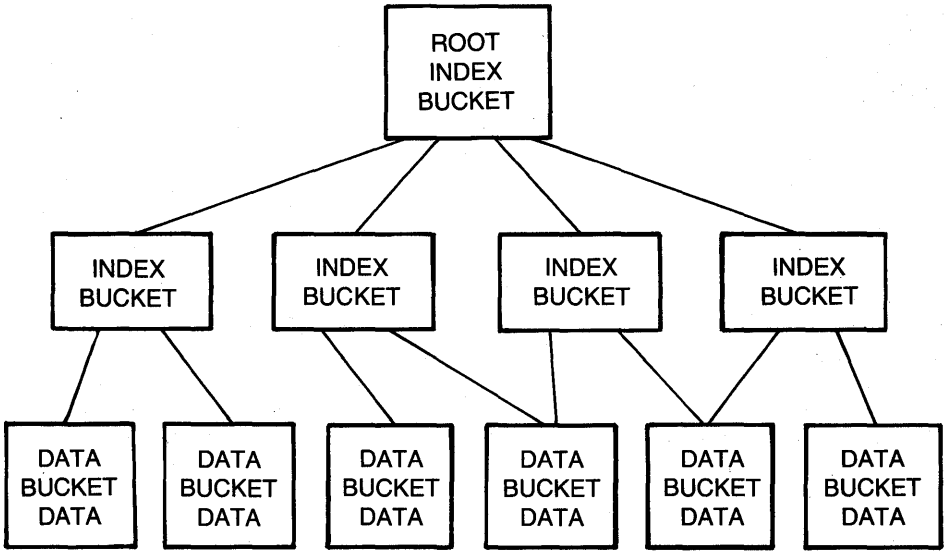


Figure 12-2: A File with Two Levels of Index

The following sections describe RMS utilities that help you design a file with more optimal bucket size and index depth. They also explain how to optmize other file attributes such as global buffers.

12.2.1 Using EDIT/FDL to Design Your File

By using the RMS utility File Definition Language (EDIT/FDL) to create a large indexed file, you have more flexibility in creating and managing files than you have with DATATRIEVE. EDIT/FDL can calculate the file allocation, file extent, and bucket size, thus optimizing I/O operations and minimizing file fragmentation.

Before invoking FDL, you need to determine the following about your record and data file:

- The total number of records your file will contain
- The number, size, and data type of your index keys
- The size of the record in bytes

ı the next four sections you learn how to invoke EDIT/FDL and use the Design hase to describe the attributes of your DATATRIEVE file.

2.2.1.1 Questions EDIT/FDL Asks -- When you have determined the basic ıcts about your file, you can invoke the EDIT/FDL utility using the following ɔmmand line:

```
DIT/FDL/SCRIPT = DESIGN filespec.fdl
```

his command invokes the prompting form of the FDL utility. EDIT/FDL asks ɔr the following information about your file:

1. Whether the file is to be indexed
2. The number of indexed keys
3. How you want EDIT/FDL to display your file design (on a line plot or on a surface plot graph)
4. What kind of emphasis you want EDIT/FDL to use when selecting bucket size (smaller buffers [buckets] or flatter files)
5. The number of records that will be initially loaded in the file
6. How you will load records into the file
7. If the records will be loaded in order of ascending primary key
8. The number of additional records you will add after the initial file load
9. If additional records will be added in order of ascending primary key
10. The fill factor of the buckets for each index key
11. Whether the record format is fixed or variable
12. The mean and maximum record size in bytes
13. The data type and length of each field
14. If you will allow duplicates in the key field
15. If you want global buffers
16. The bucket size you want to select from the three choices EDIT/FDL recommends

se the help facility within EDIT/FDL to learn more about the utility and the ıide on tuning VAX RMS for information about file parameters.

12.2.1.2 Answers to the EDIT/FDL Prompts -- Your answers to most of the questions asked by EDIT/FDL depend on obvious information you have about your file, such as record size, number of records, and number of index keys. For other questions, the following list suggests answers you can give to the prompts:

- When prompted for the type of display you would like to see, enter LINE_PLOT. This tells EDIT/FDL you want the bucket size and index depth it selects displayed on a line plot graph.

```
(Line_plot Surface_plot)
Key 0   Design Mode                (keyword)[-]: Line_plot
```

- When prompted for the emphasis you want EDIT/FDL to use in selecting bucket size, enter small. This tells EDIT/FDL to use smaller buffers (or buckets) rather than flatter files.

```
(Smaller_buffers Flatter_files)
Emphasis for Default Bucket_size (keyword) [SMALL]: Small
```

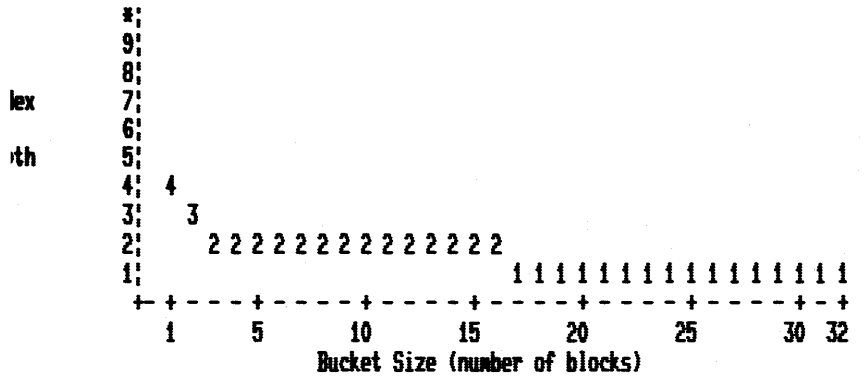
- When prompted for how you will initially load records into the file, answer RMS_Puts. This tells EDIT/FDL that DATATRIEVE will write records to these files.
- When prompted for the number of additional records you will add, enter 0. This question is useful only if you are converting an old data file.
- When prompted for fill factor, select 100 (for 100%). DATATRIEVE will fill the buckets to maximum capacity when it places data in a file, regardless of the figure you enter here.
- In the Final Design phase of EDIT/FDL, when prompted to answer whether Global Buffers are desired, answer no. After you exit EDIT/FDL you will cal culate a global buffer count using the procedure in the section on Optimizing Global Buffers.

After you answer the initial questions, EDIT/FDL displays a resulting graph that shows the relation of bucket sizes to index depths for the file you have described. The next section describes this graph, shown in Figure 12-3, and explains how to select optimum bucket size.

12.2.1.3 Selecting Optimum Bucket Size -- In EDIT/FDL you select a bucket size, first for your file's primary key structure and then for alternate key structures. The bucket size you select for your primary key is also the size of your dat buckets. This bucket size then determines the resulting index depth of your file.

ter you have selected an emphasis for bucket size (smaller buffers [or buckets] ther than flatter files) EDIT/FDL calculates a ratio of bucket size to index pth.

displays this ratio on the LINE_PLOT graph in EDIT/FDL (Figure 12-3). The tio is based on all the questions you answer, including the size of your records, e number of records you will initially load in your file, the number of records you pect to add later, and the emphasis you select.




```
-Prologue Version     3 KT-Key 0   Type     String FD-Final Design Phase
-Dup Key 0   Values Yes KL-Key 0   Length       21 KP-Key 0   Position     0
-Data Record Comp    0% KC-Data Key Comp        0% IC-Index Record Comp   0%
-Bucket Fill       100% RF-Record Format     Fixed RS-Record Size        212
-Load Method  Fast_Conv IL-Initial Load       1000 AR-Added Records     10000
    Which File Parameter    (Mnemonic)[refresh]     : fd
```

MK-01597-00

ure 12-3: EDIT/FDL Display of Index Depth versus Bucket Size

The line graph EDIT/FDL displays shows the ratio of bucket size to index depth that results from the file parameters you have selected. For example, the graph i Figure 12-3 shows that if you select:

- A bucket size of 1 block, you get an index depth of 4
- A bucket size of 2 blocks, you get an index depth of 3
- A bucket size of 3 through 18 blocks, you get an index depth of 2
- A bucket size of 19 through 32 blocks, you get in an index depth of 1

The points at which index size changes are called breakpoints. For example, the index depth changes:

- From 4 to 3 at a bucket size of 2 blocks (the breakpoint is 2)
- From 3 to 2 at a bucket size of 3 blocks (the breakpoint is 3)

After viewing the graph, to specify bucket size you must select the Final Design (FD) Phase of EDIT/FDL. EDIT/FDL then displays the following information:

```
Bucket Size Emphasis: ( Smaller_buffers )
Bucket size Breakpoints: (2, 3, 13)

Key 0  Bucket size          (1-32) [3]:
```

It shows the breakpoints (2, 3, and 13) and a default bucket size (3). (The third breakpoint, 13, represents a middle point in the range of all the remaining bucke sizes and is unimportant for your selection.)

EDIT/FDL has suggested a bucket size of 3. Note, however, that when you selec a bucket size, EDIT/FDL lets you specify any bucket size from 0 to 32. Basically though, you want to choose the smallest bucket size (the breakpoint) that corresponds to 2 or 3 levels of index. In this example, a bucket size of 3 corresponds t 2 levels of index. Always select the smallest bucket size from the range, in this case, 3 buckets rather than 4 to 18.

Note then, in this example the default bucket size of 3 is acceptable. It is the smallest bucket size that corresponds to an index depth of 2. You should always check the default bucket size as it may not be the most optimal.

While many file parameters have a significant effect on file performance, bucket size and index depth are the most important.

After you select a bucket size for your primary key structure (which is also the bucket size for your data), you answer the same series of questions about your alternate key structure and then select a bucket size for that key.

w you can exit EDIT/FDL and create your data file. You should remember the :ket sizes you selected for primary and alternate keys, as you will use them to ign global buffers to your file.

2.2 Creating the Data File

:er you exit EDIT/FDL, it creates a file definition file containing the attributes ı described. The file has an extension of .FDL. You use this file to create an pty data file with the attributes you described in EDIT/FDL.

ter the command:

```
EATE/FDL = filespec.fdl filespec.dat
```

:spec.dat is the empty data file you are creating using the RMS CREATE lity. The utility uses the file and record attributes you defined in the FDL spec to create this data file.

2.3 Optimizing Global Buffers

er you have created the data file, you will want to assign global buffers to your . Allocating buffers for indexed files gives RMS more space to store the index ucture in memory. Locating a record takes less time if the record's index is red in one of the buffer areas.

a general rule, allow enough global buffers to equal the number of index buck- in all the key structures (primary plus alternates) plus 5.

find out how many global buffers to assign, you can run the DATATRIEVE cedure provided in this section. It first calculates the number of index buckets 'our file and then the number of global buffers you should assign.

s procedure prompts you for:

The bucket size of a primary key structure (calculated in EDIT/FDL)

The size of the record in bytes

The number of records in the file

The fill factor of the primary key (you specified 100 in EDIT/FDL)

: procedure gives you the number of index buckets RMS provided for the nary key structure. It then calculates the number of index buckets in the

alternate key structure. It prompts you for:

1. The bucket size for the alternate key (calculated in EDIT/FDL)
2. The size of the alternate key
3. The fill factor of the alternate key
4. The number of duplicates you will allow in the alternate key

It then calculates the number of index buckets in the alternate key structure, th total number of index buckets, and the global buffer count.

```
DEFINE PROCEDURE BUCKETS
!
!=============================================================+
!                                                             |
!     This procedure calculates the number of index buckets   |
!     in an RMS indexed file containing FIXED length records  |
!     and then specifies a GLOBAL BUFFER count.               |
!=============================================================+
!
DECLARE MORE            PIC X
                        VALID IF MORE = "Y","N","y","n".
DECLARE PRIMARY_DONE    PIC X
                        VALID IF DONE = "Y","N", y","n".
DECLARE KEY_PROMPT      PIC X
                        VALID IF KEY_PROMPT = "P","A","p","a".
DECLARE BUCKET_SIZE     PIC 9(2)
                        VALID IF BUCKET_SIZE BETWEEN 1 AND 32.
DECLARE KEY_SIZE        PIC 9(3)
                        VALID IF KEY_SIZE BETWEEN 1 AND 125.
DECLARE REC_SIZE        PIC 9(4).
DECLARE NUM_RECS        PIC 9(6).
DECLARE REC_OVHD        PIC 9(2).
DECLARE KEY_OVHD        PIC 9(1).
DECLARE NUM_DUPS        PIC 9(3).
DECLARE FILL_FACTOR     PIC 9(3)V99.
DECLARE TMP_DRPB        PIC 9(5)V99.
DECLARE DRPB            PIC 9(5).
DECLARE NUM_DB          PIC 9(5).
DECLARE NUM_IB          PIC 9(5).
DECLARE TMP_NUM_DB      PIC 9(5)V99.
DECLARE TMP_NUM_IB      PIC 9(5)V99.
DECLARE PREV_NUM_BUCK PIC 9(5).
DECLARE IRPB            PIC 9(5).
DECLARE TOT_IDB         PIC 9(5)
                        EDIT_STRING IS ZZZZ9.
DECLARE GRAND_NUM_IDB PIC 9(5)
                        EDIT_STRING IS ZZZZ9.
DECLARE ICOUNT          PIC 9.
!=============================================================
```

```
KEY_OVHD = 3
MORE = "Y"
GRAND_NUM_IDB = 0
NUM_RECS = 0
PRIMARY_DONE = "N"
PRINT " "
PRINT "*************************************************************"
PRINT "*                                                           *"
PRINT "*  This procedure calculates the number of index buckets    *"
PRINT "*  in an RMS indexed file containing FIXED length records   *"
PRINT "*  and then specifies a GLOBAL BUFFER count.                *"
PRINT "*                                                           *"
PRINT "*************************************************************"
WHILE MORE EQ "Y"
BEGIN
 TOT_IDB = 0
 PRINT " "
  CHOICE
   (PRIMARY_DONE EQ "N") THEN BEGIN
    KEY_PROMPT = *."Primary or Alternate key structure (P or A) "
    KEY_PROMPT = FN$UPCASE(KEY_PROMPT)
  END
 ELSE KEY_PROMPT = "A"
 END_CHOICE
 CHOICE
  (KEY_PROMPT EQ "P") THEN BEGIN
    PRIMARY_DONE = "Y"
    BUCKET_SIZE = *." bucket size of the primary key structure (0-32) "
    REC_SIZE = *."record size in bytes (1-9999) "
    KEY_SIZE = *."size of the primary key in bytes (1-125) "
    NUM_RECS = *."number of records in the file (0 - 999,999) "
    FILL_FACTOR = *."fill factor of the primary key structure (0 - 100) "
    FILL_FACTOR = FILL_FACTOR / 100.0
    REC_OVHD = 7
    END
    ELSE BEGIN
     BUCKET_SIZE = *."bucket size of the alternate key struc.(0-32) "
     KEY_SIZE = *."size of the alternate key in bytes (1-125) "
     REC_SIZE = KEY_SIZE
     CHOICE
      (NUM_RECS EQ 0) THEN
       NUM_RECS = *."number of records in the file (0 - 999,999) "
      END_CHOICE
     FILL_FACTOR = *."the fill factor of the alternate key  (0 - 100) "
     FILL_FACTOR = FILL_FACTOR / 100.0
     NUM_DUPS = *."the number of dup keys in the alternate key (0 - 100) "
     CHOICE
      (NUM_DUPS EQ 0) THEN  REC_OVHD = 9
      ELSE REC_OVHD = (8 + (5 * NUM_DUPS))
     END_CHOICE
     END
  END_CHOICE

 Find the floor number of data records per bucket.

'MP_DRPB = (((BUCKET_SIZE * 512) * FILL_FACTOR) - 15) / -
          (REC_SIZE + REC_OVHD)
)RPB = FN$FLOOR(TMP_DRPB )
```

(continued on next page)

```
!
!      Find ceiling number of data buckets.
!
TMP_NUM_DB = (NUM_RECS / DRPB) + 0.49
NUM_DB = (TMP_NUM_DB )
!
IRPB = (((BUCKET_SIZE * 512) * FILL_FACTOR) - 15) / -
         (KEY_SIZE + KEY_OVHD)
PREV_NUM_BUCK = NUM_DB
NUM_IB = 0
ICOUNT = 0
WHILE (NUM_IB NE 1)
BEGIN
ICOUNT = ICOUNT + 1
!
!           Find ceiling number of index buckets.
!
TMP_NUM_IB = (PREV_NUM_BUCK / IRPB) + 0.49
NUM_IB = (TMP_NUM_IB )
CHOICE
  (NUM_IB GT 1)THEN BEGIN
    TOT_IDB = TOT_IDB + NUM_IB
    PREV_NUM_BUCK = NUM_IB
    END
END_CHOICE
END
TOT_IDB = TOT_IDB + 1
PRINT " "
PRINT "============================================================"
PRINT " "
CHOICE
  (KEY_PROMPT EQ "P") THEN
   PRINT "NUMBER OF TOTAL INDEX BUCKETS FOR -
   PRIMARY KEY STRUCTURE ==> ", TOT_IDB(-)  ELSE
   PRINT "NUMBER OF TOTAL INDEX BUCKETS FOR ALTERNATE KEY -
   STRUCTURE ==> ", TOT_IDB(-)
END_CHOICE
PRINT " "
PRINT "============================================================"
PRINT " "
!
MORE = *."Y to calculate more index structures, N to exit "
MORE = FN$UPCASE(MORE)
GRAND_NUM_IDB = GRAND_NUM_IDB + TOT_IDB
END;
PRINT " "
PRINT "************************************************************"
PRINT " "
PRINT "NUMBER OF TOTAL INDEX BUCKETS FOR ALL KEY -
       STRUCTURES ==> ", GRAND_NUM_IDB(-)
PRINT " "
!
!Add 5 to the number of index buckets
!
GRAND_NUM_IDB = GRAND_NUM_IDB + 5
PRINT "Set the GLOBAL BUFFER attribute on -
        the file to ", GRAND_NUM_IDB(-)
PRINT " "
PRINT "************************************************************"
PRINT " "
END_PROCEDURE;
```

e following example illustrates how the procedure works to calculate the num-
· of global buffers you should assign to your file.

```
\> :buckets

:*************************************************************
                                                             *
This procedure calculates the number of index buckets        *
in an RMS indexed file containing FIXED length records       *
and specifies the number of GLOBAL BUFFERS.                  *
:*************************************************************

;er Primary or Alternate key structure (P or A) : P
;er bucket size of the primary key structure (0-32) : 3
;er record size in bytes (1-9999) : 1000
;er size of the primary key in bytes (1-125) : 21
;er number of records in the file (0 - 999,999) : 10000
;er fill factor of primary key structure (0 - 100) : 100

:===========================================================

IBER OF TOTAL INDEX BUCKETS FOR PRIMARY KEY STRUCTURE ==> 83

:===========================================================

;er Y to calculate more index structures, N to exit : y

;er Primary or Alternate key structure (P or A) : a
;er bucket size of the alternate key structure (0-32) : 2
;er size of the alternate key in bytes (1-125) : 12
;er number of records in the file (0 - 999,999) : 10000
;er fill factor of the alternate key structure (0 - 100) : 100
;er number of dup keys in alternate key structure (0 - 100) : 2

:===========================================================

IBER OF TOTAL INDEX BUCKETS FOR ALTERNATE KEY STRUCTURE ==>   6

:===========================================================

er Y to calculate more index structures, N to exit : n

*************************************************************

BER OF TOTAL INDEX BUCKETS FOR ALL KEY STRUCTURES ==>     89

 the GLOBAL BUFFER attribute on the file to     94

*************************************************************

>
```

ɜ procedure calculated that there are 83 index buckets for the primary index
· in the sample file you described and six index buckets for the alternate key. It
led five to the total index buckets and arrived at a global buffer count of 94.

After you have the correct global buffer count, you can adjust the count using the SET FILE/GLOBAL_BUFFERS command:

```
SET FILE/GLOBAL_BUFFERS = n filespec.dat
```

12.2.4 Redesign and Maintenance

It is important to maintain files you use in DATATRIEVE applications, particularly if they are large indexed files. If you have added or deleted many records, changed the number of indexed keys, or adjusted the size of your records, you may have a badly fragmented file or a file bucket size or global buffer count that may be causing poor I/O performance.

It is easiest to go through the same set of procedures you used to create your initial file. Invoke EDIT/FDL and create a new .FDL file description. Before invoking EDIT/FDL, you need to know:

- The same information you needed when you first created your file such as record size and key size
- A fill factor for the file you are redesigning

You may want to use the RMS Analyze Utility to remember such information as record size, number of keys, and key sizes. You can enter the ANALYZE command as in the following example:

```
ANALYZE/RMS_FILE Filename.dat
```

12.2.4.1 Calculating a Fill Factor -- You should calculate a new fill factor before invoking EDIT/FDL. The fill factor is important when redesigning your file. The fill factor ensures that when you move the data from your old fragmented file to your new redesigned file, RMS leaves room in the buckets for additional records to be added after you load the file. This room reduces the amount bucket splitting that occurs when you add records to the file at a later time. Use the following formula to calculate the fill factor:

```
Fill Factor = 100 - ((the number of records to be added/
                     (the number of initial records in the file) * 100))
```

For example, if your file contains 10,000 records now and you intend to add approximately 1000 more before you redesign again, you would specify a fill fact of 90:

```
Fill Factor = 100 - ((1000/(10,000) *100))
```

The easiest way to determine the number of records in your file is by invoking DATATRIEVE and entering the statements:

```
)TR> READY domain-name
)TR> PRINT COUNT OF domain-name
```

Now that you know the fill factor and the other information for which EDIT/FDL prompts you, you can invoke the utility and design a new .FDL file:

```
EDIT/FDL/SCRIPT = DESIGN filespec.fdl
```

12.2.4.2 Adding Data to the File -- After you redesign your file, you will want to move data from the existing file into a new data file. It is best to use the RMS CONVERT utility to load large files. It is much faster than DATATRIEVE and loads data more optimally.

Use the following command line to create a data file using your new .FDL to describe the file and to load that new file with data from the old file:

```
CONVERT/FDL = filespec.fdl oldfile.dat newfile.dat
```

For more information on file tuning and RMS utilities refer to the guide on tuning VAX RMS.

12.3 Choosing Optimal Queries

Once you establish the file organization, you should try to choose queries that are most efficient. The following sections indicate guidelines for optimal queries.

12.3.1 Using EQUAL Rather Than CONTAINING

A query is a request for DATATRIEVE to identify all the records that satisfy a specified condition. A Boolean expression that tests records with the EQUAL (=) relational operator is more efficient than a Boolean with CONTAINING (CONT). This rule is most significant if the Boolean expression references a key field:

```
TR> PRINT YACHTS WITH BUILDER = "PEARSON"

TR> PRINT YACHTS WITH BUILDER CONT "PEARSON"
```

Although both queries yield the same results, the first query is about twice as fast as the second one.

DATATRIEVE gives optimal performance in the first case because the query specifies an exact match for the MANUFACTURER (BUILDER) field, the first elementary field of the key field TYPE. DATATRIEVE conducts a fast search through the index to retrieve the desired records.

In the second case, DATATRIEVE must search through the values of BUILDER looking for matches with the string following CONT. DATATRIEVE must check all substrings of each BUILDER value that are equal in length to the string specified in the Boolean.

To take advantage of the increased efficiency of EQUAL (=), you must specify a value that matches the field value exactly. EQUAL (=) is case-sensitive, but CONT is not case-sensitive. In the last example, if a record had the value "Pearson" for BUILDER, only the second query would find the record.

To get around the case-sensitivity problem, you can use the DATATRIEVE function FN$UPCASE in procedures that store data to ensure that all text fields are entered as uppercase. Then you can be sure a search using the EQUAL operator will find all the records you want to locate. Otherwise, to use the EQUAL operator you must remember the case of each character of a field value.

12.3.2 Using STARTING WITH Rather Than CONTAINING

To improve performance, you can sometimes substitute the STARTING WITH relational operator for CONTAINING. This operator allows you to find records in which the beginning substring of the field value exactly matches the specified value expression. If you name a key field in the query, DATATRIEVE is able to use a key-based index. Remember that this operator is case-sensitive.

Of the following two queries, the first query is more efficient because of the keyed access. DATATRIEVE does not have to check all possible substrings of each BUILDER value:

```
DTR> PRINT YACHTS WITH BUILDER STARTING WITH "ALB"

                                LENGTH
                                 OVER
MANUFACTURER    MODEL      RIG   ALL   WEIGHT BEAM  PRICE

 ALBERG       37 MK II   KETCH   37    20,000  12  $36,951
 ALBIN        79         SLOOP   26     4,200  10  $17,900
 ALBIN        BALLAD     SLOOP   30     7,276  10  $27,500
 ALBIN        VEGA       SLOOP   27     5,070  08  $18,600

DTR> PRINT YACHTS WITH BUILDER CONTAINING "ALB"

                                LENGTH
                                 OVER
MANUFACTURER    MODEL      RIG   ALL   WEIGHT BEAM  PRICE

 ALBERG       37 MK II   KETCH   37    20,000  12  $36,951
 ALBIN        79         SLOOP   26     4,200  10  $17,900
 ALBIN        BALLAD     SLOOP   30     7,276  10  $27,500
 ALBIN        VEGA       SLOOP   27     5,070  08  $18,600

DTR>
```

12.3.3 Using Domains Rather Than Collections in an RSE

DATATRIEVE cannot use indexes to retrieve records from a collection. In general, then, to get the best performance on key-based queries, use a domain rather than a collection as the source for the RSE.

The previous section noted that DATATRIEVE can do a keyed retrieval if you use the STARTING WITH relational operator. The potential gain in performance is lost if you form a collection. For example, the following queries use STARTING WITH, but DATATRIEVE uses the key-based index only in the first case:

```
DTR> PRINT YACHTS WITH BUILDER STARTING WITH "AL"

DTR> FIND YACHTS; PRINT CURRENT WITH BUILDER STARTING WITH "AL"
```

The first query is substantially faster because DATATRIEVE can do a search through the index of YACHTS. DATATRIEVE must do an exhaustive search in the second case.

2.3.4 Using the CROSS Clause and Nested FOR Loops

If you have two domains that share a common field, you can relate their records either with the CROSS clause or with nested FOR loops. For example, the YACHTS and PAYABLES domains share the field TYPE. The following queries search for records from these two sources:

```
DTR> PRINT PAYABLES CROSS YACHTS OVER TYPE
```

The query has the form "PRINT rse". The same results can be achieved with nested FOR loops. For example:

```
DTR> FOR A IN PAYABLES
CON> FOR YACHTS WITH TYPE = A.TYPE
CON>     PRINT A.PAYABLE, BOAT
```

This query is processed about as fast as the previous example with CROSS. DATATRIEVE is able to use the key-based index to YACHTS. Note these features of the two queries:

- Domains are used rather than collections as record sources, so that DATATRIEVE can use its key-based index to the records of YACHTS.
- The OVER clause uses TYPE, a key field only for YACHTS. Because TYPE is not a key field in PAYABLES, the queries specify PAYABLES before YACHTS.

- The YACHTS record stream contains many more records than PAYABLES, and is best placed in the second position in each query.

The next sections explain why these principles affect DATATRIEVE's performance.

12.3.5 Choosing Domains or Collections as Record Sources

To form a query that relates two record sources, you can use either collections or domains. Keep in mind that DATATRIEVE can do keyed access only for domains and only if the domain is other than the first record source specified. In other words, when you use CROSS or nested FOR loops to access two domains and you relate those domains through a common key field, DATATRIEVE can use keyed access for searching the second domain in the CROSS clause or the domain in the second FOR loop.

If all other conditions are equal, it is better to use a domain name rather than a collection name in the second position of a key-based relational query. There is one more factor to consider, however: collections are efficient to use if you need to refer back to the same group of records in the same DATATRIEVE session. In such a case, you may get better performance by forming and naming a collection, so that DATATRIEVE does not have to retrieve the same group of records over and over again.

Be aware of this tradeoff when choosing a record source. You gain efficiency with a domain when you can use keyed access. On the other hand, you gain efficiency with a collection if you reduce the number of times DATATRIEVE must isolate the same small group from a large body of records. A collection can also reduce the number of records in the record stream and help improve the performance of CROSS.

12.3.6 Choosing the Order of Domain Names in the CROSS Clause

DATATRIEVE can use a key-based index only for the domain that is specified second in the CROSS clause.

When using the CROSS clause, you can relate two domains that have a common field. This field can be specified in the OVER clause or in a Boolean expression that is part of the WITH clause. (In the following example, the clause OVER TYPE is equivalent to WITH TYPE = TYPE.) If this field is a key for only one of the domains, you get a faster response if you specify that domain second in the CROSS clause:

```
DTR> PRINT PAYABLES CROSS YACHTS OVER TYPE
```

ACHTS is listed second because TYPE is a key field only for YACHTS, not for AYABLES. If PAYABLES had been listed second, DATATRIEVE's response ould have been substantially slower. The query as shown is more than ten times ster than if you list PAYABLES after YACHTS.

second guideline is to specify the smaller *record stream* first in the CROSS ause:

```
R> PRINT BOAT, NAME, BOAT_NAME OF OWNERS CROSS YACHTS OVER TYPE
```

iis query is more than twice as fast as the same query with the order of the mains reversed. Since the YACHTS record stream is much larger than the WNERS record stream, you can save time by allowing DATATRIEVE to use e key-based index for YACHTS. DATATRIEVE gets each record in OWNERS, elatively small number, and then evaluates the OVER clause by means of the dex to YACHTS.

the reverse order is used, DATATRIEVE gets each record in YACHTS (113 in) and then evaluates the OVER clause by means of the index to OWNERS. nce there are only 10 OWNERS records, a search through the key-based index es not save much time. In the other case, a search through the YACHTS index ves a search through all 113 YACHTS records.

hat is crucial is the number of records in each record stream, not the records in e record source. If you are only interested in Alberg's yachts, it is more efficient place YACHTS WITH BUILDER = "ALBERG" in the first position. ATATRIEVE evaluates the Boolean using the index to BUILDER and finds one ord. Then DATATRIEVE loops through the OWNERS records only once to n the two record streams. Though the record source (the YACHTS domain) has ny records, the record stream based on the source is very small.

ese principles are important when you use CROSS with more than two mains. Assume that you have domains A, B, and C and you relate them in the lowing expression:

```
INT A CROSS B OVER X CROSS C OVER Y
```

X is a key for B, and Y is a key for C, DATATRIEVE uses both keys in evaluating the entire expression. DATATRIEVE does not use the keys if X is a key ly for A, and Y is a key only for B.

r example, you could relate the three domains PAYABLES, OWNERS, and ACHTS. OWNERS and YACHTS both have TYPE as a key field, so

DATATRIEVE is able to use both the index to OWNERS and the index to YACHTS in evaluating the following expression:

```
DTR> FOR PAYABLES CROSS OWNERS OVER TYPE CROSS YACHTS OVER TYPE
DTR>     PRINT TYPE, RIG, NAME, BOAT_NAME, PRICE, WHSLE_PRICE

                                                    BOAT                  WHSLE
MANUFACTURER    MODEL      RIG       NAME           NAME        PRICE     PRICE

 ALBIN          VEGA       SLOOP     STEVE          DELIVERANCE $18,600   $14,25(
 ALBIN          VEGA       SLOOP     HUGH           IMPULSE     $18,600   $14,25(
 ISLANDER       BAHAMA     SLOOP     JIM            POTEMKIN     $6,500    $4,95(
 ISLANDER       BAHAMA     SLOOP     ANN            POTEMKIN     $6,500    $4,95(
 ISLANDER       BAHAMA     SLOOP     STEVE          POTEMKIN     $6,500    $4,95(
 ISLANDER       BAHAMA     SLOOP     HARVEY         MANANA       $6,500    $4,95(

DTR>
```

12.3.7 Order of Domains in Nested FOR Loops

Nested FOR loops can produce the same results as CROSS, and similar rules apply. Include the domain that has the key field in the second or inner FOR loop For example:

```
DTR> FOR A IN PAYABLES
CON> FOR YACHTS WITH TYPE = A.TYPE
CON>     PRINT A.ORDR_NUM, BOAT, A.INVOICE_DUE, A.BILL_PAID
```

This query is about ten times faster than:

```
DTR> FOR A IN YACHTS
CON> FOR PAYABLES WITH TYPE = A.TYPE
CON>     PRINT ORDR_NUM, A.BOAT, INVOICE_DUE, BILL_PAID
```

In the first case, DATATRIEVE knows that the YACHTS records are ordered according to TYPE. DATATRIEVE can do a fast search through the index to YACHTS for matches on TYPE, before executing the PRINT statement. This process is substantially faster.

In the second case, however, DATATRIEVE must evaluate the Boolean "WITH TYPE = A.TYPE" without the benefit of a key-based index, because TYPE is not a key field for PAYABLES. For each record in YACHTS, DATATRIEVE must do a search through all of the PAYABLES records to find matches on TYPE.

The same rule holds concerning the relative size of the two record streams. If or record stream has many more records than the other and both have the same k field, the larger record stream should be included in the second (inner) FOR loop

12.3.8 Nested FOR Loops Followed by a Conditional Statement

Try to avoid using nested FOR loops to control the execution of a conditional statement. The following example removes the Boolean expression from the RSE and places it within an IF-THEN statement. It is extremely inefficient:

```
)TR> FOR A IN PAYABLES
:ON> FOR YACHTS
:ON>  BEGIN
:ON>    IF TYPE = A.TYPE AND LOA > 40 THEN
:ON>    PRINT A.PAYABLE, BOAT
'ON>  END
```

)ATATRIEVE gets one record from YACHTS and one from PAYABLES. It ests for the truth of the condition "TYPE = A.TYPE" AND LOA > 40. Because here are 30 records in PAYABLES and 113 records in YACHTS, DATATRIEVE nust go through this procedure 30 X 113 (3390) times. Because DATATRIEVE s evaluating the conditions for every record of YACHTS individually, the index o YACHTS based on TYPE is not used.

'he query is improved when the test is part of the WITH clause of the RSE (or in he OVER clause of CROSS). DATATRIEVE does not have to get every record of 'ACHTS 30 times. For each of the 30 PAYABLES records, DATATRIEVE can o a fast search through the index to YACHTS.

Vherever possible, you should include conditional tests as Boolean expressions /ithin the RSE. This effectively limits the number of records that DATATRIEVE as to process. For example:

```
TR> FOR A IN PAYABLES CROSS YACHTS OVER
ON>  TYPE WITH LOA > 40
ON>  PRINT A.PAYABLE, BOAT
```

12.4 Timing Procedures to Improve Efficiency

'he recommendations in the previous sections were verified by timing alternative rocedures with DATATRIEVE's timing functions, FN$INIT_TIMER and 'N$SHOW_TIMER. The first of these functions initializes a timer, and the sec- nd calculates the elapsed time. A good comparative measure is the CPU time xpended by several alternative procedures that produce the same output. You ıay find that the extra effort needed to time procedures may be repaid by)ATATRIEVE's improved performance.

: you will be invoking a procedure frequently and have a choice between two que- es, you can time each query to see which one is most efficient. To save CPU me, you might include only a subset of the records in your tests.

For example, suppose you want to display information on manufacturers who make boats with more than one type of rig. This kind of query requires that you compare records within the same domain, YACHTS. The first solution, using nested FOR loops followed by a conditional, requires DATATRIEVE to search and compare the 113 records in YACHTS 113 times. When this inefficient query was invoked, the timing functions indicated that it required 50.04 seconds of CPU time:

```
DTR> SHOW TIME1E
PROCEDURE TIME1E
FN_$INIT_TIMER
FOR A IN YACHTS
FOR B IN YACHTS
   IF B.BUILDER = A.BUILDER AND B.RIG GT A.RIG
   THEN PRINT B.BUILDER, A.RIG, B.RIG
FN_$SHOW_TIMER
END_PROCEDURE
```

However, a PRINT statement with a CROSS clause in an RSE achieved the same result with the expenditure of only 3.02 seconds of CPU time:

```
DTR> SHOW TIME1B
PROCEDURE TIME1B
FN_$INIT_TIMER
PRINT BUILDER, A.RIG, RIG OF A IN YACHTS CROSS
   B IN YACHTS OVER BUILDER WITH A.RIG GT B.RIG
FN_$SHOW_TIMER
END_PROCEDURE
```

In these procedures, FN$INIT_TIMER starts timing the processing of the records and FN$SHOW_TIMER displays the elapsed time following completion of the processing.

12.5 DATATRIEVE's Evaluation of Compound Booleans

DATATRIEVE sets up a priority when it evaluates compound Boolean expressions that include key fields. For any domain, the key that is chosen depends on three factors:

- Exact or range retrieval
  - Exact retrievals use EQUAL or STARTING WITH.
  - Bounded range retrievals use BT.
  - Range retrievals use GT, GE, LE, or LT.

Key is NO DUP or DUP

Primary or alternate key

:yed retrieval is performed on Booleans that use the relational operators QUAL, STARTING WITH, BEFORE, AFTER, GT, GE, LE, LT, or BT. Table -1 indicates DATATRIEVE's priority in choosing keys. Each line represents a mbination of the three attributes noted. The lines of the table are arranged in der of diminishing priority.

ble 12-1: DATATRIEVE's Priority in Choosing Keys

'ype of Retrieval	Dup/No Dup	Type of Key
:xact	NO DUP	Primary
		Alternate
	DUP	Primary
		Alternate
;ounded Range	NO DUP	Primary
		Alternate
	DUP	Primary
		Alternate
:ange	NO DUP	Primary
		Alternate
	DUP	Primary
		Alternate

12.6 Summary of Rules

The following guidelines can help you take advantage of DATATRIEVE's ability to use a key-based index to retrieve records:

- When defining data, make the field most commonly used in queries the primary key. If that field does not uniquely determine a record, combine it with another field so the combined fields uniquely determine a record. Allowing duplicate values of a primary key slows performance.
- If you decide to make a group field the primary key, the order of the subordinate elementary fields is important. The field most commonly used in queries should be the first elementary field listed. Remember that DATATRIEVE cannot do keyed access on group field keys that contain numeric items.
- If there are other fields that will often be used with the primary key in queries, you can designate them as alternate keys.
- Use EQUAL (=) instead of CONTAINING (CONT) in the Boolean expression of an RSE, when searching for records based on a key field value.
- When searching for field values beginning with a specified substring, use STARTING WITH instead of CONTAINING (CONT). This rule is most important when your search is based on a key field.

DATATRIEVE allows you to relate records from the same domain or two different domains with the CROSS clause or nested FOR loops. When the relationship is based on a key field of at least one of the domains, keep these guidelines in mind:

- If the field is a key for only one of the domains, make sure that domain is specified first in the CROSS clause or included in the first FOR loop.
- Use a domain rather than a collection as the second record source. DATATRIEVE cannot do keyed access on collections. A collection, however, can help performance when it greatly reduces the number of records that DATATRIEVE must evaluate in a relational query. In addition, forming and naming a collection is useful if you need to use the same subset of records several times within a DATATRIEVE session.
- Try not to use a conditional statement following nested FOR loops or following a FOR loop that contains an RSE with a CROSS clause. A better approach is to include the conditional test in a Boolean expression within the RSE in the CROSS clause or in the second FOR loop.
- When relating two or more record streams, do not specify the largest record stream in the first position of the CROSS clause or in the first FOR loop.

Part 5
DATATRIEVE and the VAX Information Architecture

Using Forms with DATATRIEVE 13

A form is a terminal screen image used to display and collect information. You can use forms to display, modify, and store data managed by DATATRIEVE.

You can often format a data display more attractively using a form image than you can without one. This is particularly true when you need to display records longer than the maximum number of characters your terminal screen can accommodate on one line. In addition, nontechnical users are often more comfortable entering data through a form interface. They can see all the fields requiring input and can judge the size of each field before they begin to enter data. If they are modifying or storing data and make errors entering data in a field, they can back up to that field and correct the error.

To use forms with DATATRIEVE, you must have VAX TDMS software or VAX FMS software (called simply TDMS and FMS) installed on your system. When you install DATATRIEVE, you specify which of these forms products you want to use.

To create a forms application, you need to:

- Use a forms editor to define a form
- Insert the form definition in a request library file (TDMS) or form library (FMS)
- Associate the form definition with a DATATRIEVE domain
- Make sure that SET FORM is in effect when your application executes

The following sections discuss each of these steps in greater detail. Because your form definition depends, at least partly, on how you plan to use the form, step 3 is discussed first.

13.1 Associating a Form with a Domain

You can use a form to display data from RMS domains, view domains, DBMS domains, Rdb domains, and remote domains.

When working with remote domains, you can use forms to store data into a remote domain and to display a selected record or a record from a record stream containing no other records. You cannot, however, use forms to display group fields from remote domains.

There are two ways you can associate a form definition with a domain:

- Use the FORM IS clause within a domain definition to identify a form with a particular domain. DATATRIEVE uses that form with any STORE, MODIFY, DISPLAY, or PRINT statement that refers to that domain.
- Use the DISPLAY_FORM statement within a DATATRIEVE statement to map data to and from specific form and record fields. For example, include the DISPLAY_FORM statement within a FOR, STORE, or MODIFY statement.

Using either method, you must specify both the name of the form and the file specification of the library file containing the form. There are advantages to using each method. The FORM IS clause lets you use a form by specifying a single line of syntax in a domain definition. The DISPLAY_FORM statement lets you specify exactly which fields you want to map between a form and a record and lets you associate more than a single form with a domain.

Note

DISPLAY_FORM is the method you should use if you intend to map numeric data between forms and records. Using the DISPLAY_FORM statement with the FORMAT value expression ensures that decimal points and signs are mapped correctly. FORMAT value expressions are discussed in the *VAX DATATRIEVE Handbook* and the chapter on value expressions in the *VAX DATATRIEVE Reference Manual.*

You can see examples of what form displays look like with some of the sample DATATRIEVE domains. To see the examples, make sure your system has a version of DATATRIEVE installed with the TDMS or the FMS forms interface

d follow these steps:

Set your default VMS directory to one that contains the data files for the YACHTS, SAILBOATS, and FAMILIES domains. If you do not have these data files in one of your directories, set your default directory to the system directory with the sample data:

```
$ SET DEFAULT DTR$LIBRARY
```

or

```
$ SET DEFAULT SYS$COMMON:[DTR]
```

Invoke DATATRIEVE and set your current dictionary to the sample forms dictionary:

```
DTR> SET DICTIONARY CDD$TOP.DTR$LIB.FORMS
```

Make sure that the form setting is in effect for your session. (SET FORM is the DATATRIEVE default, but you may have run procedures that include the SET NO FORM command.)

```
DTR> SET FORM
```

Ready the YACHTS, SAILBOATS, or FAMILIES domain.

Print some records from the domain. Records appear one at a time. Press the RETURN key each time you want to display a new record. You return to the DTR> prompt once all the records you requested have been displayed.

Press CTRL/C and then the RETURN key if you want to stop the display operation before all the records have been displayed.

1.1 The FORM IS Clause

ou include the FORM IS clause in a domain definition, you can have TATRIEVE automatically use the form to display records.

e syntax for defining a domain that automatically uses a form is:

```
FINE DOMAIN path-name USING record-path-name ON file-spec
ORM [IS] form-name [IN] form-library;
```

The following examples show the use of the FORM IS clause in domain definitions. Note that *form-library* can be a TDMS request library file or an FMS form library. The default file type for TDMS request library files is .RLB; the default file type for FMS form libraries is .FLB:

```
DEFINE DOMAIN YACHTS_F
 USING YACHT ON YACHT.DAT
 FORM IS YACHTF IN DTRFMS.FLB;

DEFINE DOMAIN PERSONNEL_F
 USING PERSONNEL_REC ON [MORRISON]PERSON.DAT
 FORM IS PERSON IN [KELLER]FORMSLIB;

DEFINE DOMAIN REMOTE_FAMILIES USING FAMILIES AT NOVA"LINTER TAD"
 FORM IS FAM IN NOVA"LINTER TAD"::DB3:[LINTER]DTRTDMS;

DEFINE DOMAIN SAILBOATS
 OF CDD$TOP.DTR$LIB.DEMO.YACHTS, CDD$TOP.DTR$LIB.DEMO.OWNERS BY
 01 SAILBOAT OCCURS FOR YACHTS.
   03 BOAT FROM YACHTS.
   03 SKIPPERS OCCURS FOR OWNERS WITH TYPE EQ BOAT.TYPE.
      05 NAME FROM OWNERS.
      FORM IS SAIL IN DTR$LIBRARY:FORMS
;

DEFINE DOMAIN PART USING PART OF DATABASE PARTS_DB
 FORM IS PARTF IN PARTS.FLB;
```

There are some disadvantages to using the FORM IS clause:

- The field names in the form definition must exactly match the correspondir field names in the record definition for the domain.
- You must define a form field for every field in the corresponding record to avoid unexpected mapping results.
- You cannot match form field names to names of REDEFINES fields or to query names.
- You may get unexpected results when mapping numeric fields.
- You cannot prevent the operator from entering data in all the fields on the form when you specify the MODIFY or STORE statements, unless the fiel were defined as Display Only by the form definer.
- When the operator presses the RETURN key or ENTER key after a STOR or MODIFY operation, data is returned from all the fields on a form, includ ing data that may have been previously mapped to a Display Only field. Th may lead to unexpected input.
- You cannot use more than the single form that you specify in the domain definition to store or modify data associated with that domain.

- You cannot ready a domain if it contains a FORM IS clause and DATATRIEVE has not been installed with a forms package.
- You may have to take special steps to use the FORM IS clause with applications that use FMS and callable DATATRIEVE. See Section 13.4.6.1 for more information.
- You must exercise caution when your application modifies view domains that use the FORM IS clause. See Section 13.4.6.3 for more information.

13.1.2 The DISPLAY_FORM Statement

The format of the DISPLAY_FORM statement is:

```
DISPLAY_FORM form-name [IN] form-library          <--- 1
        [USING statement-1]                       <--- 2
        [RETRIEVE [USING] statement-2]            <--- 3
```

Form-library can be a TDMS request library file or an FMS forms library. The default file type for TDMS request library files is .RLB; the default file type for FMS form libraries is .FLB. In the following example, parts corresponding to the three sections of the statement format are labeled for clarity:

```
TR> MODIFY YACHTS USING
ON>                                          -+
ON>  DISPLAY_FORM YACHT_FORM IN FORMSLIB      |  1
ON>                                          -+
ON>    USING                                  |
ON>      BEGIN                                |
ON>       PUT_FORM MANUFACT = MANUFACTURER    |  2
ON>       PUT_FORM MODEL    = MODEL           |
ON>       PUT_FORM PRICE    = PRICE           |
ON>      END -                                |
ON>                                          -+
ON>    RETRIEVE                               |
ON>       BEGIN                               |  3
ON>         PRICE = GET_FORM PRICE            |
ON>       END                                 |
TR>                                          -+
```

Note that, because you specify both record and form field names in the DISPLAY_FORM syntax, form and record field names need not match.

The DISPLAY_FORM statement gives you four kinds of control not provided by the FORM IS clause of the domain definition:

You can associate more than one form definition with a domain. Therefore, you can design a variety of forms to fit different purposes.

The first part of the DISPLAY_FORM statement format gives you this control. In the example, the DISPLAY_FORM statement is embedded in a

MODIFY statement and associates the form YACHT_FORM with the domain YACHTS.

- You specify which record field values you want to map to or display on the form.

  Only values from record fields you explicitly assign to form fields are displayed on the form. This allows you to mask sensitive data from certain users. The USING clause, number 2 in the format diagram, gives you this control. In the example, only the values for the record fields MANUFACTURER, MODEL, and PRICE are displayed on their corresponding form fields.

  If you omit the USING clause of the DISPLAY_FORM statement, no record field values are displayed on their corresponding form fields. You might want to omit the USING clause if you are storing records rather than modifying records.

- You specify the form fields you want to return to the record.

  Only values from form fields explicitly assigned to record fields are returned to the record. The RETRIEVE clause, number 3 in the format diagram, gives you this control. In the example, only the value in the form field PRICE replaces the existing value in the record.

  If you omit the RETRIEVE clause of the DISPLAY_FORM statement, no form field values entered by the user or sent to the form by DATATRIEVE are returned to their corresponding record fields. Omit the RETRIEVE clause if you do not want to collect data from the form, and you want to prevent inadvertent data modification during a display-only operation.

- You specify the format to display, store, and modify numeric fields correctly.

  Using the FORMAT value expression with the DISPLAY_FORM statement lets you control data transfer between numeric record fields and form fields. (See Section 13.2.2.1 for information on mapping numeric data types and 13.4.5 for information on using the FORMAT value expression.)

The only time you can use the DISPLAY_FORM statement without embedding it in another statement and without using the optional USING and RETRIEVE clauses is when you simply want to display the form itself. In this case, you must end the statement with a semicolon (;) to avoid getting an error message. For example:

```
DTR> DISPLAY_FORM YACHT IN DTR$LIBRARY:FORMS;
```

This type of DISPLAY_FORM statement lets you see the form before you use it, without exiting from DATATRIEVE. You cannot enter data in the form that is

displayed. You can use this particular DISPLAY FORM syntax to display a form that is associated with a domain by the FORM IS clause.

13.2 Defining Forms

A form definition contains information that specifies:

- The screen image of the form. The screen image includes background text and the pictures of the fields in which data can be collected and displayed.
- The length and data type of each field.
- A set of attributes for each field.
- Video highlights for the form.
- The name of a help form that the operator can display.

You can use the TDMS Form Editor or the FMS Form Editor to define a form. The *VAX TDMS Forms Manual* explains how to use the Form Definition Utility (FDU) to create and modify a TDMS form. The *Introduction to VAX FMS* explains how to create and modify an FMS form.

Sections 13.2.1 to 13.2.6 discuss some considerations you should keep in mind when creating a form to use with DATATRIEVE. The discussion assumes you are familiar with the process of creating a form definition. If you are unfamiliar with the forms product you intend to use with DATATRIEVE, you should create a few sample forms before reading these sections.

3.2.1 Defining Form Field Names

You specify form field names in the Assign phase of your form definition. Any names you specify during the Layout phase of your form definition are background text, not field names.

If the form you are creating will be referred to in the FORM IS clause of a domain definition, use the following guidelines when naming form fields:

If there are any hyphens in the DATATRIEVE field name, they should be defined as underscores to FMS because DATATRIEVE converts all hyphens to underscores when it converts names to uppercase.

Form field names can be up to 31 characters long. The form field names must match the field names in the DATATRIEVE domain.

If you convert an FMS Version 1 form definition to FMS Version 2 or TDMS and you want to associate the converted form with a DATATRIEVE domain,

make sure that the form field names match the record field names. Suppose, for example, you have an FMS Version 1 form with the form field MANUFA corresponding to the record field MANUFACTURER. If you convert that form to FMS Version 2 or TDMS, be sure to edit your new form definition and change the name of the form field to MANUFACTURER.

- When using FMS to define a form, make sure that you define a form field for each record field. When using TDMS, make sure that you define a record field for each form field. When the record field is a COMPUTED BY field, define a counterpart form field and select the Display Only attribute. The Display Only attribute prevents users from modifying the field's data.

  In FMS, if you do not define a form field for each record field, incorrect values may be mapped to record fields.

If the form you are creating will be referred to only in DISPLAY_FORM statements, the DATATRIEVE field names and form field names need not match. In addition, you do not have to define a form field for each record field.

When you are using the DISPLAY_FORM statement, you can define a form field for a DATATRIEVE variable. You may want to define a form field that maps to a variable to use in conditional statements. For example, you could collect an employee ID from a form field and return it to a variable. You could then use this ID to search a domain and display related employee data on the same form. Remember, however, that DATATRIEVE does not process form field values when the form user presses TAB to move to another field. The user must press the RETURN key before DATATRIEVE can evaluate and process data. In this example, the user would have to enter the employee identification code, ignore the remaining fields on the form, and press RETURN.

If you do not spell a record field name correctly (in DISPLAY_FORM assignment statements) or do not spell a form field name correctly (either in the form definition referred to by the FORM IS clause or in the DISPLAY_FORM assignment statements), DATATRIEVE ignores the values in those fields. You receive no error messages to alert you to possible field name errors.

13.2.2 Defining Data Type and Length of Form Fields

When you define a form, you match the form fields with the corresponding record fields in two ways:

- Use a form field picture that describes the same data type as the record definition. The field pictures control the type of data a user enters in the form field at run time.

  For example, if you define a form field of all 9s, a user cannot move to the next form field if they enter anything but a digit in that form field.

Though defining a form field picture controls the type of data a user enters, it is not the same as defining a data type. For example, you can match an alphabetic field picture on a form with a record definition of PIC A, or you can match a field picture of 9s on a form with a DATATRIEVE record definition of PIC 9s. DATATRIEVE receives data from both TDMS and FMS as strings, however, so you cannot completely match form and record definition data types. For example, if your record definition defines a field as PIC 9(4) USAGE COMP, when you define a form field you can specify the form field characters 9999, but you cannot match COMP.

Make sure your form field has the same length as your record field.

For example, if the record field is defined as PIC X(25), use 25 Xs for the form field. If the form field is longer than the record field, the form user might receive a truncation error message and reenter prompt after pressing the RETURN key. If the form field is shorter than the record field, existing values for alphanumeric fields are truncated in the form display. A form field shorter than the record field might make it impossible for the form user to enter a valid value.

ote that regardless of the form pictures or record data types you assign, ATATRIEVE passes all values between forms and records as text strings. This ects how it handles numeric field values, as discussed in the following sections data types requiring special treatment in form definitions.

.2.2.1 Numeric Fields with Decimal Points or Signs -- When matching meric record fields that contain signs or decimal points with form field pictures, u may want to follow some of the guidelines in this section. *These guidelines sume that you are using the DISPLAY FORM statement with the FORMAT lue expression.* This is the recommended method of handling numeric data es. (See Section 13.4.5 for examples of the FORMAT value expression.)

For numeric record fields that contain an implied decimal point (such as a V in the PICTURE clause or a SCALE clause) but not a sign, you can include a decimal field-marker character (FMS) or a decimal constant (TDMS) in the corresponding place on the form field.

You do this during the Layout phase of defining your form.

In the Assign phase of your form definition, you might decide to avoid the fixed decimal characteristic. Because values are passed from the form to DATATRIEVE as text strings, specifying fixed decimal does not scale numeric values for storage.

If you do assign the fixed decimal attribute to a numeric form field, perhaps because the form might also be used in applications other than

DATATRIEVE, follow these guidelines:

- Define the associated record field with an implied decimal field, for example, PIC 99V99.
- Define the form field, if it is an FMS form, as 99.99 with fixed decimal clear character 0, and zero fill attributes.
- Define the form field, if it is a TDMS form, as 99.99 with Fixed Decim and Zero Fill attributes.

- If you want a field to have both a decimal point and a sign (either explicit or implicit), in FMS use N instead of 9 for the form field.

  In TDMS, you cannot combine a signed numeric field picture (N) with the Fixed Decimal attribute. To get the effect of a scaled signed number, you ca use the N field without the Fixed Decimal attribute in TDMS. TDMS assigns a scale factor to fields you describe with an N picture based on the field picture. For a field you describe as NNN.NNN in the Layout phase of TDMS, it defaults a scale factor of -3, while a field you describe as NN.NN assigned a scale factor of -2.

Table 13-1 provides examples that match form fields to numeric record fields when using the DISPLAY FORM statement. Use the FORMAT value expressio as illustrated in the example following Table 13-1 and in Section 13.4.5 to ensur correct mapping between numeric fields.

Table 13-1: Matching Form Field Definitions to Numeric Record Fields

Record Field	Form Field	Form Field Attributes
PIC 999V99	999.99	You can use 9 or N because the data is unsigned.
PIC S9(4)V99	NNNNN.NN	An N is required for a signed field. The decimal point must agree with the record field.
WORD SCALE -3	NNN.NNN	The field must be large enough for the maximum value, sign, and decimal places.
REAL	NNNNNNNNN	The number of Ns is flexible, but no decimal point is included because the record field contains no implied decimal point.
LONG SCALE -4	NNNNNNN.NNNN	Similar to the preceding WORD example.

/hen you use the DISPLAY_FORM statement to map numeric fields use the fol-
wing technique:

When mapping fields to a form, use the FORMAT VALUE expression with an edit string and multiply the data from the record field (as described by the edit string) by the scaling factor

When getting the fields from a form, you divide by the scaling factor before storing the data.

he following procedure illustrates a typical method for mapping numeric fields
sing a form.

```
EFINE PROCEDURE FORMAT
ADY TEST_FORMAT WRITE
GIN
R TEST_FORMAT
GIN
ISPLAY_FORM TEST_FORMAT_FORM IN
LINTOV]TESTFORM.RLB USING
BEGIN
    PUT_FORM UNSIGNED_FIXED_DEC = -
       FORMAT(100 * UNSIGNED_FIXED_DEC) USING 99999
    PUT_FORM SIGNED_DECIMAL = -
       FORMAT(100 * SIGNED_DECIMAL) USING S999999
    PUT_FORM SCALE_FACTOR_3 = -
       FORMAT(1000 * SCALE_FACTOR_3) USING S99999
    PUT_FORM PLAIN_N = PLAIN_N
    PUT_FORM SCALE_FACTOR_4 = -
       FORMAT(10000 * SCALE_FACTOR_4) USING S99999
END RETRIEVE USING
BEGIN
    UNSIGNED_FIXED_DEC = -
       (GET_FORM UNSIGNED_FIXED_DEC/100)
    SIGNED_DECIMAL = -
       (GET_FORM SIGNED_DECIMAL/100)
    SCALE_FACTOR_3 = -
       (GET_FORM SCALE_FACTOR_3/1000)
    PLAIN_N = PLAIN_N
    SCALE_FACTOR_4 = -
       (GET_FORM SCALE_FACTOR_4/1000)
END
D
D-PROCEDURE
```

.2.2.2 Usage DATE Fields -- You should define date fields as 11 Xs on your rms. DATATRIEVE displays the field in the default date format (DD-MMM-YYY). Use 23 Xs to display both date and time.

If you want to display the date in other than the default format, you can do so with the DISPLAY_FORM statement. Define a variable that is computed by a format value for the date field and then display the variable rather than the date field on the form. In the following example, FIELD2, rather than FIELD1, can be mapped to the form:

```
Record field:

03  FIELD1 USAGE DATE.
```

Variable:

```
03  FIELD2 COMPUTED BY FORMAT(FIELD1) USING MMMBDDBYYYY.
```

You might want to add a COMPUTED BY date field such as FIELD2 to the record definition if your installation uses the specified date format as the company standard.

When the date field is defined as 11 Xs on your form, the form user can enter one of a variety of values and DATATRIEVE stores the date correctly. For example, DATATRIEVE stores the date value April 16, 1972 for any of the following entries:

```
4/16/72

16 4 1972

APR 16 1972

1972/APR 16
```

You can also use both the PUT_FORM and GET_FORM components of the DISPLAY_FORM statement to map date fields. (See Sections 13.4.3 and 13.4.4)

13.2.3 Specifying User Entry and Validation Criteria

DATATRIEVE validation clauses (in the record definition or DATATRIEVE statement) are not applied to any field until the form user finishes with a record and presses the RETURN key. After the RETURN key is pressed, DATATRIEVE looks at the data returned to the record fields and applies the record or statement validation clauses. If data in a field is invalid, the user is prompted again for valid data for that field.

Validation associated with the form fields can prevent users from entering incorrect data. Using 9s or Ns for numeric form fields, for instance, ensures that form users cannot tab to the next form field if they accidently enter nonnumeric characters. Even though the Fixed Decimal attribute is not passed to DATATRIEVE when field values return from the form, you can specify the attribute in your form definition if you decide it helps the user enter data correctly.

hoosing the Fixed Decimal, Right Justify, Left Justify, and Zero Fill attributes l affect how the user enters data and therefore can affect what data is returned) the record field.

ssign the Display Only attribute to any fields that the user cannot store into or odify. This will prevent the user from entering data in a field. Your ISPLAY_FORM statement can prevent the user modifications from reaching ıe stored record, but cannot prevent the user from entering that data in the form ɔld. It is also good practice to prevent form users from entering changes that are ɔt stored.

Note

TDMS allows users to assign Field Validators to form fields. DATATRIEVE ignores these validators, however, and does not use them in any DATATRIEVE/TDMS application.

).2.4 Defining Multiple Screen Forms and Forms with Scrolled Areas

ɜither FMS nor TDMS supports forms that span multiple screens. ATATRIEVE does not support forms that contain scrolled regions.

you modify and store data using the DISPLAY_FORM statement, however, ıu can display a series of individual forms to collect and display data within a ıgle procedure. Note that only one form can appear on the screen at a time.

;.2.5 Using Default Values

ATATRIEVE's default value and missing value are displayed on the form fields ıen you use the FORM IS clause or the PUT_FORM component of the SPLAY_FORM statement. The form user generally sees default and missing lues only during a store operation. Do not specify default values in your form finition. DATATRIEVE does not store these values in the record.

.2.6 Defining Forms for Domains That Contain Repeating Fields

ɔu can define a matching form field for a repeating record field (one that :ludes an OCCURS clause). For example, a form for the FAMILIES domain ıld specify KID_NAME and AGE as indexed fields. You create an indexed ele- ɜnt to match each occurrence of the record field.

you are using TDMS, align all occurrences of the repeating fields vertically or rizontally in the Layout phase of your form definition. Then, in the Assign ase, specify the index attribute for the repeating items.

If you are using FMS, follow these steps:

1. In the Layout phase of your form definition, specify the background text and field characters for the first occurrence of the repeating item.
2. Enter the Assign phase, and specify the attributes you want for the item. Type 1 for the index attribute.
3. Enter the Layout phase again and use the form editor's cut and paste function to create all additional occurrences of the repeating item. FMS automatically assigns the correct attributes for these additional occurrences.

Whether you are using TDMS or FMS, make sure you create enough repeating form fields to accommodate the maximum number of occurrences defined in the record definition.

13.3 Inserting Forms in Library Files

After you define a form with the TDMS or FMS Editor, you must insert it in a form library. The following two sections describe how to do this for both TDMS and FMS forms.

13.3.1 Inserting Forms in TDMS Library Files

To insert a TDMS form in a library, use the TDMS Request Definition Utility (RDU) to create a request library definition and build a request library file.

For example, to use the form definitions YACHT_FORM and PERSON in a DATATRIEVE application:

1. Use RDU to define a request library:

```
RDU> CREATE LIBRARY DTR_TDMS
RDUDFN>     FORM IS YACHT_FORM;
RDUDFN>     FORM IS PERSON;
RDUDFN>     FILE IS "FORMSLIB";
RDUDFN>     END DEFINITION;
RDU>
```

2. Build the request library file:

```
RDU> BUILD LIBRARY DTR_TDMS
```

   Information from the form definitions YACHT_FORM and PERSON is built into the request library file FORMSLIB.RLB.

3. Use DATATRIEVE to display the form:

```
DTR> DISPLAY_FORM YACHT_FORM IN FORMSLIB;
```

ep in mind that a TDMS request library definition and the request library file ıtain other information and instructions, but DATATRIEVE uses only the ref- ːnce to the TDMS form.

ɣou modify your form definitions in any way, you must rebuild the request rary file. To do this, enter RDU and rebuild the request library using the same JILD LIBRARY command. For example:

```
J> BUILD LIBRARY DTR_TDMS
```

)MS extracts your latest form definitions from the CDD to include in the rary.

r more information on using RDU to create and modify request libraries, refer the *VAX TDMS Request Manual.*

r information about converting FMS forms for use with TDMS and \TATRIEVE see the discussion of conversion command procedures in the *VAX)MS Forms Manual.*

.3.2 Inserting Forms in an FMS Library

u use the FMS/LIBRARY command to perform operations on form files and m libraries. After you have defined an FMS form, create a form library for the m definition and insert the forms:

```
?MS/LIBRARY/CREATE FORMSLIB.FLB YACHTF,PERSON
```

IS inserts the form definitions YACHTF and PERSON in the form library)RMSLIB.FLB.

ɣou modify the form, replace the modified form in the library. For example:

```
?MS/LIBRARY/REPLACE FORMSLIB.FLB PERSON
```

r information about converting FMS Version 1 and Version 2 form definitions TDMS form definitions, refer to the discussion of the Form Converter Utility the *VAX FMS Utilities Reference Manual* or the discussion of conversion com- ınd procedures in the *VAX TDMS Forms Manual.*

r information about converting FMS Version 1 forms to FMS Version 2 forms, er to the *VAX FMS Utilities Reference Manual.*

.4 Using Forms to Display and Collect Data

ːer you define a form and insert the form definition in a form library, you can ː the form. The following sections explain how to use DATATRIEVE com- ınds and statements to display, collect, store, and modify data on a form.

13.4.1 Enabling and Disabling Form Use

The command SET [NO] FORM determines whether DATATRIEVE uses a form If SET FORM is in effect and you ready a domain whose definition includes the FORM IS clause, or you use the DISPLAY_FORM statement, DATATRIEVE opens the form library specified.

If SET NO FORM is in effect, DATATRIEVE does not open a form library, and you cannot use a form. When you are using a domain whose definition includes a FORM IS clause, it is sometimes useful to review the contents of many records quickly . SET NO FORM allows you to override form display and use regular screen display.

Note that if you use a DATATRIEVE image installed without a forms package, using SET NO FORM does not let you use a domain definition that contains a FORM IS clause. You must edit the domain definition to remove the form reference or install DATATRIEVE with a forms package.

The default is SET FORM.

You can see if SET FORM is in effect by using the SHOW SET_UP command. You can see which forms are currently loaded by entering SHOW FORMS. To release a form loaded with the DISPLAY_FORM statement from your workspace use the RELEASE form-name command.

13.4.2 Displaying Data with Forms

When DATATRIEVE uses forms to display records, only one record at a time is displayed on the screen. To proceed to the next record, press the ENTER key or the RETURN key. If you want to skip the rest of the records, position the cursor on a nonnumeric field and press CTRL/C and then the RETURN key.

The following example shows how to display records from a domain whose definition includes the FORM IS clause:

```
DTR> SET FORM
DTR> READY YACHTS_FORM
DTR> PRINT FIRST 10 YACHTS_FORM
```

The PRINT statement in this example causes DATATRIEVE to use a form to display the first record in YACHTS_FORM. DATATRIEVE displays all the field you included in your form definition. You can press the RETURN key to display the next record. Continue pressing the RETURN key until the tenth record is displayed. The next time you press the RETURN key, you get the DTR> prompt.

f you want to display values for only a few fields, you can use a form with a)ATATRIEVE view domain or you can use the DISPLAY_FORM statement. 'our DISPLAY_FORM statement can refer to an entirely different form than the ne specified in the FORM IS clause. It can also refer to the same form.

'or example, suppose you want to display only the type and price of the first ten achts on the YACHT_FORM form. The fields MANUFACTURER, MODEL, nd PRICE in YACHTS correspond to the fields MANUFACT, MODEL,and 'RICE in the YACHT_FORM form. Use the following statements:

```
TR> FOR FIRST 10 YACHTS
ON> DISPLAY_FORM YACHT_FORM IN FORMSLIB USING
ON>     BEGIN
ON>         PUT_FORM MANUFACT = MANUFACTURER
ON>         PUT_FORM MODEL    = MODEL
ON>         PUT_FORM PRICE    = PRICE
ON>     END
```

Vith the DISPLAY_FORM statement, you can use a number of forms to display ata in one domain. For example, suppose you create two forms for YACHTS: 'YPE and SPECS. The form TYPE contains the fields MANUFACT and IODEL, corresponding to the MANUFACTURER and MODEL fields in 'ACHTS. The form SPECS contains fields that correspond to the remaining 'ACHTS fields.

'he following example shows how to display each of the first 10 YACHTS records n two forms. For each record in the FOR loop, first the TYPE form is displayed nd then the SPECS form is displayed:

```
OR FIRST 10 YACHTS
  BEGIN
  DISPLAY_FORM TYPE IN FORMSLIB USING
       BEGIN
       PUT_FORM MANUFACT = MANUFACTURER
       PUT_FORM MODEL    = MODEL
       END
  DISPLAY_FORM SPECS IN FORMSLIB USING
       BEGIN
       PUT_FORM LENGTH     = LOA
       PUT_FORM DISPLACE   = DISPLACEMENT
       PUT_FORM RIG        = RIG
       PUT_FORM BEAM       = BEAM
       PUT_FORM PRICE      = PRICE
       END
  END
```

lote that when you use the DISPLAY_FORM statement, the names of the fields ı the domain and the names of the form fields need not match. For example, you an design a form, PAY_FORM, that contains the fields BADGE, START, and AY. These fields correspond to the fields ID, START_DATE, and SALARY in ıe PERSONNEL domain, as illustrated in Figure 13-1.

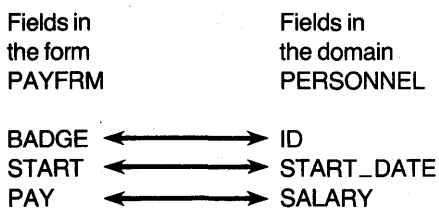


Figure 13-1: Corresponding Fields in a Domain and Form

The next example shows how to display the fields ID, START_DATE, and SALARY from PERSONNEL using the form PAY_FORM:

```
DTR> READY PERSONNEL
DTR> FOR PERSONNEL
CON> DISPLAY FORM PAY_FORM IN FORMSLIB USING
CON>    BEGIN
CON>       PUT_FORM BADGE = ID
CON>       PUT_FORM START = START_DATE
CON>       PUT_FORM PAY   = SALARY
CON>    END
```

13.4.3 Storing Data with Forms

To store records in a domain that is defined to include a FORM IS clause, use the STORE statement:

```
DTR> READY YACHTS_FORM FOR WRITE
DTR> STORE YACHTS_FORM
```

DATATRIEVE displays the YACHT_FORM form, including any default and missing values you specify in your record definition. Then DATATRIEVE waits for you to enter data.

While entering data, you can use:

- The TAB key to move to the next field
- The BACKSPACE key to move to the previous field
- The right and left arrow keys to move within a field
- The LINE FEED key to delete the contents of a field
- CTRL/C to stop storing and prevent the current record from being stored

When you finish entering data, press ENTER or the RETURN key. DATATRIEVE attempts to store the record as it appears on the screen. If there

e any validation errors, DATATRIEVE displays an error message at the bottom the screen and lets you change the field that caused the error.

)te that the FORM IS clause does not give you field-level access in a store eration. If you enter a STORE USING statement and the FORM IS clause is ur only association of a domain with a form, DATATRIEVE does not display e form. If you want a STORE USING statement to display a form, you must :lude a DISPLAY_FORM statement. Interactive data entry on a form displayed th DISPLAY_FORM is the same as that described for FORM IS.

you use the DISPLAY_FORM statement to store, you must use its RETRIEVE ıuse.

ır example, suppose you receive information for YACHTS in parts. The first ormation you receive is the manufacturer, model, rig, and overall length. You ınt to store this information and later modify your records to include further ta.

ıu can define a new form, YACHT_FORM1 with the fields VENDOR, MODEL, G, and LENGTH. The following procedure shows how to use the form ACHT_FORM1 to store partial records in the domain YACHTS:

```
)CEDURE STORE_YACHTS_1
ADY YACHTS WRITE
)RE YACHTS USING
 DISPLAY_FORM YACHT_FORM1 IN FORMSLIB RETRIEVE USING
     BEGIN
       MANUFACTURER = GET_FORM VENDOR
       MODEL        = GET_FORM MODEL
       RIG          = GET_FORM RIG
       LOA          = GET_FORM LENGTH
     END
)_PROCEDURE
```

ing both the PUT_FORM and GET_FORM components of the SPLAY_FORM statement, you can store values in fields without data entry ım the form user. This is particularly useful when storing values into date fields d primary keys. If you specify the Display Only attribute for these fields in your m definition, you can prevent the form user from overriding the values you ıd to the form. By not selecting the Display Only attribute, you can allow the m user to modify such values.

r example, suppose you create the form PERSON to store data into RSONNEL. Figure 13-2 shows the fields in the form PERSON and responding fields in the domain PERSONNEL.

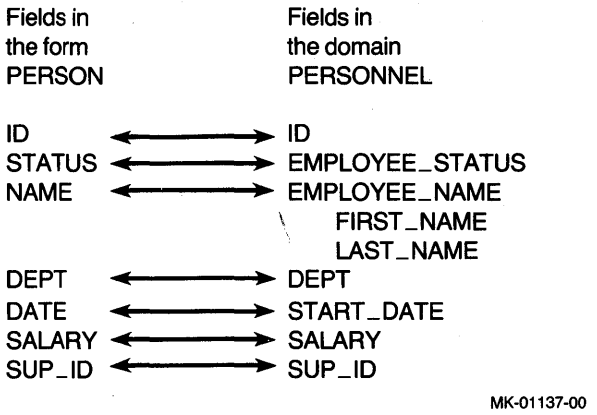


Figure 13-2: Corresponding Fields in the Form PERSON and the Domain PERSONNEL

The procedure STORE_PERSON shows how you can use the form PERSON to store PERSONNEL records:

```
! Task: store a new employee record. Generate a unique badge
! number, but allow the user to override it.
! The default starting date is "TODAY", but  the user can override
! that also.

PROCEDURE STORE_PERSON
STORE PERSONNEL USING
BEGIN

! Display the ID and DATE on the form.

  DISPLAY_FORM PERSON IN FORMSLIB USING
    BEGIN
        PUT-FORM ID     = 1 + MAX ID OF PERSONNEL
        PUT-FORM DATE   = FORMAT "TODAY" USING DD-MMM-YYYY
    END RETRIEVE USING

! The rest of the fields on the form are empty.
! Retrieve data the user enters on the form.

! The form PERSON has one field for a name. The domain
! PERSONNEL has fields for FIRST_NAME and LAST_NAME.
! Get the first and last name from the form name string.
```

```
    BEGIN

        DECLARE BLANK_POSITION WORD.
        BLANK_POSITION = FN$STR_LOCATE (GET_FORM NAME, " ")
        FIRST_NAME     = FN$STR_EXTRACT (GET_FORM NAME, 1,
                           BLANK_POSITION)
        LAST_NAME      = FN$STR_EXTRACT (GET_FORM NAME,
                           BLANK_POSITION + 1, 50)

        ID             = GET_FORM ID
        START_DATE     = GET_FORM DATE
        IF (GET_FORM STATUS  CONT "T")
            THEN STATUS = "TRAINEE" ELSE STATUS = "EXPERIENCED"
        DEPT           = GET_FORM DEPT
        SALARY         = GET_FORM SALARY
        SUP_ID         = GET_FORM SUP_ID
    END
:ND;
```

13.4.3.1 Storing Data in Hierarchical Records with Forms -- To store data in a hierarchical record, use the STORE statement and the DISPLAY_FORM statement. This procedure uses the DISPLAY_FORM statement to store data in the hierarchical record for the domain FAMILIES:

```
EFINE PROCEDURE STORE_LISTS
EADY CDD$TOP.DTR$LIB.DEMO.FAMILIES WRITE

 The variable, A, is used to establish context for the
 list field KIDS.

TORE A IN FAMILIES USING
EGIN
      DISPLAY_FORM FAMILY IN DTR$LIBRARY:FORMS RETRIEVE USING
      BEGIN
            MOTHER = GET_FORM MOTHER
            FATHER = GET_FORM FATHER
            NUMBER_KIDS = GET_FORM NUMBER_KIDS

 The MATCH statement transfers the data retrieved
 from the form with the GET_FORM KID_NAME and AGE value
 expressions to the KIDS fields.  In other words, each A.KIDS
 field value is transferred from the form to each KIDS record.

            MATCH KIDS, A.KIDS
              BEGIN
                   KID_NAME = GET_FORM KID_NAME
                   AGE = GET_FORM AGE
              END
      END
ND
ND_PROCEDURE
```

ee Chapter 6 for more information about hierarchical records.

13.4.4 Modifying Data with Forms

To modify records in a domain that uses a form, use the MODIFY statement:

```
DTR> READY YACHTS_F MODIFY
DTR> FOR YACHTS_F WITH BUILDER = *."BUILDER"
DTR> MODIFY
```

DATATRIEVE uses the form YACHTF to display the record you specify. You can now modify the fields in that record. To move through the form, use the same keys you use while storing.

If there are any validation errors, DATATRIEVE displays the error message at the bottom of the screen and lets you change the field that caused the error. If you try to modify a key field that is defined with the NO CHANGE attribute, however, DATATRIEVE prints an error message and does not modify any fields in the record. Because primary key fields are defined NO CHANGE by default, you lose all the modifications you make to a record when you try to modify its primary key field. If you do not change the primary key field, there is no problem. If you do try to modify a primary key field, DATATRIEVE returns an error and ignores the other modifications to the record. To avoid the error, you may want to define primary key form fields as display only form fields.

Note that the FORM IS clause does not give you field-level access in a modify operation. If you enter a MODIFY USING statement and the FORM IS clause is your only association of a domain with a form, DATATRIEVE does not display the form. If you want to use a MODIFY USING statement to display a form, you must include a DISPLAY_FORM statement.

You can include the DISPLAY_FORM statement within a MODIFY USING statement to modify data in a domain, whether or not a form is already assigned to the domain with the FORM IS clause.

If you use DISPLAY_FORM to modify, you include all sections of the statement. The PUT_FORM statements display the data users are to modify, and the GET_FORM value expressions store their changes.

In Section 13.4.3, the procedure STORE_YACHTS_1 created records with data for the fields MANUFACTURER, MODEL, RIG, and LENGTH_OVER_ALL. Suppose you now have data for the rest of the fields and want to update the records you stored. You do not want to modify the data already entered; you want to enter values only for the fields DISPLACEMENT, BEAM, AND PRICE.

One way to perform this task is to create a new form, YACHT_FORM2. This form contains fields that correspond to all the fields in YACHTS, as illustrated in Figure 13-3.

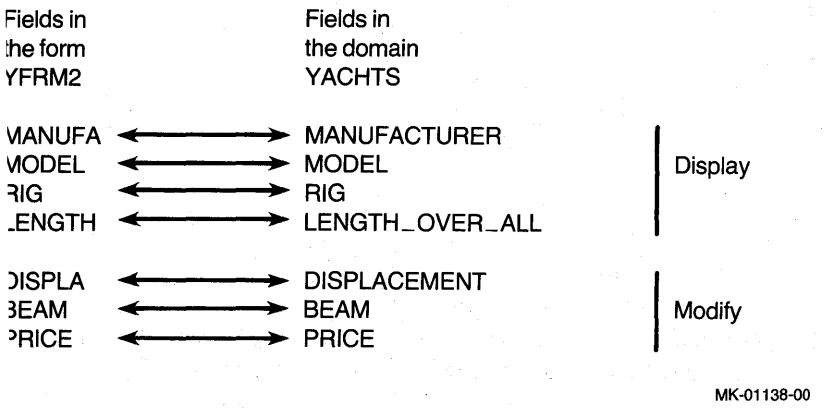


Figure 13-3: Corresponding Fields in YACHT_FORM2 and YACHTS

When creating the form YACHT_FORM2, you can assign the Display Only attribute to the first four fields. In this way, you allow the form user to enter data only for the last three fields.

The procedure STORE_YACHTS_2 shows how to modify data on a form:

```
ROCEDURE STORE_YACHTS_2
EADY YACHTS MODIFY

 Modify only YACHTS that are missing data.

OR YACHTS WITH DISPLACEMENT = 0
ODIFY USING
  BEGIN

 Use the form YACHT_FORM2 to display entire records.

  DISPLAY_FORM YACHT_FORM2 IN FORMSLIB USING

 Display the fields for which data has been entered.
 These fields are defined as Display Only to TDMS or FMS.
 The user cannot enter data in them.
 (If you do not define these fields as Display Only, the
 user can change the form fields. However, DATATRIEVE
 ignores the changes.)
```

(continued on next page)

```
        BEGIN
           PUT_FORM MANUFACT = MANUFACTURER
           PUT_FORM MODEL    = MODEL
           PUT_FORM RIG      = RIG
           PUT_FORM LENGTH   = LOA
        END   RETRIEVE USING

! Get new data from the form.

        BEGIN
          DISPLACEMENT = GET_FORM DISPLACE
          BEAM         = GET_FORM BEAM
          PRICE        = GET_FORM PRICE
        END
    END
END_PROCEDURE
```

13.4.4.1 Modifying Data in Hierarchical Records with Forms -- To modify data in a hierarchical record, use the MODIFY statement and the DISPLAY_FORM statement. This procedure uses the DISPLAY_FORM statement to modify data in the hierarchical record domain FAMILIES:

```
DEFINE PROCEDURE MODIFY_LIST
READY CDD$TOP.DTR$LIB.DEMO.FAMILIES WRITE
FIND ALL FAMILIES WITH FATHER CONT *."father's name"
FOR CURRENT
BEGIN
        MODIFY USING DISPLAY_FORM FAMILY IN DTR$LIBRARY:FORMS USING
        BEGIN
                PUT_FORM FATHER      = FATHER
                PUT_FORM MOTHER      = MOTHER
                PUT_FORM NUMBER_KIDS = NUMBER_KIDS
!
! The FOR loop establishes context for the KIDS field.
!
                FOR KIDS
                BEGIN
                        PUT_FORM KID_NAME = KID_NAME
                        PUT_FORM AGE      = AGE
                END
        END   RETRIEVE USING
        BEGIN
                FATHER      = GET_FORM FATHER
                MOTHER      = GET_FORM MOTHER
                NUMBER_KIDS = GET_FORM NUMBER_KIDS
```

(continued on next page)

```
The FOR loop establishes the context for retrieving
modified values from each occurrence of KIDS that
satisfies the RSE and for storing those values in
the KIDS record.

            FOR KIDS MODIFY USING
            BEGIN
                  KID_NAME = GET_FORM KID_NAME
                  AGE      = GET_FORM AGE
            END
      END
ND
ND-PROCEDURE
```

ee Chapter 6 for more information about hierarchical records.

3.4.5 Handling Numeric Data

lany numeric fields are stored with a fractional component and a sign. These elds may include a PICTURE string that explicitly specifies a sign and decimal oint, for example, PIC S999V99. However, numeric values for some fields iclude a sign and fractional component in storage, even though there is no picıre string to alert you to this fact. Some examples are record fields defined as SAGE REAL, USAGE WORD SCALE IS -2, or USAGE LONG SCALE IS -3. ection 13.2.2.1 explains how to define form fields that can contain both a sign ıd a decimal point.

you use the FORM IS clause to transfer data to and from record fields that clude a fraction and a sign in storage, results can be undesirable. As a general ıle, never include the FORM IS clause in a domain definition when its record efinition includes these numeric fields. Always use the DISPLAY_FORM statement to store, display, and modify such numeric fields. When you use the ISPLAY_FORM statement, you can assume control of the format and scale of ıe text string passed between the record field and the form field.

ıe following example illustrates how you handle numeric data that includes a ecimal point and a sign. The CB_BALANCE field in the CHECKBOOK domain defined as PIC S999V99. The associated BALANCE field on the CHECKS rm is defined as NNNN.NN.

ıe DISPLAY_FORM statement transfers the value from the record field as a ıole number and uses a FORMAT value expression to specify the text string as should appear on the form field. The edit string in the example includes a minus gn (-) as a numeric insertion character so that only negative values are displayed th a sign. As an alternative, you can use a plus sign (+) if you want positive as ell as negative values displayed with signs.

ecause the value CB_BALANCE is stored with two decimal digits, the JT_FORM assignment statement must multiply the field value by 100 before ansferring it to the form field. The Assignment statement in the RETRIEVE ction divides the whole number value from the form field by 100 before storing

it in the record field. Note that a FORMAT expression edit string is not necessary when the form field value returns to the record field:

```
DTR> MODIFY CHECKBOOK USING
DTR> DISPLAY_FORM CHECKS IN FORMSLIB USING
CON>    PUT_FORM BALANCE =
CON>       FORMAT (100 * CB_BALANCE) USING
CON>          -99999 RETRIEVE USING
CON>             CB_BALANCE = (GET_FORM BALANCE) / 100
```

You must multiply and divide values by the appropriate power of 10 for the recor field definition. The example multiplies and divides values by 100 because the record field stores two decimal places. If, for example, the record field you are handling includes a SCALE IS -3 clause, you multiply and divide values by 1000.

Form fields that include a decimal point and are defined as 9s require the same treatment in a DISPLAY_FORM statement. Because such fields cannot include a sign character, however, you omit the plus (+) or minus (-) character from the edit string in the FORMAT value expression.

13.4.6 Restrictions on Using Forms

The following sections describe restrictions on using DATATRIEVE with forms. They also provide examples of alternatives to these restricted uses of forms.

13.4.6.1 DATATRIEVE and FMS -- When you use the DISPLAY_FORM statement with FMS forms, DATATRIEVE passes a default field descriptor of 255 characters. If you try to concatenate fields from an FMS form, you get unexpected results. You can explicitly specify the description of a form field using the FORMAT value expression. Include an edit string in the USING clause of the FORMAT value expression so that DATATRIEVE does not use the 255-character default for an FMS form field.

The following example specifies field lengths for the fields MANUFACTURER and MODEL using FORMAT value expressions:

```
DECLARE FLD PIC X(30).
DISPLAY_FORM YACHT_FORM IN FORMSLIB;
  BEGIN
     PUT_FORM MANUFACTURER = MANUFACTURER
     PUT_FORM MODEL        = MODEL
  END RETRIEVE USING
     FLD = FORMAT (GET_FORM MANUFACTURER) USING X(20)| -
           FORMAT (GET_FORM MODEL) USING X(6);
```

When DATATRIEVE concatenates the MANUFACTURER and MODEL fields, uses 20 characters for MANUFACTURER and 6 characters for MODEL ıstead of 255 characters for each field.

n additional restriction concerns the use of FMS forms with the DATATRIEVE all Interface. If you have an application program that displays FMS forms and lso calls upon the DATATRIEVE Call Interface to display forms using the ORM IS clause in a domain definition, you must save and restore the FMS ter- ıinal control areas after each call to DATATRIEVE. This restriction applies only) use of the DATATRIEVE Call Interface with domains using the FORM IS ause. You can avoid the restriction by using DISPLAY_FORM instead of ORM IS.

ee the Introduction to VAX FMS for more information about FMS terminal con- ol areas. See the FMS documentation for more information about the restriction ı using the DATATRIEVE Call Interface.

3.4.6.2 DATATRIEVE Command Files and Forms Products -- This restriction ıd its workaround apply only to V2 and later of FMS and V1 and later of TDMS.

ATATRIEVE uses SYS$INPUT to get its commands from VMS command es. Both FMS and TDMS also use SYS$INPUT to get terminal input. ıerefore, you cannot use both DATATRIEVE and a forms product in the same ımmand file unless you set up that file using the following steps:

The default interactive assignment for SYS$INPUT is your terminal. When you run a command file, SYS$INPUT is the command file itself. Therefore, your command file must assign SYS$INPUT to SYS$COMMAND (the default device name of your terminal) so that the forms product can get input from the terminal. That assignment must precede the invocation of DATATRIEVE:

```
$ ASSIGN/USER_MODE SYS$COMMAND SYS$INPUT
```

Then you must include all the DATATRIEVE commands and statements in a procedure that is invoked by the command file.

ıe resulting command file takes this form:

```
ASSIGN/USER_MODE SYS$COMMAND SYS$INPUT
DTR EXECUTE procedure-name
```

13.4.6.3 Modifying Data Using View Domains and FORM IS -- You must be very careful when modifying records in a view based on more than one domain. The restriction documented here applies to modifying a field in a view when the modified field is the basis for selecting records from another domain. Although this restriction applies to views, it is included here to illustrate how the use of a form with views can mask modification errors.

You unintentionally modify data if you try to modify fields when all of the following conditions are true:

- You modify records in a view domain that uses FORM IS.
- The view selects records from another domain based on the value of a field i the view.
- You modify the field that forms the basis for selecting records from the second domain.

The following example shows what happens when you try to modify fields that refer to other domains in a view using a form. The sample view domain SAILBOATS has been edited to create the example.

The view domain containing a FORM IS clause:

```
DOMAIN SAILBOATS
 OF YACHTS, OWNERS BY
01 SAILBOAT OCCURS FOR YACHTS.
   03 BOAT FROM YACHTS.
   03 SKIPPERS OCCURS FOR OWNERS WITH BUILDER EQ BOAT.BUILDER.
      05 NAME FROM OWNERS.
    FORM IS SAIL IN DTR$LIBRARY:FORMS
;
```

SAILBOATS refers to the OWNERS domain based on the value for BUILDER.

Here are the OWNERS records before the modification:

```
DTR> READY OWNERS
DTR> PRINT OWNERS

OWNER
NAME       BOAT NAME        BUILDER      MODEL

SHERM MILLENNIUM FALCON ALBERG      35
STEVE DELIVERANCE       ALBIN       VEGA
HUGH  IMPULSE           ALBIN       VEGA
                .
                .
                .
```

/ou modify SAILBOATS:

```
TR> FOR FIRST 1 SAILBOATS WITH ANY SKIPPERS MODIFY
```

)ATATRIEVE displays the first record that meets the criterion WITH
3UILDER EQ BOAT.BUILDER on the form, and you modify the record to
ıclude a new value for BUILDER:

```
                    YACHT SPECIFICATION DATA


    Builder: ALBERG  ! Changed to ALBIN     Model: 37 MK II

    Length:  37          Beam:  12          Disp:  20000

    Rig:  KETCH                             PRICE: 36951

    Owners:  SHERM
```

)ATATRIEVE updates all the fields on a form when a you press ENTER or
.ETURN to complete data entry. Because you changed ALBERG to ALBIN, the
wner name from the updated record appears in the next record for which there is
BUILDER named ALBIN.

HERM now replaces STEVE in that OWNERS record.

```
[R> PRINT OWNERS

/NER
ıME      BOAT NAME       BUILDER      MODEL

[ERM MILLENNIUM FALCON ALBERG      35
[ERM DELIVERANCE       ALBIN       VEGA
     IMPULSE           ALBIN       VEGA
:M   EGRET             C&C         CORVETTE
                .
                .
                .
```

3.4.6.4 Special Graphics Characters in Forms -- When you design a form
ing characters from the VT100 Special Graphics set, the characters might not
ıtomatically work from one form invocation to the next. If this problem occurs,
ess CTRL/W to repaint the form.

Using DATATRIEVE with DBMS 14

his chapter describes the commands, statements, and clauses that let you use AX DATATRIEVE with VAX DBMS databases.

you already use DATATRIEVE, you can skip the section that deals with ırming a DATATRIEVE query statement. This chapter discusses basic DBMS ıncepts. For more information about DBMS concepts, read the *VAX DBMS ıtroduction to Database Administration* and *VAX DBMS Introduction to Data ſanipulation.*

you use DBMS but are not familiar with DATATRIEVE, you can supplement ıis chapter by reading the chapter on writing record selection expressions in this anual and Chapter 1 of the *VAX DATATRIEVE Handbook* on basic ATATRIEVE concepts. You can also read the chapter in the *VAX ATATRIEVE Guide To Writing Reports* on creating reports from a DBMS ıtabase.

this chapter, you learn to:

- Create a database definition in DATATRIEVE that represents a DBMS database
- Access the DBMS database either by readying it directly or by defining domains for each DBMS record and readying the domains
- Locate DBMS records in a variety of ways
- Print whole records or parts of records
- Store and modify records
- Erase records
- Connect, reconnect, and disconnect records from sets

- Define and access view domains and hierarchical DBMS records
- Create procedures and indirect command files that access DBMS records and sets

This chapter uses examples in the PARTS sample database included with the DATATRIEVE User Environment Test Package (UETP) and installed on your system. The following command sets the CDD default to the directory that contains the database domain definitions used in this chapter:

```
DTR > SET DICTIONARY CDD$TOP.DTR$LIB.DEMO.DBMS
```

14.1 Advantages of Using DATATRIEVE

Although VAX DBMS has its own interactive query language to access DBMS data, you may prefer to use DATATRIEVE. By using DATATRIEVE syntax with a few special DBMS extensions, you can:

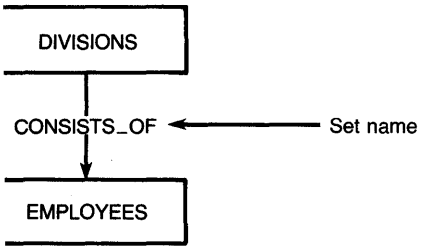
- Find and manipulate groups of records
- Modify, update, and erase DBMS data
- Create reports
- Plot graphs
- Relate data stored in RMS files and Rdb databases with data stored in DBMS databases
- Use FMS or TDMS forms to access and change DBMS data

Using DATATRIEVE, you can access individual records, groups of records, and records related according to information in a DBMS set. A *set* is a DBMS data structure that establishes a relationship among records.

For example, in the sample PARTS database, there is a set named CONSISTS_OF. This set, illustrated in Figure 14-1, contains pointers that identify which employees (in the EMPLOYEES domain) work for which divisions (in the DIVISIONS domain).

A single record in the DIVISIONS domain is called a *single record occurrence.* A single record occurrence is the data of a single record in the database. For example, the SOFTWARE division is a single record occurrence.

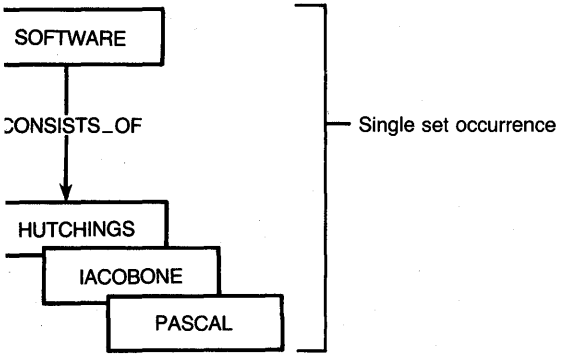
This single record occurrence is related to another set of records by the information in a *single set occurrence* of the set. A single set occurrence contains one owner record occurrence and zero or more member record occurrences.



MK-01130-01

igure 14-1: DBMS Set CONSISTS_OF

ı a DBMS database, relationships between records are always described in sets. or example, the single owner record occurrence of the SOFTWARE division is ɔnnected to employee records by the information in a single occurrence of the set ONSISTS_OF. Figure 14-2 shows this relationship.



MK-01140-01

gure 14-2: Single Set Occurrence

he information that employees HUTCHINGS, IACOBONE, and PASCAL work r the SOFTWARE division is described in the single set occurrence of the ONSISTS_OF set.

ı addition to manipulating data, a big advantage of using DATATRIEVE with BMS data is that you can easily format that data by using DATATRIEVE ·aphics and Report Writer. See the *VAX DATATRIEVE Guide to Graphics* and *AX DATATRIEVE Guide to Writing Reports* for information on using these atures of DATATRIEVE. DATATRIEVE provides specialized syntax to work

with DBMS records and these set relationships:

- Two define commands (DEFINE DATABASE and extensions to DEFINE DOMAIN)
- An extension to the READY command (READY database-path-name)
- Two SHOW commands (SHOW SETS and SHOW DATABASES)
- Three clauses that extend the record selection expression and refer to records as participants in sets (MEMBER, OWNER, and WITHIN)
- An extension to the STORE statement (CURRENCY clause)
- Three statements to work with sets (CONNECT, DISCONNECT, and RECONNECT)
- Two database commands (COMMIT and ROLLBACK)

14.2 Defining a Database: The DEFINE DATABASE Command

To access DBMS records and sets, you must define the DBMS database in DATATRIEVE terms. Using the DEFINE DATABASE command, you create a DATATRIEVE "database instance" for the DBMS database.

The DEFINE DATABASE command:

- Defines a pointer to the DBMS database and gives the database a unique DATATRIEVE name
- Identifies the DBMS schema, subschema, and root file you want to access and associates it with the DATATRIEVE database name
- Stores the new database definition in the CDD

You must specify the name of a subschema, its schema, and the associated database root file, in that order.

The following example defines a database instance. The CDD path name of the schema is CDD$TOP.DTR$LIB.DEMO.DBMS.PARTS. The name of the subschema is PART, and the root file is DTR$LIBRARY:DTRPARTDB.ROO.

```
DTR> DEFINE DATABASE PARTS_DB
DFN>     USING SUBSCHEMA PART
DFN>     OF SCHEMA CDD$TOP.DTR$LIB.DEMO.DBMS.PARTS
DFN>     ON DTR$LIBRARY:DTRPARTDB.ROO;
DTR>
```

he format for this command is:

```
EFINE DATABASE dbms-database-path      <---------- (1)

    [USING] [SUBSCHEMA] subschema-name    <------- (2)

    [OF] [SCHEMA] schema-path-name        <------- (3)

    ON root-file-spec;                    <---------- (4)
```

1. The **dbms-database-path** is the DATATRIEVE name you use for your database instance.

2. The **subschema-name** is the DBMS name of the definition that defines records and their relationship to each other. The DBMS user creates the subschema definition when the DBMS database is created. It describes just that portion of a DBMS database or schema needed by a particular application.

3. The **schema-path-name** is the DBMS name of the definition that contains all area, record, data item, and interrecord relationship (set) definitions. It describes how relationships are established and discontinued in a database, or how a record becomes a member of a set and is taken out of a set. It also describes set order.

   The DBMS user creates it in a file using the DBMS Data Definition Language (DDL). When the DBMS user compiles this definition using the DDL/COMPILE command, the definition is loaded into the CDD. A subsequent DBO/CREATE command issued by a DBMS user from DCL level creates the database. The DBMS schema can be referenced by more than one database instance.

4. The **root-file-spec** is the name of the database root file. A DBMS root file contains all the information needed by DBMS to access the schema, subschema, and the data files at run time. The DBMS user creates the root file.

1.3 Accessing the Database

ter you define a database, you can access it in one of two ways:

Ready it directly with the READY database-path-name command

Define domains for each record in the database with the DEFINE DOMAIN command and then ready each domain

Using the first method, you can ready all the records associated with that database using a single READY command. The advantage of using this method is that you need not define a domain for each record in the DBMS database. The syntax is simpler and readying a database may be faster than readying the separate domains.

Using the second method, in addition to defining the database, you need to define a domain for each DBMS record. The advantage of using this method is that you can use view domains, and you can associate a form definition with a DBMS record in the domain definition.

The next sections discuss these two methods.

14.3.1 Readying an Entire Database Directly

When you ready a database directly, you can access all the data from:

- All or selected DBMS records associated with that database in the DBMS subschema definition
- The sets in which those records participate

For example, if you ready PARTS_DB, you can access all the records in the PART subschema definition. When you do a SHOW READY command, you see all the records made available by the READY command.

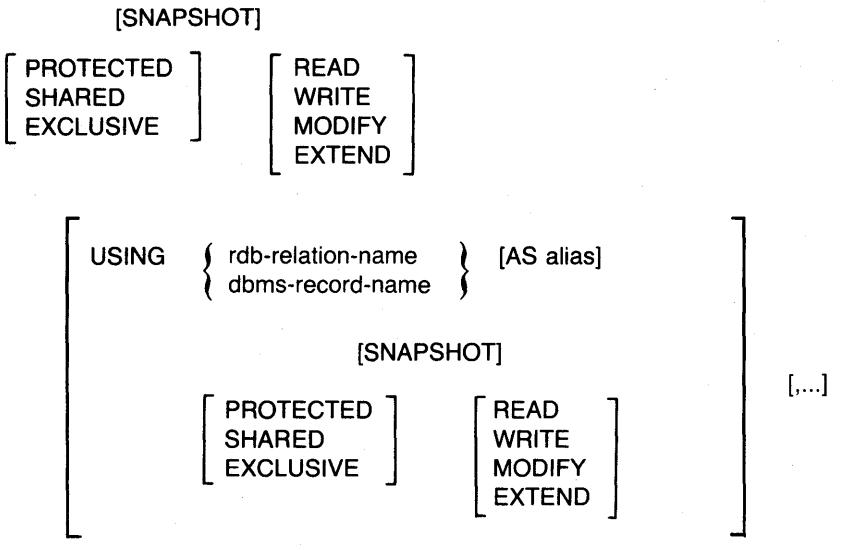
```
DTR> READY PARTS_DB

DTR> SHOW READY
Ready sources:
   CLASS:  Record, DBMS, shared read
           <CDD$TOP.DTR$LIB.DEMO.DBMS.PARTS_DB;1>
   COMPONENT:  Record, DBMS, shared read
           <CDD$TOP.DTR$LIB.DEMO.DBMS.PARTS_DB;1>
   DIVISION:  Record, DBMS, shared read
           <CDD$TOP.DTR$LIB.DEMO.DBMS.PARTS_DB;1>
   EMPLOYEE:  Record, DBMS, shared read
           <CDD$TOP.DTR$LIB.DEMO.DBMS.PARTS_DB;1>
   PART:  Record, DBMS, shared read
           <CDD$TOP.DTR$LIB.DEMO.DBMS.PARTS_DB;1>
   QUOTE:  Record, DBMS, shared read
           <CDD$TOP.DTR$LIB.DEMO.DBMS.PARTS_DB;1>
   SUPPLY:  Record, DBMS, shared read
           <CDD$TOP.DTR$LIB.DEMO.DBMS.PARTS_DB;1>
   VENDOR:  Record, DBMS, shared read
           <CDD$TOP.DTR$LIB.DEMO.DBMS.PARTS_DB;1>
No loaded tables.

DTR>
```

'he format for the READY command for databases is:

EADY database-path-name



1. The **database-path-name** is the DATATRIEVE name you defined for your database instance.

2. The **access options** (PROTECTED, SHARED, and EXCLUSIVE) and the **access modes** (READ, WRITE, MODIFY, and EXTEND) are discussed in Section 14.3.3. SHARED READ is the default access for DBMS databases.

3. The **USING clause** allows you to limit database access to specified DBMS records. If you omit the USING clause of the READY command, all records in the database are readied.

4. The **dbms-record-name** is the name used by the DBMS user to define DBMS records in the subschema definition.

5. The **alias name** is a name you use to refer to the DBMS record specified. If you include an alias, you must use it in all the DATATRIEVE statements and commands that refer to the DBMS record.

ote that you can ready selected DBMS records in the database. You need not ady them all. You can specify access options for each record, as well as options r the entire database.

For example:

```
DTR> READY PARTS_DB USING EMPLOYEE READ, DIVISION WRITE
DTR>
DTR> SHOW READY
Ready sources:
   DIVISION:   Record, DBMS, shared write
           <CDD$TOP.DTR$LIB.DEMO.DBMS.PARTS_DB;1>
   EMPLOYEE:  Record, DBMS, shared read
           <CDD$TOP.DTR$LIB.DEMO.DBMS.PARTS_DB;1>
No loaded tables.

DTR>
```

This READY command provides access to all the data from:

- The records associated with those domains (EMPLOYEE and DIVISION records)
- The sets in which those records participate (MANAGES, CONSISTS_OF, ALL_EMPLOYEES)

14.3.2 Defining and Readying DBMS Domains

You can also access DBMS data by defining a domain for *each DBMS record* you require. DATATRIEVE automatically maps database records to these domains.

In the following example, you:

- Define a DBMS domain (PART_S)
- Identify the associated DBMS record type (PART)
- Identify the DATATRIEVE instance (PARTS_DB) of the DBMS database

```
DTR> DEFINE DOMAIN PART_S USING
CON>   PART
CON>   OF DATABASE PARTS_DB;
DTR>
```

The format for the DEFINE DOMAIN command for DBMS domains is:

```
DEFINE DOMAIN domain-path-name                          <------ (1)

        [USING] record-path-name                        <------ (2)

        [OF] [DATABASE] database-path-name              <------ (3)

        [FORM [IS] form-name [IN] form-library];  <------ (4)
```

1. The **domain-path-name** is the name you assign the DBMS domain you are creating. This DBMS domain is a DATATRIEVE structure that points to the DBMS record type. To avoid confusion, this name should not be the same as the DBMS record name and cannot be the same as the DBMS database name.

2. The **record-path-name** is the name of a record type contained in the subschema of the specified database. You must define a domain and specify a single DBMS record name for each record type in the DBMS database you want to access.

3. The **database-path-name** is the database instance you defined in the DATATRIEVE DEFINE DATABASE command.

4. The **form-name** is the name of a form associated with the DBMS domain. The **form-library** is the file specification of a form library.

or the full syntax of this command, see the *VAX DATATRIEVE Reference Manual.*

he following examples define domains for all the records (in addition to the ART record defined above) in the DBMS PARTS database. The domains are LASSES, COMPONENTS, DIVISIONS, EMPLOYEES, QUOTES, UPPLIES, VENDORS.

ote that you establish a domain definition for each record type you want to ccess in the PARTS_DB database. (Blank lines separate the definitions for arity.)

```
[R> DEFINE DOMAIN CLASSES USING
)N>   CLASS OF DATABASE PARTS_DB;
[R>
[R> DEFINE DOMAIN QUOTES USING
)N>   PR_QUOTE OF DATABASE PARTS_DB;
[R>
[R> DEFINE DOMAIN SUPPLIES USING
)N>   SUPPLY OF DATABASE PARTS_DB;
[R>
[R> DEFINE DOMAIN VENDORS USING
)N>   VENDOR OF DATABASE PARTS_DB;
:R>
:R> DEFINE DOMAIN EMPLOYEES USING
)N>   EMPLOYEE OF DATABASE PARTS_DB;
:R>
:R> DEFINE DOMAIN COMPONENTS USING
)N>   COMPONENT OF DATABASE PARTS_DB;
:R>
:R> DEFINE DOMAIN DIVISIONS USING
)N>   DIVISION OF DATABASE PARTS_DB;
:R>
```

As with the database or DATATRIEVE RMS domains, you must ready DBMS domains before you can access data from those domains. When you use the READY command on a DBMS domain, you get access not only to a record type, but to the sets in which the record type participates.

For example, the following command readies the domains EMPLOYEES and DIVISIONS:

```
DTR> READY EMPLOYEES, DIVISIONS
```

It provides access to all the data from:

- The records associated with those domains (EMPLOYEE and DIVISION records)
- The sets in which those records participate (MANAGES, CONSISTS_OF, ALL_EMPLOYEES)

The format for the READY command is:

READY domain-path-name [AT node-spec] [AS alias-1]

[PROTECTED] [READ] [,...]
[SHARED] [WRITE]
[EXCLUSIVE] [MODIFY]
[EXTEND]

[SNAPSHOT]

Note

You must ready all participants in a set (both owner and member domains) to access data identified by that set. The descripton of all of the arguments to the READY command is in the *VAX DATATRIEVE Reference Manual.*

14.3.3 Results of the READY Command

When you ready a DBMS database, DBMS record, or a DBMS domain, all the realms in which a DBMS record participates are automatically readied. A **realm** i one or more schema areas and is defined in a subschema. A realm lets you restric or grant access to sections of a database.

Ask your system administrator for a listing of the realms for the subschema you are using. The realms are ultimately associated with **storage areas**, DBMS units contained in single files. It is actually these files that are opened, through a READY command that readies at least one of the domains in that file.

You can ready a database or a domain as SHARED, PROTECTED, or EXCLUSIVE. The READY options you choose determine the level of VAX DBMS locking. Locking affects both you and other active users. You can also specify how you access the domains or records for READ, WRITE, MODIFY, or EXTEND access.

See the *VAX DATATRIEVE Reference Manual* and the *VAX DBMS Database Design Guide* for information on access options and modes.

When used with DBMS domains, DBMS records or the entire DBMS database, the READY command has these effects:

Each specified domain or DBMS record is readied with the requested access.

The default access mode is SHARED READ. If you do not specify EXCLUSIVE or PROTECTED access, DATATRIEVE always readies for SHARED access.

When you ready more than a single domain or DBMS record in the realm:

- If you enter a SHOW READY command, you see the DBMS records and domains with the access option and mode you specified in the READY command. For example:

```
DTR> READY PARTS_DB USING SUPPLY, VENDOR EXCLUSIVE WRITE

DTR> SHOW READY
Ready sources:
   SUPPLY:  Record, DBMS, shared read
            <CDD$TOP.DTR$LIB.DEMO.DBMS.PARTS_DB;1>
   VENDOR:  Record, DBMS, exclusive write
            <CDD$TOP.DTR$LIB.DEMO.DBMS.PARTS_DB;1>
```

- However, other users' access to those records or domains is limited by the most restrictive access you specify in a READY command. This restrictive access applies until you ready the DBMS domain or record again, or until you do a final FINISH on the database. (The DATATRIEVE FINISH command ends access to DBMS domains and records and executes a DBMS COMMIT. See Section 14.10.4 for more information.) Thus, DATATRIEVE applies the access mode and option of the most restrictive domain or DBMS record in the realm to all domains in that realm.

  In the preceding example, therefore, as long as VENDOR is readied with EXCLUSIVE WRITE, the access applied to SUPPLY is also EXCLUSIVE WRITE.

- To change access to a domain or a DBMS record, you must ready the domain or DBMS record again. Because DATATRIEVE will ready other domains or DBMS records in the realm again if the new access is more restrictive, changing access to one domain or record may actually result in the entire realm being readied again.
- If you print, modify, or store data into a domain, record, or database, DATATRIEVE allows you to reready with a more restrictive access only after you do a COMMIT, a FINISH on the record or domain, or a ROLLBACK. DATATRIEVE displays a message indicating it is releasing the collections automatically to allow such a reready.

EXCLUSIVE WRITE access lets you store and modify records but prevents other users from even retrieving records from the domain until you end your access to it or ready it again with a different access mode.

You can use the SHOW command to see:

- The database records or domains that are readied (SHOW READY)
- The fields in the records that are readied (SHOW FIELDS)
- The sets that are made accessible, plus the access mode and access option with which you readied the database or domain (SHOW SETS)

14.3.3.1 The SHOW FIELDS Command -- The following SHOW FIELDS command displays the fields of the record types EMPLOYEE and DIVISION:

```
DTR> SET DICTIONARY CDD$TOP.DTR$LIB.DEMO.DBMS
DTR> READY EMPLOYEES, DIVISIONS
DTR> SHOW FIELDS

DIVISIONS
   DIVISION
      DIV_NAME  <Character string>
EMPLOYEES
   EMPLOYEE
      EMP_ID (ID)        <Number>
      EMP_LAST_NAME (LAST_NAME) <Character string>
      EMP_FIRST_NAME (FIRST_NAME)       <Character string>
      EMP_PHONE (PHONE_NUMBER)  <Number>
      EMP_LOC (LOCATION)        <Character string>
No global variables are declared.
```

14.3.3.2 The SHOW SETS Command -- Using the SHOW SETS command, you can see the sets in which the DBMS domains and record types participate. The SHOW SETS command identifies the domains that participate in the sets, i you defined domains. Otherwise, it shows the DBMS records that participate in each set.

n DATATRIEVE, because you can define a domain for each DBMS record, a HOW SETS command indicating the domains implicitly shows the DBMS ecords that participate in the sets.)

or example, two domains, EMPLOYEES and DIVISIONS (and their associated ecords, EMPLOYEE and DIVISION), participate in the set MANAGES. When ou have defined domains, if you enter a SHOW SETS command, you see the omain names you defined. The EMPLOYEES domain represents the MPLOYEE record and the DIVISIONS domain represents the DIVISION ecord.

```
TR> SHOW SETS
   Set: MANAGES
        Owner: EMPLOYEES
        Member: DIVISIONS, automatic optional

   Set: CONSISTS_OF
        Owner: DIVISIONS
        Member: EMPLOYEES, manual optional

   Set: ALL_EMPLOYEES
        Member: EMPLOYEES, automatic fixed
```

he terms "member" and "owner" refer to set characteristics that are described the section on finding and printing DBMS records.

he terms "automatic", "manual", "optional", and "fixed" are discussed in the ction on erasing and disconnecting records and sets.

ne next section on finding data in a DBMS database discusses sets further.

4.4 Forming a DATATRIEVE Query

nce you have defined and readied your DBMS database, you may want to:

- Find and display a record or records from that database
- Find data related to that record through DBMS set relationships

ATATRIEVE provides you with an English-like query language for DBMS tabases.

his section discusses how to form a simple DATATRIEVE query; it is for DBMS ers unfamiliar with DATATRIEVE. The next section discusses how to add the ATATRIEVE clauses (WITHIN, MEMBER, and OWNER) to find records lated by a set.

In DATATRIEVE, you can access a single record at a time or you can work with a group of records. Using the DATATRIEVE record selection expression (RSE), you can query, print, modify, or store data in a DBMS database. A DATATRIEVE RSE identifies and limits the records you want to include in a record stream, as shown in Figure 14-3.

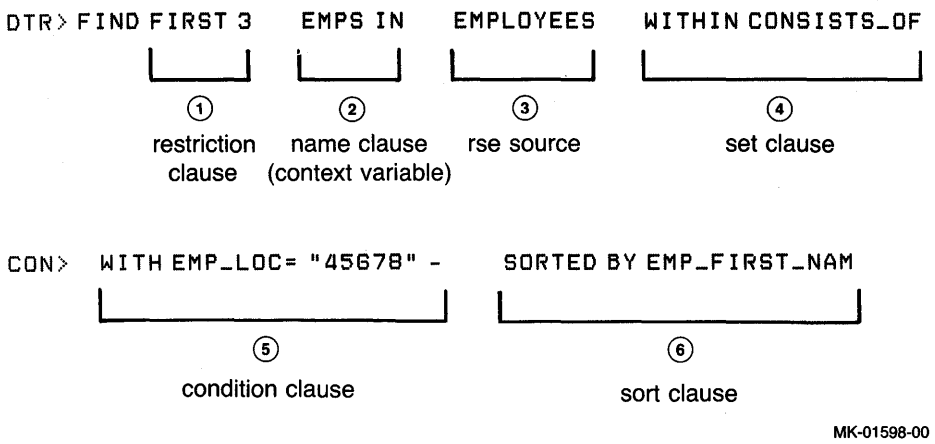


Figure 14-3: The Parts of an RSE

An RSE consists of the following components:

1. An optional restriction clause (ALL or a number) -- to tell DATATRIEVE how many records to include in a record stream.
2. An optional name clause -- to qualify or provide a name for the record source. You can use this name later to identify the record stream and to access records in that stream. The clause is also called a context variable when you use it to name a record stream or modify field names in a FOR loop.
3. A required record source -- to identify an RMS, Rdb, or DBMS domain name, a view or network domain, a DBMS record name, an Rdb relation name, a collection, or a list.
4. A DBMS set clause (WITHIN, OWNER, or MEMBER) -- to identify a set name and the relationship to that set.
5. An optional selection condition (WITH Boolean expression) -- to tell DATATRIEVE what values to look for in a record.
6. An optional sort clause (SORTED BY) -- to sort the record stream in the order you specify.

ıe format for the RSE follows. Note that it includes the components just identi-
ɜd as well as the following two additional elements:

An optional relational clause (CROSS) -- to combine data from more than one domain

An optional reduction clause (REDUCED TO) -- to retain only unique values

```
IRST n ]    [context-var IN] rse-source
.LL    ]

;ROSS [context-var IN] rse-source [OVER field-name]] [...]

/ITH boolean-expression] [REDUCED TO reduce-key [,...]]

ORTED BY sort-key [,...]]
```

he format for rse-source is:

```
{ domain-name      }
{ collection-name  }
{ list             }
{ rdb-relation-name}
{ dbms-record-name }  [ { MEMBER } [OF] [context-name.set-name] ]
                      [ { OWNER  }                              ]
                      [ { WITHIN }                              ]
```

ote that the domain name in the preceding syntax includes DBMS domains as
ell as Rdb, RMS, view, and remote domains. Note that the MEMBER,
WNER, and WITHIN set-name syntax is used only with a DBMS domain name
DBMS record name.

ɔr a more complete explanation of RSE syntax, see Chapter 5 of the *VAX ATATRIEVE Reference Manual* and the chapter on writing record selection ɨpressions in this manual. For an example of the CROSS clause, see the section this chapter on using the CROSS syntax.

1.5 Forming a DATATRIEVE/DBMS Query

/pically, you can use two methods to form a DATATRIEVE query that accesses BMS data:

The FIND and SELECT statements, to establish a collection of records and point to a specific record from that collection

The FOR statement, which uses an RSE to form a temporary record stream

These statements are discussed and illustrated in the next sections.

14.5.1 Forming a DATATRIEVE Collection of DBMS Records

You can access DBMS data in DATATRIEVE by forming a **collection**. A collection is a group of records that you can access until you:

- Form a new collection (unless you assign a name to the new or old collection).
- Remove the collection with the RELEASE command.
- Finish the domain that owns the collection using the FINISH command.

14.5.1.1 Using the FIND Statement -- You form a collection using the DATATRIEVE FIND statement. You can create collections from a readied DBMS domain or from a DBMS record in a database you have readied through DATATRIEVE.

When you form a collection, DATATRIEVE gives that collection the name CURRENT. You can access this collection with the name CURRENT, or you can assign an additional name for a collection and use this name to access the collection. You use the name clause as illustrated in the RSE format to name a collection.

If you name a collection, it is not deleted when you form another collection. Therefore, you can have several named collections of records available at any tim in DATATRIEVE.

The following example, for instance, forms a collection from the DBMS record EMPLOYEE, readied when you readied the PARTS_DB database:

1. Form a collection of the first five employees and assign the name EMP to that collection.
2. Form a second collection by limiting the EMP collection to those employee with EMP_LOC = 45678 and assign the name EMP45.
3. Show that DATATRIEVE retains information about both collections EMP45 (also CURRENT) and EMP.
4. Print the CURRENT collection.
5. Print the EMP collection.

```
'R> READY EMPLOYEE
'R> FIND FIRST 5 EMP IN EMPLOYEE        <----------  (1)
 records found]
'R> FIND EMP45 IN EMP WITH EMP_LOC = "45678" <-----  (2)
 records found]
'R> SHOW COLLECTIONS                    <----------  (3)
llections:
     EMP45       (CURRENT)
     EMP

'R> PRINT CURRENT                       <----------  (4)

                                          Phone
ent  Last Name---------- First Name  Number    Loc

 998 HILL                 OLA          124567 45678
2234 HORTI                BRUCE        124567 45678

'R> PRINT EMP                           <----------  (5)

                                          Phone
ent  Last Name---------- First Name  Number    Loc

5624 FRASER               BOB        8902345 23456
2333 HOFFMAN              MIKE       4568901 89012
 998 HILL                 OLA         124567 45678
2234 HORTI                BRUCE       124567 45678
7777 PASCAL               RICHARD    4568901 89012
```

ote that you can still access the records in the EMP collection after you create a cond collection with the FIND statement. As long as you name the collections, ATATRIEVE retains them when you create new collections.

you name a collection, you can use this name as a context variable in an RSE to odify a record stream. Later sections contain examples showing how you can do is.

.5.1.2 Using the SELECT Statement -- After you form a collection, you use e SELECT statement to choose a record from that collection. This selected cord is the target of other DATATRIEVE statements such as PRINT, ODIFY, and ERASE. The SELECT statement establishes DBMS currency for record.

ou can specify the record you want to access either by using an integer ELECT 5 gets the fifth record of the collection) or by such syntax as FIRST, EXT, LAST, and so on.

In the following example, the SELECT statement:

1. Selects the first record in the EMP45 collection, which is also the CURRENT collection. When you do not specify a collection name, the SELECT statement defaults to the first record in the CURRENT collection.
2. Selects the second record in the EMP collection.

```
DTR> SELECT                    <-------------------------  (1)
DTR> PRINT

                                                  Phone
Ident  Last Name----------- First Name  Number    Loc

   998 HILL                  OLA          124567 45678


DTR> SELECT 2 EMP              <-------------------------  (2)
DTR> PRINT

                                                  Phone
Ident  Last Name----------- First Name  Number    Loc

 12333 HOFFMAN               MIKE        4568901 89012
```

For a complete discussion of the FIND and SELECT statements, see the FIND and SELECT statements in the *VAX DATATRIEVE Reference Manual.*

14.5.2 Forming a Record Stream of DBMS Records

The FOR statement differs from using collections in that it creates a temporary record stream that DATATRIEVE knows about only while it is executing that statement. It places locks on fewer DBMS records than a FIND statement.

DATATRIEVE can access each record in that stream one by one, displaying, printing, or modifying each record according to your specifications. As each record is accessed, it becomes current.

For instance, in the following example, you:

1. Form a temporary DATATRIEVE record stream of employee records with EMP_LOC = 45678
2. Display only those employees in the record stream
3. Try to print the CURRENT collection, which does not exist

```
TR> READY EMPLOYEES
TR> FOR EMPLOYEES WITH EMP_LOC = "45678"    <---------- (1)
Looking for statement]
ON> PRINT EMPLOYEE                          <---------- (2)

                                        Phone
dent  Last Name---------- First Name  Number   Loc

  998 HILL                OLA         124567 45678
22234 HORTI               BRUCE       124567 45678
11141 SCHATZEL            BETH        124567 45678
12322 THOMPSON            STEPHEN     124567 45678
12345 HUNTER              BUTCH       124567 45678
          .
          .
          .


TR> SHOW CURRENT                            <-------- (3)
 current collection has not been established.
```

4.6 Forming a DATATRIEVE/DBMS Query of Data Related by Sets

ı the previous section, you used simple DATATRIEVE queries to access infor-
ıation from a single DBMS record or domain.

lote that DATATRIEVE also makes set information available when you ready a
omain. In the following sections, you access information from both the data in a
ngle record and data in records related through information in a set.

s with DBMS data in a single record, you can use the FIND and SELECT syn-
ıx to form collections, or you can use the FOR syntax to create a temporary
ecord stream.

ote two important concepts about using DATATRIEVE statements to access
BMS data related by set information:

You must use either the FIND and SELECT or the FOR statement to establish the single record context, called currency, in DBMS. DBMS needs this single record context to find related records and sets.

You must specify a DBMS set name to identify the sets in which a record participates, unless you use the Context Searcher. (See Section 14.6.4.)

When you access information through DBMS sets, you access:

- First, a particular record from a domain or record (for example, a department from the DIVISIONS domain)
- Second, data related to that record from other DBMS domains or records through a set (for example, employees from the EMPLOYEES domain related through the set CONSISTS_OF)

The examples in this section use the set CONSISTS_OF. It represents the relationship between a department in an organization (DIVISIONS domain) and the employees that make up that department (EMPLOYEES domain). Figure 14-1 shows this relationship.

The following sections illustrate DATATRIEVE syntax you use to access information in sets.

14.6.1 Forming Collections of DBMS Set Data

As with data in a single DBMS record, you can form a collection of DBMS data related through set information. You can then access those records by the collection name.

In the following example, for instance, you form a collection (DIV) from the DIVISIONS domain. You form a second collection (EMP) of employee records. The employee records in EMP collection are related to the selected record from the DIV collection. DATATRIEVE now knows about both collections, DIV and EMP. You can print records from both of these collections:

1. Use the FIND statement to create a collection of records from the DIVISIONS domain. The SELECT statement identifies a single record occurrence and establishes context (currency in DBMS) with the set information.
2. Use the FIND statement again to establish a collection of employee information from the EMPLOYEES domain. By using the WITHIN set-name syntax, you identify the set that you want DATATRIEVE to use. This set identifies the employee records related to the SOFTWARE division.

```
TR> SET DICTIONARY CDD$TOP.DTR$LIB.DEMO.DBMS
TR> READY DIVISIONS, EMPLOYEES

TR> FIND DIV IN DIVISIONS WITH DIV_NAME = "SOFTWARE"
1 record found]
TR> SELECT

TR> FIND EMP IN EMPLOYEES WITHIN CONSISTS_OF
3 records found]
TR> PRINT DIV_NAME, EMPLOYEE OF EMPLOYEES

                                                  Phone
ivision Name---Ident  Last Name----First Name  Number   Loc
OFTWARE         23451 HUTCHINGS      BRUCE      2346789 67890
OFTWARE         43215 IACOBONE       ANTHONY     124567 45678
OFTWARE         77777 PASCAL         RICHARD    4568901 89012
```

1

2

lote that:

1. You must identify a single record occurrence using FIND and SELECT so that DATATRIEVE can establish context (currency in DBMS) for the related set information.

   In the first part of the example, DATATRIEVE uses the selected single record, SOFTWARE, to establish context for the set information.

2. Specify the set that contains pointers relating the data from one domain (or DBMS record) to another.

   In the second part of the example, the WITHIN clause tells DATATRIEVE to look at the single occurrence of the set CONSISTS_OF. The single set occurrence contains pointers that point from the single record occurrence DATATRIEVE currently knows about, the SOFTWARE division, to related records in the domain, EMPLOYEES.

4.6.2 Forming Record Streams of DBMS Set Data

ou can use the FOR loop to access the information from two domains. The ested FOR loop in the following example creates two record streams and allows ou to access records from each stream. In this example:

1. You form a temporary record stream of all records from the DIVISIONS domain with a DIV_NAME that contains the string VT (groups that develop video terminals). You assign that record stream a name (VID).

2. You form a second temporary record stream of the records from the related employee records in the EMPLOYEES domain identified by the set CONSISTS_OF

3. You print the division name and the related employee information.

```
DTR> FOR VID IN DIVISIONS WITH DIV_NAME CONT "VT"  <------- (1)
CON>    FOR EMP IN EMPLOYEES WITHIN CONSISTS_OF    <------- (2)
CON>    PRINT VID.DIV_NAME, EMP.EMPLOYEE           <------- (3)

                                                          Phone
Division Name-------Ident  Last Name------ First Name  Number   Loc

VT100 DEVELOPMENT    65437 FRANK           BEBI        4568901 89012
VT100 DEVELOPMENT    12333 HOFFMAN         MIKE        4568901 89012
VT100 DEVELOPMENT    54332 IGLESIAS        RAFAEL      2346789 67890
VT52  DEVELOPMENT     9867 FLETCHER        BRUCE        124567 45678
VT52  DEVELOPMENT    43221 HYNES           RICH        8902345 23456
```

Notice that you can name the record stream in the FOR statement (VID) and use that name to qualify a field name (VID.DIV_NAME). In this example, the qualifying name is not necessary to identify fields uniquely.

FOR loops allow you to access records more quickly than FIND statements. Note that when you use a FOR loop, you need not use the SELECT syntax; DATATRIEVE selects a single record each time through the loop. For more information on the FOR loop, see the FOR statement in the *VAX DATATRIEVE Reference Manual.*

14.6.3 Using OWNER and MEMBER Clauses to Identify Sets

In the previous section, you used the WITHIN clause to identify:

- The domain name (EMPLOYEES) from which you wanted the employee data
- The set name (CONSISTS_OF) that identified the employee record you wanted

The WITHIN clause allows you to specify a set name without having to know if the domain is a member or owner of the set.

There are two additional clauses of the record selection expression that allow you to specify access to records through DBMS sets:

- The MEMBER clause
- The OWNER clause

The OWNER and MEMBER clauses specify whether a record is a member or an owner of a set. In many cases, you can use the WITHIN clause in place of the MEMBER and OWNER clauses. The SHOW SETS command lets you see

vhether a domain is a member or an owner of a set:

```
'TR> SHOW SETS
   Set: MANAGES
        Owner: EMPLOYEES                          <--------- (1)
        Member: DIVISIONS, automatic optional

   Set: CONSISTS_OF
        Owner: DIVISIONS                          <--------- (2)
        Member: EMPLOYEES, manual optional

   Set: ALL_EMPLOYEES                             <--------- (3)
        Member: EMPLOYEES, automatic fixed
```

'he SHOW SETS command indicates that:

1. The EMPLOYEES domain (in DBMS, the EMPLOYEE record) owns the set MANAGES. The DIVISIONS domain (in DBMS, the DIVISION record) is a member of the set MANAGES.
2. The DIVISIONS domain (in DBMS, the DIVISION record) owns the set CONSISTS_OF. The EMPLOYEES domain (in DBMS, the EMPLOYEE record) is a member of the set CONSISTS_OF.
3. The EMPLOYEES domain is also a member of the ALL_EMPLOYEES set, a system-owned set.

Note

A system-owned set is a set owned by DBMS instead of a user-defined record. System-owned sets have only one occurrence in the database and are used for relationships with a large number of member occurrences, or as entry points into a database. For more information on system owned sets, see the *VAX DBMS Database Design Guide.*

igure 14-4 shows all these set relationships.

he following sections show you how to use the MEMBER and OWNER clauses) access DBMS records.

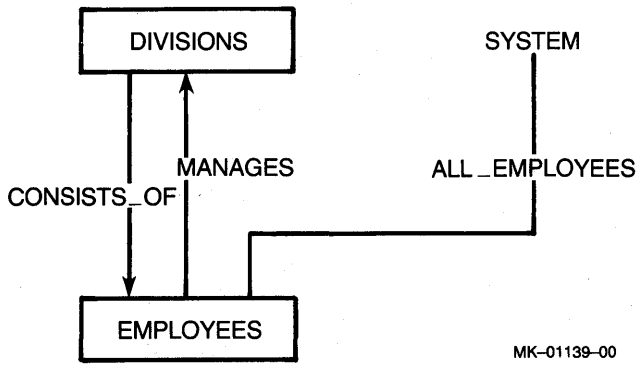


Figure 14-4: Set Relationships in Sample DBMS Database

14.6.3.1 The MEMBER Clause -- The MEMBER clause lets you access the member records of a set. Conceptually, you are telling DATATRIEVE to "look down" from a specified position (determined by a FIND/SELECT or a FOR statement) and find the members of the set that are linked to the selected record.

For example, because EMPLOYEES is a member of the set CONSISTS_OF, you can use the MEMBER syntax rather than the WITHIN syntax you used in the previous section:

```
DTR> FIND DIV IN DIVISIONS WITH DIV_NAME = "SOFTWARE"
DTR> SELECT
DTR> FIND EMPLOYEES MEMBER OF CONSISTS_OF
DTR> PRINT ALL DIV_NAME, EMPLOYEE

                                                          Phone
Division Name------- Ident  Last Name---------- First Name  Number   Lo

SOFTWARE              23451 HUTCHINGS           BRUCE       2346789 678
SOFTWARE              43215 IACOBONE            ANTHONY      124567 456
SOFTWARE              77777 PASCAL              RICHARD     4568901 890
```

14.6.3.2 The OWNER Clause -- The OWNER clause tells DATATRIEVE to "look up" from a position in the database, thereby giving you access to the owner record of a set. The OWNER clause operates like the MEMBER clause; the only difference is in the direction that DATATRIEVE looks.

For example, you know an employee named Richard Pascal and want to know the division in which he works. Because DIVISIONS is the owner of the set

)ONSISTS_OF, you:

1. Form a record stream with the single employee record
2. Use the OWNER clause with the CONSISTS_OF set to find the department in which the employee works

```
TR> FOR EMP IN EMPLOYEES WITH EMP_LAST_NAME= "PASCAL"      <------- (1)
ON>  PRINT ALL EMPLOYEE, DIV_NAME OF
ON>  DIVISIONS OWNER OF CONSISTS_OF                        <------- (2)
TR>

                                          Phone
dent  Last Name---------- First Name  Number   Loc  Division Name---

7777 PASCAL               RICHARD     4568901 89012   SOFTWARE

TR>
```

4.6.4 Using the SET SEARCH Command to Access Sets

ıs a DATATRIEVE user, you can use the SET SEARCH command to establish ontext for list fields in records or for fields in hierarchical views.

ı addition, as a DATATRIEVE user accessing a DBMS database, you can use ıe SET SEARCH command to search for records related by set information. 'ou use this command in place of the WITHIN, OWNER, or MEMBER clauses.

he SET SEARCH command instructs the DATATRIEVE Context Searcher to hoose the shortest route between DBMS record types when executing a PRINT :atement. DATATRIEVE resolves the context for you so that you need not ɔecify the set relationship.

he following example executes a SET SEARCH statement and prints the ɛlated division data without requiring you to specify the set CONSISTS_OF:

```
[R> SET SEARCH
[R> FIND FIRST  1 EMPLOYEES
[ record found]
[R> SELECT
[R> PRINT DIV_NAME
ɔt enough context.  Some field names resolved by Context Searcher.

.vision Name-------

IG STOCKROOM

:R>
```

The following example walks through all occurrences of the set type CLASS_PART, instructing DATATRIEVE to display on your terminal only the class code number, part identification numbers, and part descriptions of records owned by CLASSES:

```
DTR> SET SEARCH
DTR> PRINT CLASS_CODE, PART_ID, PART_DESC OF CLASSES
Not enough context. Some field names resolved by Context Searcher.

        Part
Code    Number    ----------------Part Description----------------

 BR    BR-1234-56 LA34
       BR-3467-91 LA120
       BR-8901-23 LA36
 BT    BT-0456-78 VT52
       BT-1634-56 VT100
 BU    BU-0345-67 TERMINAL TABLE VT52
       BU-1045-68 FREE-STANDING FRAME ASSEMBLY
             .
             .
             .
       CG-3256-40 VT100 KEYBOARD KED ASSY
       CG-3454-38 PLASTIC KEY NUM. STYLE C
       CG-4567-89 PLASTIC KEY ALPHA. STYLE A
       CG-8767-78 VT100 SCREEN
       CG-8901-23 VT100 HOUSING
       CG-9435-61 KEY BASES
       CG-9562-13 VT100 NUMERIC KEY CAP SET
```

In this case, the Context Searcher found many records. You can find the owners and members of database sets by using the PRINT and SET SEARCH statements. It is important to remember that SET SEARCH guesses: it always takes the shortest route in a set structure and, therefore, might not return the right answer.

Note that if you did not use the Context Searcher in the preceding example, the full DATATREIVE query would use an inner print list. For the statement "PRINT CLASS_CODE, PART_ID, PART_DESC OF CLASSES", you would need the following query:

```
DTR> PRINT CLASS_CODE, ALL PART_ID, PART_DESC OF
CON> PART_S MEMBER OF CLASS_PART OF CLASSES
DTR>
```

The expanded statement includes an inner print list.

The following example readies two more domains and instructs DATATRIEVE t display the name and description of the parts supplied by the vendor with the

ıame "QUALITY COMPS":

```
)TR> SET SEARCH
)TR> READY SUPPLIES, VENDORS
)TR> PRINT VEND_NAME, PART_DESC OF
:ON>  VENDORS WITH VEND_NAME = "QUALITY COMPS"
lot enough context. Some field names resolved by Context Searcher.

------------Vendor Name---------------

UALITY COMPS
T100 KEYBOARD ASSY
UMERIC KEYPAD FRAME
T52 HOUSING
```

'his PRINT statement resulted in the display of all parts associated with the pecified vendor. The PRINT statement is equivalent to:

```
TR> PRINT VEND_NAME, ALL ALL PART_DESC OF PART_S OWNER OF
ON> PART_INFO OF SUPPLIES MEMBER OF VENDOR_SUPPLY OF VENDORS WITH
ON> VEND_NAME = "QUALITY COMPS"
```

4.7 Finding Data from Two or More Domains

n previous sections, you found records from several domains by first finding a ingle record in one domain and then related data in a second domain through a et relationship.

)ATATRIEVE provides several ways for you to access records from two or more omains, in addition to using the simple DATATRIEVE queries shown in the pre- ious sections. These methods become particularly important when the data you ant may reside in more than two domains. These methods include:

- Combining the MEMBER and OWNER clauses to "walk the DBMS sets"
- Using the CROSS clause of the RSE to join the data from several records or domains
- Defining a domain called a VIEW domain that lets you form simple queries and keeps the complex set relationships in the domain definition

he example used in the following sections uses the VENDORS, PART_S, and UPPLIES domains. The data in the VENDORS and PART_S domains has what BMS calls a many-to-many relationship.

ɔr example:

- A single vendor might supply many different parts.
- A single part might be supplied by many different vendors.

In a DBMS database, there cannot be a direct relationship between records (DATATRIEVE domains) that have a many-to-many relationship. If you attempt to select a particular vendor from the VENDORS domain, for instance, and then try to display an associated part from the PART_S domain, DATATRIEVE gives you an error message indicating you have not established the correct context for parts:

```
DTR> READY VENDORS, SUPPLIES, PART_S
DTR> FIND VENDOR WITH VEND_NAME = "QUALTIY COMPS"
[1 record found]
DTR> SELECT
DTR> FIND PART_S WITHIN VENDOR_SUPPLY
Set "VENDOR_SUPPLY" is undefined or used out of context.
DTR>
```

To relate the parts and vendor data, you must go through a third domain or record that is owned by both the PARTS and VENDORS domains.

Figure 14-5 shows the VAX DBMS representation of this many-to-many relationship.

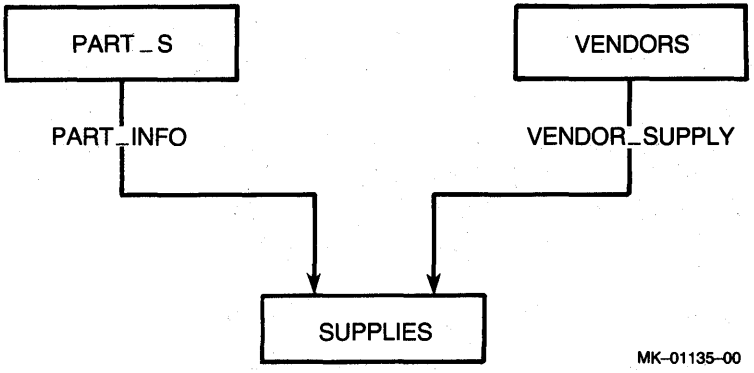


Figure 14-5: DBMS Set Relating Three Domains

14.7.1 Walking the Sets

Suppose you want information that involves access to PARTS, VENDORS, and SUPPLIES. You want:

- The names for a specific vendor (VENDORS domain)
- The types of parts that vendor supplies (PARTS domain)
- Delivery lag time for that part (SUPPLIES domain)

do this, you:

Ready all three DBMS domains. These READY commands also ready the set relationships among the domains.

Find a vendor (for example, the company called QUALITY COMPS).

Find the parts supplied by QUALITY COMPS. Since you cannot directly access part information in the PART_S domain from the VENDORS domain, you must go through the SUPPLIES domain. To do this you:

a. Find all the member records in SUPPLIES related to the selected VENDOR record.

b. Form a FOR loop that establishes a single record context for each record in the SUPPLIES domain. Note that when you use a FOR loop, you do not use the SELECT statement.

c. Print the three parts associated with that SUPPLIES record by using the OWNER clause to find related parts through the PART_INFO set.

tice that you must continue to provide a single record context for
ATATRIEVE with either the FIND and SELECT statements or the FOR loop.

```
:> SET DICTIONARY CDD$TOP.DTR$LIB.DEMO.DBMS
:> READY VENDORS, PART_S, SUPPLIES                    <------- (1)
:>
:> FIND VENDORS WITH VEND-NAME = "QUALITY COMPS"      <------- (2)
record found]
:> SELECT
:> PRINT VEND_ID, VEND_NAME

ndor
ID    --------------Vendor Name---------------

89012  QUALITY COMPS

:>
> FIND SUP IN SUPPLIES MEMBER OF VENDOR_SUPPLY    <---- (3a)
records found]
> PRINT SUP
record selected, printing whole collection.

g Type  Lag Time

  MEMO 4-6 WEEKS
  REPR 1-2 MONTHS
  WSUP 7-8 WEEKS
```

(continued on next page)

```
DTR> FOR SUP                                                    <---- (3b)
CON>  PRINT PART_ID, PART_DESC OF                               <---- (3c)
CON>        PART_S OWNER OF PART_INFO

   Part
  Number    -----------------Part Description-----------------

CG-3161-34 VT100 KEYBOARD ASSY
CG-1052-00 NUMERIC KEYPAD FRAME
CG-0956-78 VT52 HOUSING
DTR>
```

14.7.2 Using the CROSS Clause

You can combine records from several DBMS domains (or DBMS records readie by the READY command for databases) with the CROSS clause from the record selection expression. The CROSS clause lets you compare and combine records from two or more sources into a single record stream. It forms temporary relationships between records stored in different domains (or DBMS records) and let you treat the data as though it were derived from one domain or record. You can also combine data from DBMS records with Rdb or RMS records by using a CROSS clause.

In the following example, you use the three domains from the previous section. Using the CROSS clause, you combine several domains in one collection and write queries against that collection. In this example, you:

1. Form a named collection (QUAL_PART) of the vendor record with a vendo name of QUALITY COMPS.
2. Use the CROSS clause to join this vendor collection with the two other domains. You can do this with a FIND or FOR statement that uses two cross clauses.
3. Display the vendor and all the parts made by that vendor from the VENDORS and PART_S domains.

```
DTR> FIND QUAL_PART IN VENDORS WITH VEND_NAME = "QUALTIY COMPS"  (1)
[1 record found]
DTR> FIND QUAL_PART CROSS                                  <---------- (2)
CON> SUPPLIES MEMBER VENDOR_SUPPLY  CROSS
CON> PART_S OWNER PART_INFO
[3 records found]
DTR> PRINT ALL VEND_ID, PART_ID, PART_DESC,SUP_TYPE  <----------(3)

 Vendor      Part
   ID       Number  -----------------Part Description---------Type

55789012 CG-3161-34 VT100 KEYBOARD ASSY                        MEMO
55789012 CG-1052-00 NUMERIC KEYPAD FRAME                       REPR
55789012 CG-0956-78 VT52 HOUSING                               WSUP
```

4.7.3 Using View Domains

f you define domains for each DBMS record, you can use DATATRIEVE view omains to access data in those records. You cannot form views of DBMS records eadied by the READY command for databases.

'ou can define a simple DATATRIEVE view domain to see a subset of fields from single domain. The following view, for example, lets you define a view domain hat accesses only three fields in the PART_S domain. This view is further efined by the Boolean expression that limits the records to those with 'ART_SUPPORT = "FS":

```
TR> DEFINE DOMAIN VIEW_PARTS_PUBLIC of PART_S USING
FN> 01 PARTV OCCURS FOR PART_S WITH PART_SUPPORT = "FS".
FN>      03 PART_ID FROM PART_S.
FN>      03 PART_DESC FROM PART_S.
FN>      03 PART_PRICE FROM PART_S.
FN> ;
TR>
```

4.7.3.1 Hierarchical Views -- You can also use view domains to combine ecords from several DBMS domains. A view domain that describes data from ıore than a single domain and does not use a CROSS clause is called a hierarchi- ıl view.

hierarchical view domain, unlike a CROSS clause (which also combines data om two or more domains), lets you define the relationship of records from sev- ·al domains and store it in the CDD.

nce you define this relationship, you can display and modify data without having) consider set relationships. You can read and modify selected records from two · more domains as if the data were all in one domain.

ote that because data is not stored in a view, you cannot use a STORE state- ent with a view domain as your record source.

he following view combines the fields DIV_NAME from the domain DIVISIONS ıd EMP_ID and EMP_LAST_NAME from EMPLOYEES. Note that once you :fine the relationship, you can ready the domains and print records from those)mains.

```
'R> SHOW DIV_VIEW
MAIN DIV_VIEW OF DIVISIONS, EMPLOYEES USING
  GRP OCCURS FOR DIVISIONS.
02 DIV_NAME FROM DIVISIONS.
02 WORKERS OCCURS FOR EMPLOYEES WITHIN CONSISTS_OF.
    04 EMP_ID FROM EMPLOYEES.
    04 EMP_LAST_NAME FROM EMPLOYEES.
```

(continued on next page)

```
DTR> READY DIV_VIEW
DTR> PRINT FIRST 5 DIV_VIEW

Division Name------- Ident  Last Name-----------

LA34 DEVELOPMENT      65438 FRATUS
SOFTWARE              23451 HUTCHINGS
                      43215 IACOBONE
                      77777 PASCAL
RMO5 DEVELOPMENT      99998 PAYNE
ENG BUILD & TEST      75624 FRASER
                      55675 HORYMSKI
                          0 HUMPHRY
                       9789 MASE
                      66666 PARVIAINEN
VT100 DEVELOPMENT     65437 FRANK
                      12333 HOFFMAN
                      54332 IGLESIAS
```

Note that when you define a view domain with two or more domains, the data is displayed in hierarchical form, unless you use a CROSS clause in the view definition.

In the previous example, the view of two domains displays one occurrence of the field DIV_NAME and a variable number of employees. In DATATRIEVE, to access individual fields in this hierarchical structure, you must use specialized DATATRIEVE syntax for retrieving values from list fields. See Chapter 6 for a complete explanation of this syntax.

14.7.3.2 Flat Views -- When you combine a view domain with the relational CROSS clause, it flattens the hierarchical relationships. The following flat view combines fields from the domains VENDORS, SUPPLIES, and PART_S:

```
DTR> DEFINE DOMAIN FLAT_PART_VIEW OF PART_S, VENDORS, SUPPLIES USING
DFN> 01 A OCCURS FOR PART_S CROSS  SUPPLIES MEMBER OF PART_INFO CROSS
DFN>    VENDORS OWNER OF VENDOR_SUPPLY.
DFN>         02 PART_ID FROM PART_S.
DFN>         02 SUP_TYPE FROM SUPPLIES.
DFN>         02 VEND_NAME FROM VENDORS.
DFN>;
DTR>
```

The biggest advantage of defining a flat view is that you can refer to each of the fields more easily than in a hierarchical view. That is, you need not use an inner print list; you can access hierarchical fields as though they belong to a single record. The following example prints the first five records in a flat view and then

splays a specific record from the VENDORS domain:

```
'R> READY FLAT_PART_VIEW
'R> PRINT FIRST 5 FLAT_PART_VIEW

 Part
Number   Type --------------Vendor Name---------------

-3556-78 MEMO U.S. SEALS
-1110-85 REPR HIGH ENERGY CORP
-7896-12 REPR EMI TECHNOLOGY INC
-8767-78 WSUP ELECTRONIC SUPPLY CO.
-4058-32 CALL SYSTEMS HDWE REPS

R> PRINT VEND_NAME WITH PART_ID="CF405832"

------------Vendor Name---------------

STEMS HDWE REPS
```

1.8 Sample Procedures Using DBMS Domains

ɔu can define DATATRIEVE procedures that let you query a VAX DBMS ıtabase. A DATATRIEVE procedure is a fixed sequence of commands and state-ents that you create, name, and store in the Common Data Dictionary. For ıost any series of commands and statements you use repeatedly, you can save ıurself time by defining a procedure.

procedure can contain any number of the following DATATRIEVE elements:

- Full DATATRIEVE commands and statements
- Command and statement clauses and arguments
- Comments

ı define a procedure, you enter the DEFINE PROCEDURE command at the ΓR> prompt. DATATRIEVE prompts with the DFN> prompt to indicate that ıu can enter a procedure definition. You end the procedure definition with an ND_PROCEDURE keyword on a line by itself.

ır example, the following procedure searches for a division associated with the ıployee name you specify.

. Ready the domains EMPLOYEES and DIVISIONS.

. Form a temporary record stream of the employee record you want. The prompt option (*) lets you specify the employee's name when you execute the procedure.

3. Display the owner of the CONSISTS_OF set of which the employee is a member.

```
DTR> DEFINE PROCEDURE EMPLOYEE_SEARCH
DFN> READY EMPLOYEES, DIVISIONS                          <---------- (1)
DFN> PRINT "This procedure searches to find"
DFN> PRINT "the division associated with"
DFN> PRINT "the employee you specify."
DFN> PRINT SKIP
DFN> FOR EMPLOYEES WITH EMP_LAST_NAME =                  <---------- (2)
DFN>  *."the employee's last name in capital letters"
DFN> PRINT DIVISIONS OWNER CONSISTS_OF                    <---------- (3)
DFN> COMMIT EMPLOYEES, DIVISIONS
DFN> END_PROCEDURE
DTR>
```

To execute the procedure, enter:

```
DTR> :EMPLOYEE_SEARCH

This procedure searches to find
the division associated with the
employee you specify.


Enter the employee's last name in capital letters: ZOTTO


Division Name-------

RKO5 DEVELOPMENT
```

For information on the COMMIT statement, see Section 14.10.4. In another example, you can define a procedure to find a vendor name and all the parts produced by that vendor. This procedure:

1. Readies the domains VENDORS, SUPPLIES, and PART_S
2. Uppercases the vendor names you enter following the prompt and finds them in the VENDORS domain
3. Finishes the related records in the SUPPLIES domain
4. Uses the context from the FOR statement in step 3 to print the related part number from the PART_S domain
5. Finishes the readied domains

```
[R> DEFINE PROCEDURE VENDOR_PARTS
'N> READY VENDORS, SUPPLIES, PART_S                    <---------- (1)
'N> PRINT "This procedure searches to find"
'N> PRINT "the part associated with "
'N> PRINT "the vendor you specify."
'N> PRINT SKIP
'N> FOR VENDORS WITH VEND_NAME =                       <---------- (2)
'N>   FN$UPCASE(*." Name of Vendor ")
'N> FOR SUPPLIES MEMBER OF VENDOR_SUPPLY               <---------- (3)
'N> PRINT ALL PART_ID OF PART_S OWNER OF PART_INFO     <---------- (4)
'N> FINISH VENDORS, SUPPLIES, PART_S                   <---------- (5)
'N> END_PROCEDURE
[R>
```

o execute the procedure, enter:

```
[R> :VENDOR_PARTS
ıis procedure searches to find
ıe part number associated with the
:ndor you specify.

ıter Name of Vendor: quality comps

   Part
  Number

}-3162-34
}-1052-00
}-0956-78

[R>
```

4.9 Modifying Individual Fields in a Record

ou can modify a field in a DBMS record just as you do in an RMS domain using ıe DATATRIEVE MODIFY statement. The following example modifies the MP_PHONE field of the EMPLOYEE record:

1. Ready the DBMS EMPLOYEES domain for WRITE access.
2. Modify the field EMP_PHONE.

```
:R> READY EMPLOYEES WRITE    <---------- (1)
:R>
:R> MODIFY EMP_PHONE OF      <---------- (2)
)N>  EMPLOYEES WITH EMP_ID = "53456"
ıter EMP_PHONE: 5345
:R>
```

For a complete discussion of the MODIFY statement in DATATRIEVE, see Chapter 4. Using the syntax described in Chapter 4, you can modify all or some fields within a single record occurrence or within a collection of records.

If you change a field that has a DBMS CHECK clause, DBMS checks the value you enter for that field. If the value violates the CHECK clause, DATATRIEVE returns a DBMS error and does not prompt for the field.

In general, modifying a record affects at least the data portion of the record. If, however, you modify a field that is a sort key or a hash key for a set, DBMS automatically reorders the members of the set. See the *VAX DBMS Introduction to Data Manipulation* for more information about modifying sort or hash keys.

14.10 Storing DBMS Records and Modifying Sets

When you add a record to a DBMS database, you can affect other members of the sets in which the new record participates. You may also wish to disconnect a record from a particular set occurrence and perhaps reconnect it with another set occurrence.

DATATRIEVE provides you with several statements you can use with DBMS domains to manipulate records as owners and members of sets:

- The STORE statement -- adds a new record to the database and automatically connects the record to each set of which it is an automatic member
- The CONNECT statement -- connects a selected member record to a set
- The DISCONNECT statement -- disconnects a member record from each set you specify (you cannot disconnect owner records)
- The RECONNECT statement -- disconnects a member record from each set occurrence you specify and connects the record to another set occurrence you specify (you cannot reconnect owner records)

14.10.1 Storing and Connecting Records

When you store a new record or when you want to connect a particular record to a set occurrence, the procedure you use depends on whether the set is an automatic or manual member of a set.

nsertion into a set can be:

Automatic

DATATRIEVE automatically inserts the record into the set when you store it.

Manual

After modifying or storing the record, you can connect it to the set of which it is a member or leave it unconnected in the database.

'or example, when you use the SHOW SETS command you can see the characeristics identified by DATATRIEVE for member domains:

```
TR> SET DICTIONARY CDD$TOP.DTR$LIB.DEMO.DBMS
TR> READY VENDORS, SUPPLIES
TR> SHOW SETS

et: VENDOR_SUPPLY
      Owner:  VENDORS
      Member: SUPPLIES, automatic fixed

et: ALL_VENDORS
     Member: Vendors, automatic fixed
```

UPPLIES is an automatic member of the set VENDOR_SUPPLY. The autoıatic characteristic indicates that when you store a new supplies record, it is ıtomatically connected to the set VENDOR_SUPPLY.

UPPLIES is also a member of a system-owned set ALL_VENDORS. It autoıatically participates in this set.

he fixed characteristic means that the domain SUPPLIES must be a member of ıe set. Also, the record occurrence cannot be connected to any set occurrence :her than the one to which it belongs when it is stored. For example, a particular UPPLY record cannot be connected to any other vendor through ENDOR_SUPPLY than the one it was associated with at the time the record as stored. This characteristic is discussed in the section on erasing and disconecting records from sets.

he following two sections discuss automatic and manual insertion in a set.

1.10.1.1 Automatic Insertion -- If a record is an automatic member of a set, hen you use the DATATRIEVE STORE statement to store new records in the BMS database, the record is automatically inserted into the set.

If you are storing a new record and only want to connect it with a system-owned set, you do not have to establish context to insert a record. After readying the domain for WRITE access, all you need to store a record into a system-owned set is a STORE statement. For example:

```
DTR> READY EMPLOYEES WRITE
DTR> STORE EMPLOYEES
Enter EMP_ID: 53456
Enter EMP_LAST_NAME: WINSLEE
Enter EMP_FIRST_NAME: JOANNE
Enter EMP_PHONE: 5324
Enter EMP_LOC: AS

DTR>
```

The new record automatically becomes a member of system-owned sets in which the record (domain in DATATRIEVE) participates. For example, a new employees record is automatically part of the ALL_EMPLOYEES set in this example.

If a newly stored record is an automatic member of a set *not owned by the system*, though DATATRIEVE automatically connects the record to the set in which the record participates, *you must provide the context* for the set occurrence to which you want to connect the record.

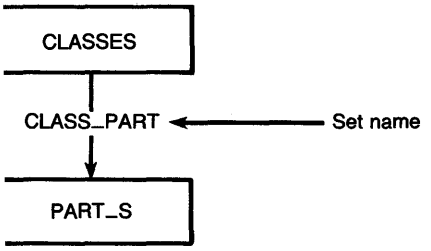
Therefore, when you use the STORE statement:

- You use the CURRENCY clause to provide context for automatic members of sets that are not owned by the system
- The record is automatically connected to each set of which it is an automatic member

As with the display and print operations, DATATRIEVE uses the single-record context you supply to identify the set occurrence (and sometimes to select the position in that occurrence) when you modify sets or store records. If you fail to supply the single record context, DATATRIEVE may insert a record in the wrong place, the record may not be moved, or you may receive an error message. Once you have established a single record context, you can use this context to store many records without providing new single record context each time.

The following example stores a new parts record in the PART_S domain and connects it to a specific occurrence of the CLASS_PART set. PART_S is an automatic member of the set CLASS_PART, shown in Figure 14-6. You:

- Establish a single-record context for the record *before* you store the record
- Store the record using the DATATRIEVE CURRENCY clause



igure 14-6: DBMS Set CLASS_PART

1. Ready the PART_S and CLASSES domains for WRITE access.
2. Use SHOW SETS to see that PART_S is an automatic member of the set CLASS_PART and that CLASS_PART is owned by another domain, CLASSES.
3. Establish a context with the single-set occurrence of the set CLASS_PART with which you want to connect the new part record.

   You do this by using the FIND and SELECT statements to establish a single-record occurrence of the domain CLASSES with the class code of BR. When you select the record in the domain CLASSES, you establish context with the correct occurrence of the set CLASS_PART.
4. Store the new record in the PART_S domain and connect it to the current occurrence of the CLASS_PART set. DATATRIEVE prompts you for data values for the new record.
5. Check the new record. Because the CLASSES record BR is still the selected record, you can use the PRINT statement with the MEMBER clause of the record selection expression to see if the GUDGEON record is now a member of the class BR.

```
[R> READY PART_S WRITE, CLASSES WRITE          <---------- (1)
[R>
[R> SHOW SETS                                  <---------- (2)
:t: CLASS_PART
      Owner: CLASSES
      Member: PART_S, automatic mandatory
      .
      .
      .
```

(continued on next page)

```
DTR> FIND BR IN CLASSES WITH CLASS_CODE = "BR"        <-------- (3)
[1 record found]
DTR> SELECT
DTR>
DTR> STORE PART_S CURRENCY BR.CLASS_PART              <-------- (4)
Enter PART_ID: BR902334
Enter PART_DESC: GUDGEON
Enter PART_STATUS: G
Enter PART_PRICE: 902.00
Enter PART_COST: 231.00
Enter PART_SUPPORT: X
DTR>
DTR> PRINT PART_ID, PART_DESC OF PART_S MEMBER        <--------- (5)
[Looking for set name]
CON> CLASS_PART

   Part
  Number    -----------------Part Description----------------

BR-1234-56 LA34
BR-3467-91 LA120
BR-8901-23 LA36
BR-9023-34 GUDGEON

DTR>
```

Note that you use the CURRENCY clause of the STORE statement to specify the exact set occurrence to which DATATRIEVE should connect the record. Note that the record being stored with a CURRENCY clause must be an automatic member of the set.

14.10.1.2 Manual Insertion -- If a newly stored record is a manual member of a set, you must use the CONNECT statement to insert the record into the set. (You can also leave the record unconnected to any set.) You must establish the context when you connect the record to the set.

The following example connects an EMPLOYEE record to a DIVISIONS set. Because EMPLOYEES is a manual member of the set CONSISTS_OF, you must specifically insert the new employee record in the set CONSISTS_OF.

1. Ready both the EMPLOYEES and DIVISIONS domains for WRITE access.
2. Store the new employee record. (DBMS automatically connects the record of the new employee to the system-owned set, ALL_EMPLOYEES, and to the sets owned by the employee record, MANAGES and RESPONSIBLE_FOR.)
3. Specify each record in the DIVISIONS domain to which you want to connect the new employee record. Specify the context variable DIV, so you can use it later to qualify the set name.

4. Specify the new employee record you stored in step 2.

5. Use the CONNECT statement to connect the new employee record to the SOFTWARE division through the set CONSISTS_OF.

6. Print the employees from the set occurrence to see that you made the correct set connection.

```
ΓR> READY EMPLOYEES WRITE, DIVISIONS WRITE          <-------- (1)
ΓR> STORE EMPLOYEES                                 <-------- (2)
ıter EMP_ID: 53456
ıter EMP_LAST_NAME: JANOV
ıter EMP_FIRST_NAME: LESLEY
ıter EMP_PHONE: 5324
ıter EMP_LOC: AS

ΓR> FOR DIV IN DIVISIONS WITH DIV_NAME = "SOFTWARE"   <----- (3)
)N>    FOR EMP IN EMPLOYEES WITH
)N>                          EMP_LAST_NAME = "JANOV"  <----- (4)
)N>    BEGIN
)N>    CONNECT EMP TO DIV.CONSISTS_OF                 <----- (5)
)N> PRINT EMP_LAST_NAME OF EMPLOYEES MEMBER           <----- (6)
)N>      DIV.CONSISTS_OF
)N>    END

ıst Name------

)TCHINGS
\COBONE
\SCAL
\NOV

:R>
```

ote that you connect the employee named JANOV to the occurrence of the IVISIONS domain (DIV) through the set CONSISTS_OF. Where the record is serted into the set depends on the set-ordering criteria defined in the schema. ee the *VAX DBMS Database Design Guide* for more information on set order.

1.10.2 Erasing, Disconnecting, and Reconnecting Records with Sets

ou use the ERASE statement to remove a record. Because the ERASE statement can delete more than you intend, use it with caution. Accidental deletions ın occur because of the ERASE statement's "cascading effect." This cascading fect can happen whenever the erased record is the owner of a set. Thus, if the ırrent record is an owner of a set type, ERASE deletes all of the following:

The current record

All records in sets owned by the current record

Any records in sets owned by those members, and so forth

You can remove a record from a set either by erasing it with the DATATRIEVE ERASE statement or disconnecting and reconnecting it with sets.

The removal characteristics of a record determine the way in which records can be removed from sets, and whether they can be removed from sets. The removal characteristic of a record is one of the following:

- Fixed

  You cannot disconnect the record from its set occurrence unless you erase the record from the database.

- Mandatory

  You cannot use DISCONNECT to remove the record from a set occurrence. However, you can use RECONNECT to move it from one occurrence of the set type to another.

- Optional

  You can use either DISCONNECT or RECONNECT to remove the record from a set occurrence.

The following sections discuss removal from a set in more detail.

14.10.2.1 Erasing DBMS Records -- Records that are fixed members of sets, once connected to a set occurrence, must be a member of that specific set occurrence until they are deleted from the database. They cannot be disconnected and remain in the database or reconnected to some other set occurrence.

The fixed characteristic is very common with system-owned sets that are used to keep large numbers of records on file. For example, an organization usually keeps a generalized listing of all employees. Such a listing can be maintained by a system-owned set, as in the PARTS database with the ALL_EMPLOYEES set.

In a previous example, you added the record of the employee named JANOV to the ALL_EMPLOYEES set and connected it to the SOFTWARE division. The following example erases the record JANOV from the database, deleting it from the system-owned ALL_EMPLOYEES set. As a result of being erased, the record in the example is also disconnected from the SOFTWARE Group:

1. Erasing the record for JANOV from the database involves the domains EMPLOYEES, DIVISIONS, and PART_S. Ready for WRITE access all domains that are affected by the loss of an employee.

   If you do not ready all necessary domains, you might encounter a problem when you attempt to erase the record. Realms are ultimately associated with storage areas, unless the files themselves are readied through a READY command that readies at least one of the domains in that file.

Therefore, you might receive an error from DBMS stating that a particular storage area has not been readied.

2. Find and select the record you want to erase.

   If the record you erase is the owner of any sets (EMPLOYEES is owner of the sets RESPONSIBLE_FOR and MANAGES), records in member domains (PART_S and DIVISIONS) are also erased. Therefore, be sure that you know exactly what you are erasing. Table 14-1 summarizes the effects of erasing a record on the record and its members.

3. Make sure that this is the record you want to erase and then erase the record.

4. Show that DATATRIEVE prints nothing in response to the PRINT CURRENT statement. The current collection is now empty.

5. Try to find the record you erased.

```
TR> READY EMPLOYEES WRITE, DIVISIONS WRITE,          <-------- (1)
ON>  PART_S WRITE
TR>
TR> FIND EMPLOYEES WITH EMP_LAST_NAME = "JANOV"      <-------- (2)
1 record found]
TR> SELECT;PRINT EMP_LAST_NAME                       <-------- (3)

ast Name-----------

ANOV

TR> ERASE
TR> PRINT CURRENT                                    <-------- (4)
TR>
TR> FIND EMPLOYEES WITH EMP_LAST_NAME = "JANOV"      <-------- (5)
) records found]
TR>
```

he employee named JANOV is no longer in the database. As a result of being rased, his employee record has also been disconnected from all the sets of which was a member.

4.10.2.2 Disconnecting and Reconnecting DBMS Records from Sets --

ecords that are mandatory members of sets can move from one occurrence of set to another. However, records with mandatory membership in a set must ways be members of some occurrence of that set type once they have been nnected.

he advantage of such membership is the ability to change your mind about the tributes of a member record.

For example, suppose your inventory supervisor wants to move a part record (terminal stands) from one class of the domain CLASSES (terminal assemblies) to another class in that same domain (video terminals). The PART_S domain, which contains the parts record, is a mandatory member of the set CLASS_PART.

In the following example, you use the set CLASS_PART.

1. Ready the necessary domains. Because you are disconnecting a PART_S record from an occurrence of CLASSES and then connecting it to another occurrence, you must ready both those domains for MODIFY access.
2. Display the PART_S records that contain the letters BU in their PART_ID number. You want to move the last two parts (the terminal tables for the VT52 and VT100) to the occurrence of CLASS_PART owned by BT.
3. For these two part records, show that they are connected to the particular occurrence of the set CLASS_PART that is owned by CLASSES record BU. Those two records belong to the class called TERMINAL ASSEMBLIES.
4. Now, determine where you would like to move these two records. First, find and display all the PART_S records that contain the letters BT in their PART_ID.
5. Then, using a FOR loop, find and display the record occurrence of the domain CLASSES to which these two records are related. This is the record occurrence, VIDEO TERMINALS, to which you want to reconnect your two terminal tables.
6. Create the necessary context to reconnect the terminal tables to the video terminal set. Use nested FOR loops to create the context and a RECONNECT statement to move the records.
7. Make sure that you actually moved the records. Find and select the CLASSES record with the value BT for CODE.

```
DTR> READY PART_S MODIFY, CLASSES MODIFY          <---------- (1)

DTR> FIND PART_S WITH PART_ID CONT "BU"           <---------- (2)
[7 records found]
```

```
TR> PRINT PART_ID, PART_DESC OF CURRENT

  Part
 Number    -----------------Part Description-----------------

U-1045-68 FREE-STANDING FRAME ASSEMBLY
U-2345-67 VIDEO TUBE
U-3161-25 VT100 NUMERIC KEYPAD ASSY
U-7014-68 VT100 MONITOR UNIT
U-7014-65 VT100 KEYBOARD UNIT
U-3456-70 TERMINAL TABLE VT100
U-0345-67 TERMINAL TABLE VT52

TR> FOR PART_S WITH PART_ID CONT "BU345670",          <--------- (3)
)N> "BU034567" PRINT CLASSES WITHIN CLASS_PART

)de -Class Description-- St

3U  TERMINAL ASSEMBLIES  Y

TR> FIND VIDEO_TERMS IN PART_S WITH PART_ID CONT "BT"    <--------(4)
2 records found]
TR> PRINT PART_ID, PART_DESC OF VIDEO_TERMS

  Part
 Number    -----------------Part Description-----------------

T-1634-56 VT100
T-0456-78 VT52


R> FOR VIDEO_TERMS PRINT CLASSES WITHIN CLASS_PART       <--------(5)

)de -Class Description-- St

T  VIDEO TERMINALS      G
T  VIDEO TERMINALS      G

R> FOR VT IN CLASSES WITH CODE = "BT"                    <---------(6)
N> FOR TABLES IN PART_S WITH PART_ID = "BU345670" OR
N> PART_ID = "BU034567" RECONNECT TABLES TO VT.CLASS_PART
R>

R> FIND CLASSES WITH CODE = "BT"                <--------------------(7)
 record found]
R> SELECT
R> PRINT PART_ID, PART_DESC OF PART_S MEMBER OF CLASS_PART

  Part
 Number    -----------------Part Description-----------------

-0456-78 VT52
-1634-56 VT100
-0345-67 TERMINAL TABLE VT52
-3456-70 TERMINAL TABLE VT100

R>
```

Note that the terminal tables are now members of a new set occurrence. Check the former location of those records to make sure they are no longer there:

```
DTR> FIND CLASSES WITH CLASS_CODE = "BU"
[1 record found]
DTR> SELECT
DTR> PRINT CURRENT

Code -Class Description-- St

 BU  TERMINAL ASSEMBLIES  Y

DTR> PRINT PART_ID, PART_DESC OF PART_S MEMBER CLASS_PART

   Part
  Number    -----------------Part Description-----------------

BU-1045-68 FREE-STANDING FRAME ASSEMBLY
BU-2345-67 VIDEO TUBE
BU-3161-25 VT100 NUMERIC KEYPAD ASSY
BU-7014-65 VT100 KEYBOARD UNIT
BU-7014-68 VT100 MONITOR UNIT
DTR>
```

The terminal tables are no longer in the occurrence of CLASS_PART owned by BU.

14.10.2.3 Disconnecting and Connecting DBMS Records from Sets --

Records that are optional members of sets can belong to an occurrence of a set type or not belong to any occurrence at all. For example, the PARTS database has two system sets, ALL_PARTS and ALL_PARTS_ACTIVE. The ALL_PARTS set describes all parts cataloged by a firm. The ALL_PARTS_ACTIVE set consists of all parts currently in production or inventory. As a part is retired, it may be removed from the ALL_PARTS_ACTIVE set but retained in the ALL_PARTS listing of everything ever made.

You can use the DISCONNECT statement to remove OPTIONAL members of sets from those sets. You can later use the CONNECT statement to insert the disconnected record into another occurrence of the same set type or you can let that record remain disconnected in the database.

The following example disconnects a part from the ALL_PARTS_ACTIVE set:

1. Ready the necessary domains. Because you are removing a PART_S record from a system-owned set, you need only ready PART_S for WRITE access.

2. Display the record that you want to remove from the ALL_PARTS_ACTIVE set.

   Because ALL_PARTS_ACTIVE is a system-owned set, you do not have to establish context with the SELECT statement when you want to display the members of the set.

3. Establish the context you need to disconnnect a record by specifying in a FOR loop the record you want removed from the set. Then use the DISCONNECT statement to remove it.

4. Check the ALL_PARTS_ACTIVE set to make sure the record is no longer there. DATATRIEVE responds with the DTR> prompt rather than a display of the record; the part record is no longer a member of the ALL_PARTS_ACTIVE set.

```
TR> READY PART_S WRITE                              <---------- (1)
TR>
TR> PRINT PART_S MEMBER ALL_PARTS_ACTIVE WITH       <------- (2)
)N> PART_ID = "BU104568"

  Part                                                  Unit
 Number    -----------Part Description------------ST   Price

I-1045-68 FREE-STANDING FRAME ASSEMBLY             G    $305


'R> FOR P IN PART_S WITH PART_ID = "BU104568"       <------- (3)
IN> DISCONNECT P FROM ALL_PARTS_ACTIVE
'R>
'R> PRINT PART_S MEMBER ALL_PARTS_ACTIVE WITH
IN> PART_ID = "BU104568"                            <------- (4)
'R>
```

1.10.3 Summary of Membership Characteristics

ıe record membership criteria limit the changes you can make to a database. ıble 14-1 summarizes the effects of various statements on the record being odified.

Table 14-1: Insertion, Retention, and Database Operations

INSERTION RETENTION	Effect on Target Record						Effect on Member	
	CONNECT	DISCONNECT	ERASE	MODIFY	RECONNECT	STORE	ERASE ALL	ERASE
AUTOMATIC FIXED	Not Possible	Not Allowed	Erase	Reorder	Not Allowed	Insert	Erase	Erase
AUTOMATIC MANDATORY	Not Possible	Not Allowed	Erase	Reorder	Move Reorder	Insert	Erase	Not Allo
AUTOMATIC OPTIONAL	Insert	Remove	Erase	Reorder	Move Reorder	Insert	Erase	Remove
MANUAL FIXED	Insert	Not Allowed	Erase	Reorder	Not Allowed	No Effect	Erase	Erase
MANUAL MANDATORY	Insert	Not Allowed	Erase	Reorder	Move Reorder	No Effect	Erase	Not Allo
MANUAL OPTIONAL	Insert	Remove	Erase	Reorder	Move Reorder	No Effect	Erase	Remove

Notes to Table

Insert

Connects a record into an occurrence of the given set type.

Move

Reconnects a record from one occurrence of the given set type to another occurrence of the same set type. This operation is equivalent to "Remove" followed by "Insert."

Remove

Disconnects a record from an occurrence of the given set type.

Reorder

Can affect set ordering. Can cause reordering within s set occurrence.

No Effect

Does not affect set membership.

Not Allowed

Returns an exception.

Not Possible

Cannot be done.

MK-005

14.10.4 Writing Changes to the Database

To write the changes you made to the database, you must enter a COMMIT statement. If, however, you do not want to save the changes you made, you can enter a ROLLBACK statement and leave the database as it was. The following statement rolls back any changes:

```
DTR> ROLLBACK
ROLLBACK executed; collection CURRENT automatically released
DTR>
```

DATATRIEVE automatically readies database domains again after you have con mitted or rolled back.

- A COMMIT statement performs a DBMS COMMIT RETAINING.
- A ROLLBACK statement is equivalent to a DATATRIEVE ABORT.

The DATATRIEVE FINISH and EXIT commands end access to domains or DBMS records. The FINISH command executes a DBMS COMMIT (without th

ETAINING argument) when you finish the last readied domain or record, or
ıish them all at once. The EXIT command also executes a DBMS COMMIT
atement.

Note

There is an important difference between the DATATRIEVE EXIT command and the DBQ EXIT command: the DATATRIEVE EXIT command executes a DBMS COMMIT statement; the DBQ EXIT command issues a DBMS ROLLBACK.

ɔ write changes to the database, end access to the domains or DBMS records,
ıd remain in DATATRIEVE, use the FINISH command. To write changes to
e database and end your DATATRIEVE session as well as access to domains
ıd records, use the EXIT command:

```
R> FINISH
R> EXIT
```

4.11 Optimizing Performance

hen using DATATRIEVE to work with DBMS databases, keep in mind the fol-
wing considerations:

Unless you specify otherwise, DATATRIEVE always starts reading database areas at page one, line one. DBMS is designed to optimize access paths to records through set chain pointers, indexes, and hashing algorithms. Use a set name whenever possible to optimize your database access paths and prevent sequential reads of database areas.

To minimize record locking, be sure to issue COMMIT or ROLLBACK statements regularly to explicitly end database transactions. Locks prevent other users from accessing a record and can prevent access to other records because that record contains pointer information that also gets locked.

Using DATATRIEVE with Rdb 15

'ou can use VAX DATATRIEVE to access VAX Rdb databases, the DIGITAL ımily of relational database management systems. Rdb provides the advantages f a database management system, including data security and integrity. At the ame time, its relational model of data organization is easier to understand and to se than the network (CODASYL-style) model of data organization.

)ATATRIEVE alone provides excellent data access when your database contains ;wer than 5000 records. If your database is larger than that, using Rdb for data ;orage optimizes response time for your DATATRIEVE queries, data mainten- nce, and report-writing tasks.

he DATATRIEVE statements you use for data queries and report writing are ıe same, whether you are accessing a file-structured database or an Rdb one. ' you currently use DATATRIEVE to create and maintain file-structured ıtabases, you need to learn some extensions to the DATATRIEVE language to ;cess and maintain an Rdb database.

5.1 Getting Started with DATATRIEVE and Rdb

ı an Rdb database, data is organized into *relations*. Relations are simply tables. table has a horizontal dimension (rows) and a vertical dimension (columns). A ıw in a relation is a set of data fields, analogous to a record in a file. The fields in ıch row define the columns. In this chapter, the term *record* refers to an entire ıw in a database relation. Figure 15-1 shows part of the structure of a relation ılled DEPARTMENTS in the PERSONNEL database installed with the ATATRIEVE UETP (User Environment Test Package).

igure 15-2 shows the relations and fields for the sample PERSONNEL ıtabase. Examples in this chapter refer to the relation and field names in the ERSONNEL database. The data shown in the examples may be different than ıe data that appears on your screen.

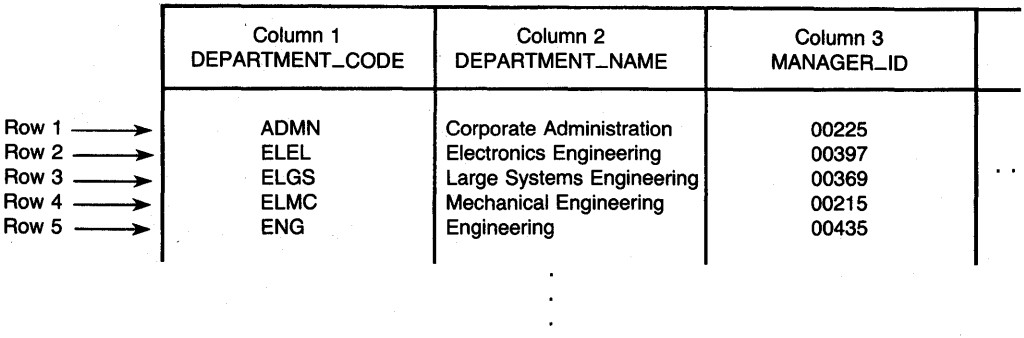


Figure 15-1: Sample Rdb Relation

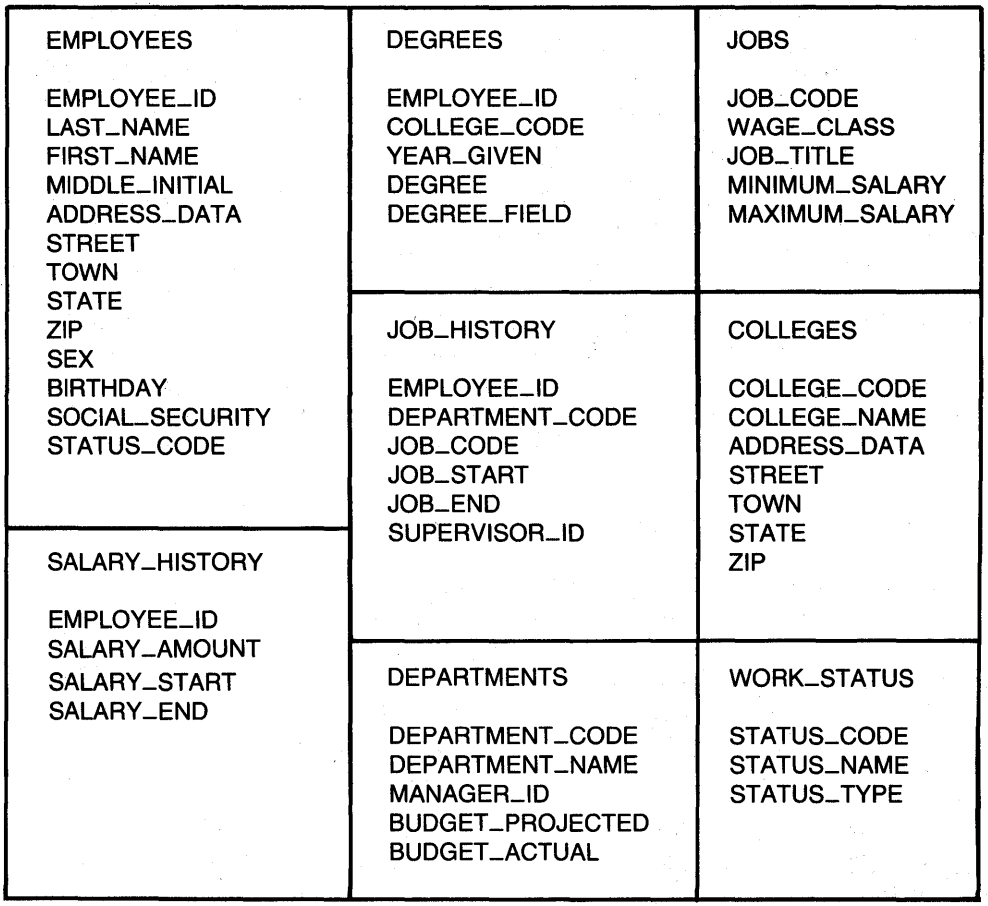


Figure 15-2: Sample Rdb Database

he following command sets the CDD default to the directory that contains the ıtabase definition and the domain definitions used in this chapter.

```
'R> SET DICTIONARY CDD$TOP.DTR$LIB.DEMO.RDB
```

he examples and references in this chapter refer to Rdb/VMS. DATATRIEVE ɔes, however, support the family of Rdb relational database products. For exam-es of using DATATRIEVE with Rdb/ELN, see the documentation for that ·oduct.

5.2 Creating a Path Name for the Database

ou must use an interactive Rdb utility, RDO in Rdb/VMS, to define your Rdb ıtabase. The *VAX Rdb/VMS Guide to Database Design and Definition* tells you ɔw to do this for an Rdb/VMS database.

ɔ define an Rdb/ELN database, use the data definition language compiler for db/ELN. The *VAX Rdb/ELN Application Development Guide* tells you how to ɔ this.

ɔ access an Rdb/VMS database with DATATRIEVE, it must have a CDD path ıme. A path name may be specified when an Rdb database is created. If the Rdb ıtabase you want to access with DATATRIEVE already has a path name, you n use that path name to access the database.

you want to access an Rdb database that does not have a path name, you can eate a path name with the DATATRIEVE DEFINE DATABASE command:

EFINE DATABASE Rdb-database-path ON root-file-spec;

ɔb-database-path

the path name you want to assign the database.

ot-file-spec

the file specification of the database file.

ıe following examples illustrate three valid commands to establish a path name : the PERSONNEL database:

```
R> DEFINE DATABASE CDD$TOP.DEPT29.PERSONNEL ON
N>                 DBA2:[D29.DAT]PERSONNEL.RDB;
R>

R> DEFINE DATABASE CDD$TOP.DTR$LIB.DEMO.RDB.PERSONNEL
N> ON DTR$LIBRARY:PERSONNEL;
R>

R> DEFINE DATABASE PERSONNEL ON PERSONNEL;
R>
```

The first two examples specify full path names and file specifications for the databases. The third example relies entirely on current defaults, both for the path name and the file specification.

Because VAX Rdb/ELN does not use the CDD, you must issue a DATATRIEVE DEFINE DATABASE command to establish a path name for the Rdb/ELN database.

15.3 Accessing the Database

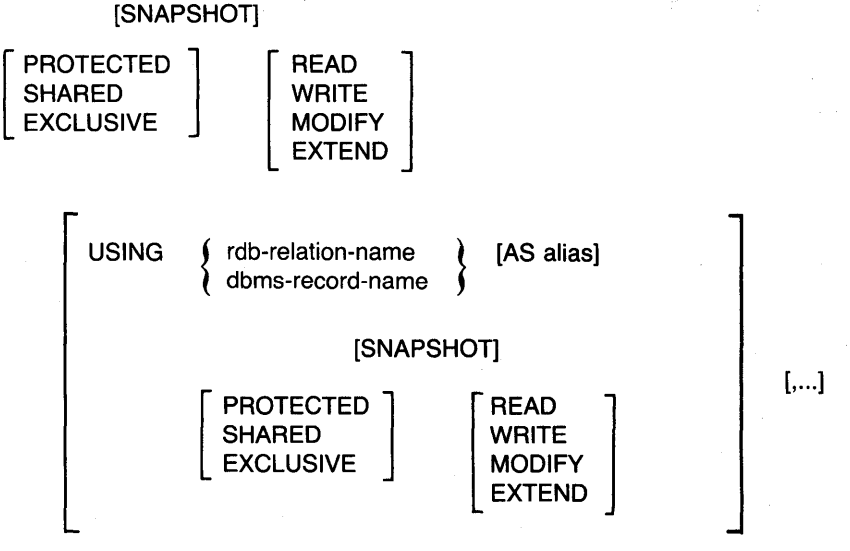
After you create a path name for your Rdb database, you can access it with DATATRIEVE in either of two ways:

- Ready the database directly.
- Define a domain for each Rdb relation that you want to access, and then ready the domains you want to use.

15.3.1 Readying an Rdb Database Directly

Using the following syntax, you can ready an Rdb database without defining DATATRIEVE domains for any relations:

READY database-path-name



latabase-path-name

s the CDD path name of the Rdb database.

ıccess mode

s the method (SNAPSHOT, PROTECTED, SHARED, or EXCLUSIVE) by
/hich you access the data. In Rdb, SNAPSHOT or read-only, is the default access
ıode.

ccess option

s the option (READ, WRITE, MODIFY, or EXTEND) by which you access the
ldb data.

ISING clause

imits access to specified relations. If you omit the USING clause of the READY
)ATABASE command, all relations in the database are readied.

elation-name

s the name used in the Rdb utility to define the relation.

lias

s a name you use to refer to the relation specified. If you include an alias, you
ıust use it in all the DATATRIEVE statements and commands that refer to the
eadied relation.

he following examples illustrate various ways to ready a database directly.

eady an entire database:

```
TR> READY PERSONNEL
TR>

TR> READY CDD$TOP.DEPT39.PERSONNEL MODIFY
TR>
```

eady selected relations:

```
R> READY CDD$TOP.DEPT39.PERSONNEL USING EMPLOYEES
R>

R> READY PERSONNEL USING EMPLOYEES, SALARY_HISTORY WRITE
R>
```

The results of the READY command are discussed in Section 15.3.3 of this chapter.

15.3.2 Defining and Readying Rdb Domains

You can define a DATATRIEVE domain for each Rdb relation that you want to access. Then, you use the domain names to ready the database relations you want to access.

Accessing your database through domains slows DATATRIEVE performance; accessing a database directly works more quickly. However, you may want to define domains because:

- You can define DATATRIEVE view domains of Rdb relations
- You can link a form definition with a domain definition using the FORM IS syntax

You can use the DISPLAY FORM statement to link a form with a relation that is not defined as a domain. However, if you want the relation to display data automatically on a form, you must create a domain definition and include the FORM IS clause:

```
DEFINE DOMAIN domain-name [USING] relation-name
                [OF [DATABASE]] database-path
                [FORM [IS] form-name [IN] form-library];
```

domain-name

Is the name you want for the domain. It can be the same as the relation name, but it does not have to be the same.

relation-name

Is the name used to define the relation in the Rdb database.

database-path

Is the CDD path name of your Rdb database.

form-name

Is the name given the form when it was created.

form-library

Is the name of the form library.

For example, the following commands define domains for the EMPLOYEES and SALARY_HISTORY relations in the PERSONNEL database:

```
DTR> DEFINE DOMAIN EMPLOYEES USING EMPLOYEES OF
DFN> DATABASE CDD$TOP.DTR$LIB.DEMO.RDB.PERSONNEL;
DTR>

DTR> DEFINE DOMAIN SALARY_HISTORY USING SALARY_HISTORY OF
DFN> DATABASE CDD$TOP.DTR$LIB.DEMO.RDB.PERSONNEL
DFN> FORM IS SALARYHST IN FORMSLIB;
DTR>
```

After you define a domain for a relation in an Rdb database, you can ready it. You do not supply a DATATRIEVE definition of the fields and indexes associated with each domain. DATATRIEVE retrieves this information from Rdb when you ready the domains:

```
DTR> READY SALARY_HISTORY, EMPLOYEES
DTR>
```

5.3.3 Results of the READY Command

If you do not specify an access mode or an access option for an Rdb database, Rdb domain, or Rdb relation, the default access is SNAPSHOT. Other users can have READ, WRITE, MODIFY or EXTEND access to the database, domain, or relation. The following example shows the result of readying a database directly. There are no domains defined for the relations in this example:

```
DTR> READY PERSONNEL
DTR> SHOW READY
Ready sources:
  WORK_STATUS:  Relation, Rdb, snapshot read
          <CDD$TOP.DTR$LIB.DEMO.RDB.PERSONNEL;1>
  DEGREES:  Relation, Rdb, snapshot read
          <CDD$TOP.DTR$LIB.DEMO.RDB.PERSONNEL;1>
  COLLEGES:  Relation, Rdb, snapshot read
          <CDD$TOP.DTR$LIB.DEMO.RDB.PERSONNEL;1>
  DEPARTMENTS:  Relation, Rdb, snapshot read
          <CDD$TOP.DTR$LIB.DEMO.RDB.PERSONNEL;1>
  JOBS:  Relation, Rdb, snapshot read
          <CDD$TOP.DTR$LIB.DEMO.RDB.PERSONNEL;1>
  SALARY_HISTORY:  Relation, Rdb, snapshot read
          <CDD$TOP.DTR$LIB.DEMO.RDB.PERSONNEL;1>
  JOB_HISTORY:  Relation, Rdb, snapshot read
          <CDD$TOP.DTR$LIB.DEMO.RDB.PERSONNEL;1>
  EMPLOYEES:  Relation, Rdb, snapshot read
          <CDD$TOP.DTR$LIB.DEMO.RDB.PERSONNEL;1>
0 loaded tables.

DTR>
```

The default access mode, SNAPSHOT, is called a read-only or "snapshot" ready. It lets you read data without locking other users out of the database. In such an access mode, you see a "picture" of the database, exactly as it was when you readied the relation, domain, or database. Other users can access and change the data in the database you have readied.

You do not see changes other users make until you issue a FINISH, COMMIT, ROLLBACK, or another READY command. the relation or database. When you use this access mode, there is no write locking of the database. Rdb does place read locks in SNAPSHOT mode, however.

In order to have SNAPSHOT access, you must ready all relations that you want to use in a database with SNAPSHOT access. Thus, when you ready a database, relation, or domain with SNAPSHOT access, and you also ready one or more relations or domains from that database with a more restrictive mode, you actually have SHARED READ access rather than SNAPSHOT access to all the domains or relations. In SHARED READ mode, DATATRIEVE uses standard read locks and you see other users' modifications as they are committed.

15.4 Using Views

A view is a "virtual" relation. Its definition specifies fields from one or more source relations. A view contains no data. You cannot always use views for store or modify operations, but you can use them to display records.

You might want to create a view that is a subset of the fields in a relation when you do not want certain users to see confidential data. You would give these users access to the view, but not to the relation itself.

You might want to create a view that joins fields from more than one relation when you know that users will often request that combination of fields. It is simpler for users to access and display the view than it is for them to repeat a query that accomplishes the same relational join.

You can create views using either an Rdb utility (in which case, DATATRIEVE treats the relations as any other relations in the Rdb database), or DATATRIEVE, or both.

15.4.1 Using Rdb Views

You create a view relation using the RDO utility. After you create a view with the RDO utility, you can access it with the READY DATABASE command, just as you can a simple relation. If you want your view to use a form automatically, define a DATATRIEVE domain for it.

here are advantages to using Rdb views rather than view domains:

DATATRIEVE's response time is faster when you use Rdb views.

You can store or modify records using Rdb views if they contain fields from only one source relation. You cannot store records using DATATRIEVE view domains, even when those views access fields from only one relation.

5.4.2 Defining and Using View Domains

iew domains provide the following advantages:

You can create views that combine fields from an Rdb database with fields from RMS (file-structured) and DBMS domains. The ability to combine fields from different types of databases is a powerful feature of DATATRIEVE view domains.

Your installation might decide that creating an Rdb view does not benefit enough database users to warrant creating one. In this case, you can create a DATATRIEVE view that benefits you personally.

You can refer to DATATRIEVE domain tables with a view domain. This allows you to establish constraints that are not defined for the Rdb database, but that are appropriate for your personal application.

ı such cases, create a view domain with the DEFINE DOMAIN command.

```
EFINE DOMAIN view-path-name OF domain-path-name-1 [,...]  [BY | USING]

    level-number-1 field-name-1     OCCURS FOR rse-1 .

    level-number-2 field-name-2     {OCCURS FOR rse-n | FROM domain-path-name-n}

    .                .                .
    .                .                .
    .                .                .

    [FORM [IS] form-name [IN] form-library]
```

Note

You cannot base view domains directly on relations. You must first define a DATATRIEVE domain for each relation the view accesses.

Refer to the *VAX DATATRIEVE Reference Manual* for a detailed explanation of the arguments and restrictions that apply to view domains.

The following examples define view domains for the PERSONNEL database. The view domain MAILING_INFO uses a subset of fields from the EMPLOYEES domain:

```
DTR> DEFINE DOMAIN MAILING_INFO OF EMPLOYEES
DFN> 01 NAME_AND_ADDRESS OCCURS FOR EMPLOYEES.
DFN>     03 FIRST_NAME         FROM EMPLOYEES.
DFN>     03 MIDDLE_INITIAL     FROM EMPLOYEES.
DFN>     03 LAST_NAME          FROM EMPLOYEES.
DFN>     03 ADDRESS_DATA       FROM EMPLOYEES.
DFN>     03 STREET             FROM EMPLOYEES.
DFN>     03 TOWN               FROM EMPLOYEES.
DFN>     03 STATE              FROM EMPLOYEES.
DFN>     03 ZIP                FROM EMPLOYEES.
DFN> FORM IS MAILFORM IN FORMSLIB;
DTR>
```

The view domain MANAGER_NAMES uses fields from both the EMPLOYEES and DEPARTMENTS domains and a domain table, MANAGERS_TABLE, that is based on current values in the DEPARTMENTS domain. EMPLOYEE_ID is defined as a query name for MANAGER_ID in the DEPARTMENTS domain. MANAGERS_TABLE pairs the query name EMPLOYEE_ID with the associated DEPARTMENT_CODE:

```
DTR> DEFINE DOMAIN MANAGER_NAMES OF DEPARTMENTS, EMPLOYEES USING
DTR> 01 DEPARTMENT OCCURS FOR DEPARTMENTS.
DFN>     03 DEPARTMENT_CODE    FROM DEPARTMENTS.
DFN>     03 DEPARTMENT_NAME    FROM DEPARTMENTS.
DFN>     03 MANAGED_BY OCCURS FOR EMPLOYEES WITH
DFN>                            EMPLOYEE_ID IN MANAGERS_TABLE.
DFN>        06 FIRST_NAME      FROM EMPLOYEES.
DFN>        06 MIDDLE_INITIAL  FROM EMPLOYEES.
DFN>        06 LAST_NAME       FROM EMPLOYEES.
DFN> ;
DTR>
```

You access a view domain by readying it, just as you would a simple domain.

15.5 Displaying Information About Readied Relations and Domains

The SHOW READY and SHOW FIELDS commands provide information about the relations and fields you can access. For example, the following commmands ready the domain SALARY_HISTORY and show the fields you can access in the SALARY_HISTORY relation.

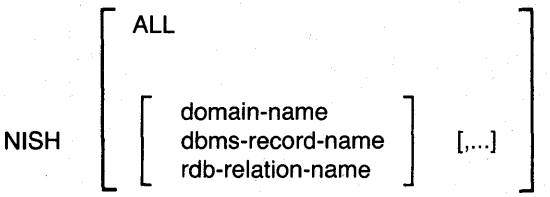
```
TR> READY SALARY_HISTORY
TR> SHOW READY
ady sources:
 SALARY_HISTORY: Domain, Rdb, snapshot
          <CDD$TOP.DEPT29.PERSONNEL.SALARY_HISTORY;1>
 loaded tables.

TR> SHOW FIELDS FOR SALARY_HISTORY
 SALARY_HISTORY
     EMPLOYEE_ID        <Number>
     SALARY_AMOUNT      <Number>
     SALARY_START       <Date>
     SALARY_END         <Date>

TR>
```

5.6 Ending Access to Domains, Relations, and Views

se the FINISH command to end access to the database or databases, or to lected parts of the database:



main-name

the name of a DATATRIEVE domain or view domain.

lation-name

the name of an Rdb relation or view relation.

you do not specify any names in the FINISH command, DATATRIEVE ends cess to everything currently readied.

5.7 Storing and Maintaining Data in an Rdb Database

vo extensions to the DATATRIEVE language apply only when your data is naged by a database management system. These extensions are the COMMIT d ROLLBACK statements. To understand what these statements do, you must derstand the way Rdb stores and updates information in your database and w this differs from storing and updating information in a file-structured tabase.

When you ready a file-structured domain, DATATRIEVE opens the data file associated with the domain and makes any changes to the data file as you enter them. Once the changes are made to the file, you can consider them permanent.

When you ready an Rdb database directly or when you ready Rdb domains, DATATRIEVE makes any changes to the database as you enter them. However, those changes are not permanent until one of the following occurs:

- You enter a COMMIT statement, either interactively or as part of a procedure
- You enter a final FINISH statement for the last readied domain or the last readied relation (depending upon your chosen method of database access)
- You exit from DATATRIEVE

If you decide that you do not want your entries to take effect, you can enter a ROLLBACK statement. When you use the ROLLBACK statement, all the changes made since execution of the last COMMIT or ROLLBACK statement are undone. If neither statement executed, implicitly or explicitly, then the ROLLBACK statement undoes all the changes made since the beginning of your session.

Note that a COMMIT or a ROLLBACK affects all readied databases.

DATATRIEVE executes an implicit COMMIT statement when you finish your last readied domain or relation and when you exit DATATRIEVE. In this case, there is no chance of losing your database modifications because you forgot to enter COMMIT before you finished all your domains or relations, or before you exited DATATRIEVE.

If your system fails, an implicit ROLLBACK executes. In this case, Rdb undoes all changes made to the database since execution of the last COMMIT or ROLLBACK statement. If you open a DATATRIEVE log file at the beginning of your session, it is easy to find out what changes need reentry after a system failure. On a more sophisticated level, Rdb provides journaling and other facilities that help you recover from an accident. In any event, do not rely on reports that were generated before the system failure to determine what changes have been permanently stored.

Once you access data (print, store, or modify it) in a domain or relation, you must enter a COMMIT or ROLLBACK before you can ready the domain or relation again. If, for instance, you want to change the access mode for a relation, you must first issue a COMMIT or ROLLBACK.

f you are working with more than one domain or relation at a time, finishing ne or more, but not all, readied domains or relations has no effect on data ermanence. For instance, assume you ready three domains or relations)NE_DOMAIN, TWO_DOMAIN, and THREE_DOMAIN) and store data in)NE_DOMAIN. You can finish ONE_DOMAIN, but data is not permanently tored until you have explicitly finished or committed all the readied domains)r that database.

nother way of thinking about COMMIT and ROLLBACK statements is to nderstand that they end transactions. When working with Rdb databases, it is nportant to think in terms of *transactions*. Because Rdb gives many users :cess to a database at the same time, it controls their activities to avoid access)nflicts and data inconsistencies. Rdb, therefore, requires each user to identify a nit of database activity, called a transaction.

transaction is an operation on the database that must complete as a unit or not)mplete at all. In DATATRIEVE, a transaction on an Rdb database begins with READY command and ends with either a COMMIT, FINISH, or ROLLBACK. ransactions cannot be nested; they can only be performed consecutively.

ecause you cannot selectively commit or rollback some parts of a transaction ıd not others, it is important to keep transactions short. In addition, you should y to conduct transactions with SHARED access if possible.

he next two sections discuss the COMMIT and ROLLBACK statements in reater detail and present some examples of their use.

5.7.1 Using the COMMIT Statement

he format of the COMMIT statement is:

OMMIT

Then you enter a COMMIT statement, you make permanent all the changes ade to the database since execution of the last COMMIT or ROLLBACK state- ent and release all locks held during the transaction. If neither of these was :ecuted since the beginning of your session, entering COMMIT means that you ish to make permanent all the database changes you have made during your ATATRIEVE session.

ote that when you issue a COMMIT statement, it affects all databases that may : readied, including other Rdb or DBMS databases.

ıe COMMIT statement maintains all collections of Rdb records. When you issue COMMIT statement, Rdb starts a new transaction. The collection will include ımmitted changes other users make to the records in your collection since you rmed the collection.

The following examples illustrate explicit and implicit execution of the COMMIT statement:

- Using an explicit COMMIT statement:

```
DTR> READY PERSONNEL USING JOB_HISTORY WRITE
DTR> STORE JOB_HISTORY
Enter EMPLOYEE_ID: 00166
Enter JOB_CODE: APGM
Enter JOB_START: 11-Nov-1979
Enter JOB_END: 8-Aug-1981
Enter DEPARTMENT_CODE: PRMG
Enter SUPERVISOR_ID: 00319
DTR> COMMIT
```

- Using the FINISH command (implicit COMMIT):

```
DTR> READY PERSONNEL USING JOB_HISTORY WRITE
DTR> REPEAT 2 STORE JOB_HISTORY
Enter EMPLOYEE_ID: 00164
Enter JOB_CODE: DMGR
Enter JOB_START: 9-Sept-1981
Enter JOB_END: 18-Feb-1983
Enter DEPARTMENT_CODE: MBMN
Enter SUPERVISOR_ID: 00359
Enter EMPLOYEE_ID: 12487
Enter JOB_CODE: SPGM
Enter JOB_START: 07-Jul-1980
Enter JOB_END: 9-Sep-1981
Enter DEPARTMENT_CODE: MCBM
Enter SUPERVISOR_ID: 04164
DTR> FINISH
```

- Exiting from DATATRIEVE (implicit COMMIT):

```
DTR> SHOW STORE_JOB_HISTORY
PROCEDURE STORE_JOB_HISTORY
SET ABORT
READY PERSONNEL USING JOB_HISTORY WRITE
DECLARE REC_NUM PIC 999.
REC_NUM = *."number of records you are adding"
REPEAT REC_NUM STORE JOB_HISTORY
DTR> :STORE_JOB_HISTORY
Enter number of records you are adding: 5
Enter DEPARTMENT_CODE: ELGS
                     .
                     .
                     .
DTR> EXIT
```

the first example, one record is permanently stored in the relation
)B_HISTORY following the COMMIT statement.

the second example, the FINISH command does not specify any domain or
lation name and so ends access to the entire database, not just JOB_HISTORY.
therefore results in an implicit COMMIT statement. In this case, two records
e permanently stored in JOB_HISTORY after the FINISH command.

the third example, the EXIT command (or CTRL/Z entered at the DTR>
ompt) implicitly executes a COMMIT statement. The results are the same as in
e second example, except that five records are permanently stored in
)B_HISTORY.

Note

The use of CTRL/Y exits you from DATATRIEVE but signals abnormal termination. When you enter CTRL/Y, Rdb executes a ROLLBACK command. You receive no message from DATATRIEVE, however, to tell you this has been done.

all these examples, the COMMIT statement could affect previous database
anges (those entered after execution of a COMMIT or ROLLBACK statement,
t before the READY commands in the examples). An explicit COMMIT, as in
e first example, maintains any collections of Rdb records. However, an implicit
)MMIT, as in the second and third examples, releases collections of Rdb
:ords.

Note

It is always better to end a transaction explicitly with a COMMIT or ROLLBACK statement than to rely on DATATRIEVE to interpret a statement as an implicit end to a transaction. That way, you can be sure which operations are included in each transaction.

.7.2 Using the ROLLBACK Statement

e format of the ROLLBACK statement is:

)LLBACK

When you enter a ROLLBACK statement, you undo all the changes made to the database since execution of the last COMMIT or ROLLBACK statement. If neither of these was executed since the beginning of your session, entering ROLLBACK means that you wish to undo all the database changes you have made during your DATATRIEVE session.

The ROLLBACK statement releases all collections of Rdb records.

Note that when you issue a ROLLBACK statement, it affects all readied databases, including other Rdb and DBMS databases.

ROLLBACK has the same effect as an ABORT statement. This means that it ca alter the flow of execution of procedures, command files, and nested statements.

If you make a mistake when entering data for one of the records you are storing, you can still use CTRL/Z to keep that record from being stored. When you use CTRL/Z in this way, you affect only the record on which you are working, not an other record entries you might have made.

The following examples illustrate the use of the ROLLBACK statement:

- Using the ROLLBACK statement interactively:

```
DTR> READY DEPARTMENTS WRITE
DTR> REPEAT 2 STORE DEPARTMENTS
Enter DEPARTMENT_CODE: ADMN
Enter DEPARTMENT_NAME: Corporate Administration
Enter MANAGER_ID: 00225
Enter BUDGET_PROJECTED: 50000
Enter BUDGET_ACTUAL: 52000
Enter DEPARTMENT_CODE: ELEL
Enter DEPARTMENT_NAME: Electronics Engineering
Enter MANAGER_ID: 00397
Enter BUDGET_PROJECTED: 140000
Enter BUDGET_ACTUAL: 172000
DTR> ROLLBACK
```

- Entering an ineffective ROLLBACK statement:

```
DTR> READY DEPARTMENTS WRITE
DTR> REPEAT 2 STORE DEPARTMENTS
Enter DEPARTMENT_CODE: ELGS
Enter DEPARTMENT_NAME: Large Systems Engineering
Enter MANAGER_ID: 00369
Enter BUDGET_PROJECTED: 75000
Enter BUDGET_ACTUAL: 72000
Enter DEPARTMENT_CODE: ELMC
Enter DEPARTMENT_NAME: Mechanical Engineering
Enter MANAGER_ID: 00435
Enter BUDGET_PROJECTED: 42000
Enter BUDGET_ACTUAL: 42200
DTR> FINISH
DTR> ROLLBACK
```

Using the ROLLBACK statement in a procedure:

```
DTR> SHOW STORE_DEPARTMENTS
PROCEDURE STORE_DEPARTMENTS
SET ABORT
READY DEPARTMENTS WRITE
DECLARE REC_NUM PIC 999.
DECLARE COMM_OR_ROLL PIC X.
PRINT SKIP
REC_NUM = *."number of records you are adding"
DECLARE VALID_ANSWER PIC X.
VALID_ANSWER = "1"
SET NO ABORT
REPEAT REC_NUM STORE DEPARTMENTS
PRINT SKIP
COMM_OR_ROLL = *."Y if you want the records stored, N if not"
WHILE VALID_ANSWER = "1"
  BEGIN
    CHOICE
      COMM_OR_ROLL = "N", "n" THEN
       BEGIN
        PRINT SKIP, "The record(s) you added will be deleted."
        VALID_ANSWER = "0"
        ROLLBACK
       END
      COMM_OR_ROLL = "Y", "y" THEN
       BEGIN
        PRINT SKIP, "The record(s) you added will be permanently stored."
        VALID_ANSWER = "0"
        COMMIT
       END
      ELSE
       BEGIN
        PRINT SKIP, "Try again....", SKIP
        COMM_OR_ROLL = *."Y if you want the records stored, N if not"
       END
    END_CHOICE
  END
FINISH DEPARTMENTS
PRINT SKIP, "End of access to DEPARTMENTS."
END_PROCEDURE

DTR> :STORE_DEPARTMENTS

Enter number of records you are adding: 5
Enter DEPARTMENT_CODE: MBMF

                              .
                              .
                              .

Enter Y if you want these records stored, N if you don't: N

The record(s) you added will be deleted.

End of access to DEPARTMENTS.

DTR>
```

In the first example, the ROLLBACK statement means that two records are stored in the database, then deleted. A ROLLBACK statement could also undo changes to the database that were entered prior to either STORE statement entry and not noted in this example.

In the second example, a FINISH command that ends access to all readied domains immediately precedes the ROLLBACK statement. Because that FINISH command implicitly executes a COMMIT statement, the two records are permanently stored. ROLLBACK does not do what the user intended.

In the third example, the ROLLBACK statement means that all of the records stored when the REPEAT statement executes are deleted. If the user enters all five records, then those five records are deleted. If the user enters CTRL/Z while storing data for one of the records, then execution of the REPEAT statement stops. The rollback affects, however, many records that have been entered, plus any other database modifications made since execution of the last COMMIT or ROLLBACK statement or since the beginning of the DATATRIEVE session.

Note placement of the SET [NO] ABORT statements in the procedure. If the SET ABORT statement were not in effect, the displays for storing records would appear whether or not WRITE access to the domain or relation were secured. If the SET NO ABORT statement were not in effect, the domain or relation would not be finished. Because the ROLLBACK statement has the same effect as an ABORT statement, the Assignment and PRINT statements associated with the rollback branch of the procedure precede the ROLLBACK statement itself.

If you are designing procedures to be used by people unfamiliar with DATATRIEVE, you probably want to include the FINISH command in your procedure, particularly if a domain or relation has been readied for protected WRITE access. Otherwise, the domain or relation remains locked to all users after your procedure executes. If your procedure includes a ROLLBACK option, however, make sure that the FINISH statement does not execute before the ROLLBACK statement. Otherwise, if the FINISH statement ends user access to the last readied domain (or to all of them at once), a COMMIT statement executes before the ROLLBACK statement does.

15.8 Querying the Database, Writing Reports, and Using Collections

The statements you use for queries and report writing are the same for Rdb relations or domains as they are for other domains.

Here are some reminders if you plan to use collections:

- A COMMIT statement maintains collections of records. (Note, however, that if you ready the source [domain, relation, or database] for that collection with a different access mode, the collection is not maintained).

  The ROLLBACK statement releases any collections that contain Rdb records.

  The FINISH statement releases any collections containing records from the domain, relation, or database being finished.

Make sure you ready domains, relations, or the database with the access you need for the collections you plan to create. If you inadvertently ready the database for READ access, form a large collection, print some records, and then try to modify records, you get an error message. At this point, you must enter COMMIT before you can ready the database for MODIFY access.

When designing procedures that produce reports or displays that include database changes, remember to enter a COMMIT statement before the PRINT or REPORT statement. Otherwise, in the event of a rollback, your report or display will include data changes that were subsequently deleted from the database. Similarly, any query based on assumed database modifications should be preceded by a COMMIT statement.

5.9 Using Rdb's Segmented String Data Type in DATATRIEVE

DATATRIEVE provides limited support for a special Rdb data type called *segmented string.*

Fields defined with the segmented string data type can contain completely unstructured data. Segmented string fields have these special characteristics:

- You can store any type of data in a segmented string field. Segmented strings can contain ASCII text, binary code, Remote Graphics Instruction Set (ReGIS) graphics, or any other data type.

  You do not have to specify the length of data in a segmented string. This makes segmented strings useful for storing data that is arbitrarily long, such as text files or graphic data.

  Rdb does not allocate any storage space for segmented string fields unless you actually store data in the field. This makes segmented string fields a good choice for optional comments or descriptions associated with a record.

Using DATATRIEVE, you can display, modify, and store data in segmented string fields.

The remainder of this section describes:

- Defining segmented string fields in Rdb
- Displaying segmented string fields in DATATRIEVE
- Storing and modifying segmented string fields in DATATRIEVE
- Restrictions and usage notes

15.9.1 Defining Segmented String Fields in Rdb

You cannot use DATATRIEVE to define segmented string fields. To define segmented string fields, use Rdb's RDO utility.

The following example shows how to use RDO to define a segmented string field and a relation that uses it. Once defined, you can use the relation as part of the sample Rdb database, PERSONNEL, created during installation of DATATRIEVE.

The segmented string field RESUME contains the resume for an employee in the PERSONNEL database. The EMPLOYEE_ID field links the RESUME relations with other relations in the PERSONNEL database.

```
$ RUN SYS$SYSTEM:RDO
RDO> SET DICTIONARY CDD$TOP.DTR$LIB.DEMO.RDB
RDO> ! First invoke the sample PERSONNEL database:
RDO> INVOKE DATABASE PATHNAME PERSONNEL
RDO> ! Define a segmented string field called RESUME:
RDO> DEFINE FIELD RESUME
cont>    DATATYPE IS SEGMENTED STRING.
RDO> ! Define a relation using RESUME and EMPLOYEE_ID, which is based
RDO> ! on an already-defined field, ID_NUMBER:
RDO> DEFINE RELATION RESUMES.
cont>    EMPLOYEE_ID
cont>             BASED ON ID_NUMBER.
cont>    RESUME.
cont> END RESUMES RELATION.
RDO> ! Display the fields for the relation RESUMES:
RDO> SHOW FIELDS FOR RESUMES
 Fields for relation  RESUMES
     EMPLOYEE_ID                        text size is  5
       based on global field  ID_NUMBER
     RESUME                             segmented string
                                          segment_length 512
RDO> ! Use the COMMIT statement to store the
RDO> ! new field and relation for PERSONNEL:
RDO> COMMIT
RDO> EXIT
```

15.9.2 Displaying Segmented String Fields in DATATRIEVE

To display data in segmented string fields from within DATATRIEVE, follow these steps:

- Ready the database. For best performance, use only the relations that you need to work with (in this example, the RESUMES relation defined in the preceding section).
- Use DATATRIEVE PRINT or LIST statements to display data in the segmented string field.

The following example illustrates these steps:

```
TR> SET DICTIONARY CDD$TOP.DTR$LIB.DEMO.RDB
TR> SHOW DATABASES
atabases:
        PERSONNEL

TR> ! Ready the database:
TR> READY PERSONNEL USING RESUMES
TR> ! Check that RESUMES is ready:
TR> SHOW READY
eady sources:
  RESUMES:  Relation, Rdb, snapshot read
          <_CDD$TOP.DTR$LIB.DEMO.RDB.PERSONNEL>
o loaded tables.

TR> ! Show the fields for RESUMES:
TR> SHOW FIELDS FOR RESUMES
ESUMES
  RESUMES
     EMPLOYEE_ID        <Character string>
     RESUME     <Segmented string>

TR> PRINT RESUMES WITH EMPLOYEE_ID = 99800

MPLOYEE
  ID
                                        RESUME

99800
                Frank B. Harold
                1492 County Road
                Hicktown, US 54321


BJECTIVE        Junior Lab Technician


)UCATION        B.S. Chemical Engineering, Quinnipiac College, 1983.
                Hicktown Senior High School, 1979
```

(continued on next page)

```
DTR> LIST RESUMES WITH EMPLOYEE_ID = 99800

EMPLOYEE_ID : 99800
RESUME      :
                Frank B. Harold
                1492 County Road
                Hicktown, US 54321


OBJECTIVE       Junior Lab Technician


EDUCATION       B.S. Chemical Engineering, Quinnipiac College, 1983.
                Hicktown Senior High School, 1979

DTR>
```

Note

You cannot use relational operators or the SORTED BY clause in RSEs that refer to segmented string fields. See the section on restrictions and usage notes for more information.

15.9.3 Storing and Modifying Segmented String Fields in DATATRIEVE

The unstructured nature of segmented strings requires different conventions in STORE or MODIFY statements than are used in updating other fields in DATATRIEVE.

In STORE or MODIFY statements that prompt for input, DATATRIEVE repeats the prompt for a segmented string field until you press the TAB key followed by the RETURN key in response to the prompt. Pressing only the RETURN key causes DATATRIEVE to redisplay the prompt for another segment of the field. (Pressing the TAB and RETURN keys after entering text also redisplays the prompt for another segment.)

```
DTR> STORE RESUMES
Enter EMPLOYEE_ID: 23456
Enter RESUME: This is the first line of the RESUME field.<RETURN>
Enter RESUME: This is the second line.<RETURN>
Enter RESUME: To end a segmented string, press the TAB key<RETURN>
Enter RESUME: then the RETURN key at the "Enter" prompt.<RETURN>
Enter RESUME: <TAB><RETURN>
```

```
TR> ! Now print the RESUMES record just stored:
TR> PRINT RESUMES WITH EMPLOYEE_ID = 23456

MPLOYEE
  ID
                                        RESUME

23456
his is the first line of the RESUME field.
his is the second line.
o end a segmented string, press the TAB key
hen the RETURN key at the "Enter" prompt.

TR>
```

ı STORE or MODIFY statements with a USING clause, repeat assignment tatements within a BEGIN-END block for each line of the segmented string eld:

```
TR> MODIFY RESUMES WITH EMPLOYEE_ID = 23456 USING
)N> BEGIN
)N> RESUME = "This example modifies the RESUME field"
)N> RESUME = "of the same record we stored in the "
)N> RESUME = "previous example.  You use as many assignment"
)N> RESUME = "statements as you need.  End the modify or"
)N> RESUME = "store operation with an END statement."
)N> END
TR> PRINT RESUMES WITH EMPLOYEE_ID = 23456

MPLOYEE
  ID
                                        RESUME

23456
is example modifies the RESUME field
 the same record we stored in the
evious example.  You use as many assignment
atements as you need.  End the modify or
ore operation with an END statement.

R>
```

ote that you cannot store or modify only part of a segmented string. Although ou create separate "segments" of the field with STORE or MODIFY statements, you cannot store or modify an individual segment. You must group assignment statements for each line of the segmented string field in the same BEGIN-ND block.

ıe previous examples showed that you can store or modify segmented string lds interactively with DATATRIEVE. You can also create a domain, record, d procedure to simplify storing entire files in a segmented string field.

You can use the following domain and record definitions to store files in the segmented string field RESUMES:

```
DTR> SHOW TEMP_RESUME
DOMAIN TEMP_RESUME ! TEMP_RESUME is a temporary domain used to
                   ! associate the file you want to
                   ! store in the RESUME segmented
                   ! string field with a general
                   ! record definition that divides
                   ! the file into records that
                   ! DATATRIEVE can store.
                   !
        USING TEMP_RESUME_REC ON
        TEMP_RESUME.DAT
                   ! TEMP_RESUME.DAT is the file you want to
                   ! store in the segmented string field.
;

DTR> SHOW TEMP_RESUME_REC
RECORD TEMP_RESUME_REC USING
01 TOP PIC X(255).   ! TEMP_RESUME_REC only has one
                     ! field of 255
                     ! characters (the maximum size
                     ! DATATRIEVE allows for a segment).
                     ! Its only purpose is to separate
                     ! the file you wish to store in a
                     ! segmented string field into
                     ! records.  You can then use
                     ! DATATRIEVE to store the records
                     ! as individual segments in the
                     ! segmented string.
;

DTR>
```

For example, suppose you want to add to the resume stored in the RESUMES record for employee number 99800. The following example displays the resume, then writes it to TEMP_RESUME.DAT:

```
DTR> PRINT RESUMES WITH EMPLOYEE_ID = 99800

EMPLOYEE
   ID
                                        RESUME

 99800
                Frank B. Harold
                1492 County Road
                Hicktown, US 54321


OBJECTIVE       Junior Lab Technician


EDUCATION       B.S. Chemical Engineering, Quinnipiac College, 1983.
                Hicktown Senior High School, 1979

DTR> PRINT RESUME (-) OF RESUMES WITH
DTR>     EMPLOYEE_ID = 99800 ON TEMP_RESUME.DAT
DTR>
```

Exit DATATRIEVE and make any changes you want to TEMP_RESUME.DAT:

```
 ! You have edited TEMP_RESUME.DAT and this is how it looks:
TYPE TEMP_RESUME.DAT
                Frank B. Harold
                1492 County Road
                Hicktown, US 54321


BJECTIVE        Junior Lab Technician


DUCATION        B.S. Chemical Engineering, Quinnipiac College, 1983.
                Hicktown Senior High School, 1979


MPLOYMENT       Chemistry Tutor, Quinnipiac College (9/82 - 5/83)
                Helped freshman chemistry students learn the
                concepts of atomic weights, valence and
                covalence bonding, and empirical formulas

EFERENCES       Available upon request

 ! To store the file TEMP_RESUME.DAT into a segmented string,
 ! you need to know the length of the longest record in the
 ! file.  Use the command ANALYZE/RMS:
ANALYZE/RMS TEMP_RESUME.DAT

ieck RMS File Integrity                    24-JUN-1985 06:28:28.09    Page 1
SER$DISK:[SERLE.RDBDEMO]TEMP_RESUME.DAT;2
        .
        .
        .

IS FILE ATTRIBUTES

        File Organization: sequential
        Record Format: variable
        Record Attributes:    carriage-return
        Maximum Record Size: 255
        Longest Record: 62
        .
        .
        .
```

inally, you can use a procedure, MODIFY_ANY_RESUME, to update the ESUME field for employee number 99800. The procedure:

Readies the RESUMES relation of the PERSONNEL database for MODIFY access.

Readies TEMP_RESUME.

Creates a DATATRIEVE command file called SEGMENT.COM.

- Prints statements in SEGMENT.COM to store the file TEMP_RESUME.DAT in the RESUME segmented string field:

  - A MODIFY USING statement that prompts the user for the employee number of the employee whose resume is to be updated.

  - An assignment statement (RESUME =) for each line of the file TEMP_RESUME.DAT. The clause FORMAT (TOP) USING X(62) in the assignment statement uses the length of the longest record, determined from the ANALYZE/RMS command, for the length of the segments in the segmented string.

- Executes the DATATRIEVE command file SEGMENT.COM.

This example displays the procedure MODIFY_ANY_RESUME and executes it to store TEMP_RESUME.DAT in the RESUME field of employee number 99800:

```
DTR> SHOW MODIFY_ANY_RESUME
PROCEDURE MODIFY_ANY_RESUME
!
! Take the file named in the domain TEMP_RESUME
! and store it in the RESUME field of the record
! specified by the employee number entered by the user.
!
SET COLUMNS_PAGE = 132;
READY PERSONNEL SHARED MODIFY USING RESUMES;
READY TEMP_RESUME;
ON SEGMENT.COM
BEGIN
  PRINT "MODIFY RESUMES WITH EMPLOYEE_ID = *.'employee_id' USING BEGIN";
  FOR TEMP_RESUME
     PRINT "RESUME  ="||"'"||FORMAT (TOP) USING X(62)||"'";
  PRINT "END";
END;
@SEGMENT
COMMIT
FINISH TEMP_RESUME
END_PROCEDURE

DTR> :MODIFY_ANY_RESUME
Enter employee_id: 99800
DTR> PRINT RESUMES WITH EMPLOYEE_ID = 99800
```

```
PLOYEE
 ID
                                        RESUME

9800
                Frank B. Harold
                1492 County Road
                Hicktown, US 54321


JECTIVE         Junior Lab Technician


UCATION         B.S. Chemical Engineering, Quinnipiac College, 1983.
                Hicktown Senior High School, 1979


PLOYMENT        Chemistry Tutor, Quinnipiac College (9/82 - 5/83)
                Helped freshman chemistry students learn the
                concepts of atomic weights, valence and
                covalence bonding, and empirical formulas


?ERENCES        Available upon request

?> COMMIT
?> FINISH
```

e DATATRIEVE command file SEGMENT.COM created by the procedure
ks like this:

```
)IFY RESUMES WITH EMPLOYEE_ID = *.'employee_id' USING BEGIN
;UME  ="                Frank B. Harold"
;UME  ="                1492 County Road"
;UME  ="                Hicktown, US 54321"
;UME  ="                "
;UME  ="                "
;UME  ="OBJECTIVE       Junior Lab Technician"
;UME  ="                "
;UME  ="                "
;UME  ="EDUCATION       B.S. Chemical Engineering, Quinnipiac College, 1983."
;UME  ="                Hicktown Senior High School, 1979"
;UME  =""
;UME  =""
;UME  ="EMPLOYMENT      Chemistry Tutor, Quinnipiac College (9/82 - 5/83)"
;UME  ="                Helped freshman chemistry students learn the"
;UME  ="                concepts of atomic weights, valence and"
;UME  ="                covalence bonding, and empirical formulas"
;UME  =""
;UME  ="REFERENCES      Available upon request"
```

Note

Source files for TEMP_RESUME that contain single quotation marks require special treatment. This is because the procedure MODIFY_ANY_RESUME uses single quotation marks as delimiters for the character string literals stored by the MODIFY USING statement. For the orocedure to work, you must make sure the file contains two consecutive single quotation marks for every one you want stored.

15.9.4 Restrictions and Usage Notes for Segmented String Fields

The folowing restrictions and usage notes concern defining segmented string fields within RDO:

- When you define a segmented string field in RDO with the DATATYPE IS SEGMENTED STRING clause, the SUB_TYPE designation, if any, is ignored by DATATRIEVE.
- You cannot specify a MISSING_VALUE IS clause when you define a segmented string field in RDO. This is an Rdb restriction.
- You cannot specify a DEFAULT_VALUE FOR DTR clause when you defin a segmented string field in RDO. DATATRIEVE gives a warning message when you ready the relation and any such default value is ignored.
- DATATRIEVE does allow the QUERY_HEADER FOR DATATRIEVE an the QUERY_NAME FOR DATATRIEVE clauses when you define a segmented string field in RDO.
- If you use an EDIT_STRING FOR DTR clause when you define a segment string field in RDO, you can only specify a T edit string. If you use any oth type of edit string, DATATRIEVE issues a warning message when you rea the relation containing the field. The length of the T edit string defaults to the current setting of COLUMNS_PAGE. (The default for DATATRIEVE 80.)

  For example, if COLUMNS_PAGE is set to 80, the default edit string for DATATRIEVE is T(80). The T(80) edit string is also used if no EDIT_STRING has been defined in the RDO field definition.

Therefore, the SET COLUMNS_PAGE command in DATATRIEVE can be used to change where a segment line breaks when printed, if no edit string was defined for the field in RDO. (The change takes effect on READY, COMMIT, or ROLLBACK). Changing the EDIT_STRING clause in the Rdb field definition also changes the position where a segment line breaks when printed in DATATRIEVE.

he following restrictions and usage notes concern the use of segmented string elds within DATATRIEVE:

When you ready a domain or relation containing segmented string fields, DATATRIEVE places the segmented string fields last in the relation, regardless of the position of the segmented string when the relation was defined in RDO. If more than one segmented string field is defined in a relation, the segmented string fields are placed last, in the order in which they were defined in the relation.

The SHOW FIELDS command displays the order of the fields within DATATRIEVE and denotes which fields are segmented string fields.

A segmented string field has no data type. DATATRIEVE cannot convert or validate the contents of a segmented string field.

If a segmented string segment is longer than 255 bytes, it is output in subsegments of 255 bytes that break at the COLUMNS_PAGE setting (or at the position in the T edit string, if one is specified in the field definition in RDO). If a segment is shorter than the current COLUMNS_PAGE setting, the segment is left-justified and blank-filled.

When you use a PRINT statement to display a segmented string field, the header for the segmented string field begins on a separate line, following the header lines of fields of other data types. If more than one segmented string field is in the domain, each field has its own header. DATATRIEVE centers the header based on the current COLUMNS_PAGE setting.

Each segment of a segmented string starts in the first column. The T edit string defined for each segmented string field (or the COLUMNS_PAGE setting, if none is defined) controls where a segmented string line breaks. DATATRIEVE displays a blank line for a missing segmented string. Individual segments cannot be displayed separately.

When you use a LIST statement to display a segmented string field, DATATRIEVE prints the segments of the field below the field header, instead of beside it. DATATRIEVE displays a blank line for a missing segmented string.

You cannot use the DISPLAY statement with a segmented string field.

- You cannot store segments larger than 255 bytes in a segmented string field through DATATRIEVE. This is due to a general DATATRIEVE limit on the number of characters that can be input on a line. If this segment exceeds 255, the segment will be truncated.

- You must enter all segments of a segmented string in a single STORE or MODIFY statement. Individual segments cannot be entered or retrieved separately.

  In STORE or MODIFY statements that prompt for input, DATATRIEVE repeats the prompt for each segment to be entered in the segmented string.

  Pressing the TAB key followed by the RETURN key in response to a promp for a segmented string field terminates the segmented string if data has already been entered in response to previous segment prompts. Pressing the TAB key followed by the RETURN key in response to the initial prompt for a segmented string field results in nothing being stored in the field.

- You cannot modify portions of a segmented string field. The MODIFY state ment creates an entirely new segmented string for a record. It does not update an existing segmented string.

- Data can be stored in segmented strings only in response to segment prompts or as character string literals assigned to a segmented string field i a STORE or MODIFY USING statement. You cannot assign values of field: or declared variables to a segmented string field.

  If data is input as a character string literal in a STORE or MODIFY USINC statement, each character string represents one segment. The field name must be repeated for each character string literal being assigned to a segment of the segmented string field. As soon as a new field name is encountered, the string is terminated.

- You cannot retrieve or store segmented string fields from remote domains. When you ready a remote domain, DATATRIEVE ignores the segmented string fields and gives a warning message naming the fields. DATATRIEVE does ready the remote domain, however, and you can retrieve or store fields of other data types.

- Segmented string fields cannot be used with forms or plots.

- You cannot refer to segmented strings with the following elements of a record selection expression:

  - Relational operators
  - Boolean operators (AND, OR, NOT, BUT)

- SORTED BY clause
- REDUCED TO clause
- CROSS clause

5.10 Modifying the Structure of an Rdb Domain or Relation

ou cannot modify the structure of an Rdb relation or view relation using ATATRIEVE. If you want to add, change, or delete fields or indexes, you ust restructure your Rdb database using techniques described in your Rdb cumentation.

5.11 Ensuring Data Security

ke any other domain, each Rdb domain has an associated DATATRIEVE ACL at specifies access privileges. There is also an ACL associated with the database th name. In addition, Rdb provides an access control list for each relation in e database. When a user readies a domain, DATATRIEVE first checks the main's ACL, then checks the database path ACL, and finally checks Rdb access ivileges for the associated relation. Users are denied the requested access if any these access control lists denies them the required privilege.

you want the Rdb access control list to be the main means of access control for lb domains, you might consider opening up the DATATRIEVE ACLs to allow users READ, WRITE, MODIFY, and EXTEND privileges. This allows users ass through" to the Rdb access control list.

you are not using domains to access the database, the ACL associated with the tabase path name and the Rdb access control list are the only means of access ntrol. In this case, you have to maintain only two access control lists.

5.12 Validating Data for Rdb Relations and Domains

ep in mind that when you are storing into or modifying fields whose definitions e common to more than one Rdb relation, you affect values only in the relations at you specify. You do not automatically change data values in the same fields any other relations.

e EMPLOYEE_ID field definition, for example, is common to several Rdb relans in the PERSONNEL database. A user could assign an EMPLOYEE_ID lue for Jack Jones in the EMPLOYEES relations and give him a different MPLOYEE_ID value in the JOB_HISTORY relation. You should design valida-n procedures to protect against such an occurrence. Design your database so at the minimum number of identical fields exist from one relation to another. mit common fields to the keys you need in order to match records.

One way to check data for validity is through Rdb's VALID IF clause of the Rdb DEFINE FIELD statement. In addition, with Rdb/VMS, you can use the DEFINE CONSTRAINT statement, which is more flexible for checking validity. (Note that one option of the DEFINE CONSTRAINT statement specifies that the validation criteria are not evaluated until a COMMIT statement is issued. If a database is set up to check constraints at the time a COMMIT statement executes, validation errors can occur later than interactive DATATRIEVE users expect to receive them.)

15.13 Optimizing Performance

To optimize DATATRIEVE performance, keep these points in mind when using DATATRIEVE with Rdb:

- Avoid using the FIND statement unless the resulting collection contains a small number of records. After you form a collection, DATATRIEVE cannot use the Rdb index structure to search the data contained in the collection. Data retrieval using keyed fields to search relations is faster than an exhaustive search of large collections. For the same reason, forming a new collection from another collection is likely to be time-consuming. If a collection contains a small number of records, however, you may find that DATATRIEVE responds more quickly when you specify that collection as a record source.

- Using the READY DATABASE command to access your Rdb database works more quickly than using domains defined for relations. Therefore, define domains for relations only when you cannot do what you want by readying the entire database directly.

- DATATRIEVE and Rdb use different default settings for waiting on locked records. Using the DATATRIEVE default may cause Rdb to generate error messages you do not expect. In DATATRIEVE, when your Rdb transaction encounters a locked record, the default setting (SET NO LOCK_WAIT) causes Rdb to generate an error message and returns to the DATATRIEVE prompt. In Rdb, the default setting (START_TRANSACTION WAIT) is for transaction to wait until a locked record is released and then continue the operation.

  The following example shows the error message generated when a user tries to update a locked record of the sample EMPLOYEES Rdb domain with the default DATATRIEVE setting in effect:

```
DTR> MODIFY FIRST_NAME
Enter FIRST_NAME: Norman
%RDB-E-LOCK_CONFLICT, NO WAIT request failed because resource was l
-RDMS-F-LCKCNFLCT, lock conflict on area 25
DTR>
```

To change the setting in DATATRIEVE, issue the SET LOCK_WAIT command. Instead of generating the error messages and returning to the DATATRIEVE prompt, your Rdb transaction will wait until the other user begins another transaction and the record is released. (DATATRIEVE commands and statements that begin Rdb transactions are READY, COMMIT, and ROLLBACK.)

Changing the default setting to LOCK_WAIT will affect performance in the sense that your transaction must wait for a locked record to be released.

Accessing Remote Data 16

This chapter explains how to define network domains and access distributed domains.

16.1 Defining Network Domains and Accessing Remote Domains

With VAX DATATRIEVE, you can access domains defined on other systems linked to yours by the DECnet network. Each system must have DATATRIEVE installed. In the following discussion, the term "local" refers to your system and the term "remote" refers to a system connected to yours by the DECnet network.

The term "network domain" refers to the domain you define at your local node containing a network address. The term "remote domain" or "distributed domain" refers to the domain located at the remote node.

To access a remote domain, you must tell DATATRIEVE the network address of the remote domain. You can do that in one of two ways:

You can include the network address of the remote domain in the READY command:

```
DTR> READY CDD$TOP.DEPT32.SMITH.PERSONNEL AT BIGVAX
DTR>
```

At your local node, you can define a domain (called a network domain) that contains the address of the remote domain. Then you ready the network domain at your local node just as you would any domain definition:

```
DTR> DEFINE DOMAIN REM_PERSONNEL
DFN>    USING CDD$TOP.DEPT32.SMITH.PERSONNEL
DFN>    AT BIGVAX"SMITH PASSMETHROUGH";
DTR> READY REM_PERSONNEL
```

When you ready a remote domain, either directly using the network address in the READY command or by readying a local domain that contains a network address, DATATRIEVE:

- Recognizes that the desired domain resides at another node in your network
- Starts a process on that remote node
- Invokes the DATATRIEVE Distributed Data Manipulation Facility (DDMF) at the remote node to process the DATATRIEVE statements that refer to the domain at that node
- Terminates the remote process when you finish the domain

The DDMF keeps a trace file of your requests and its responses. It writes this file to the login directory of the remote process. If the remote node is a VAX/VMS system, the trace file is NETSERVER.LOG. If the remote node is a PDP-11 system, the trace file is DDMF.LOG.

If the DDMF is handling more than one domain, the remote process ends when you finish the last domain.

The following sections explain the process of defining network domains and how to access remote domains.

16.1.1 Defining Network Domains

To define a network domain, you define a DATATRIEVE domain at the local node that specifies the link with the domain definition at the remote node.

The following example defines a network domain for a domain on a remote VAX/VMS system:

```
DTR> DEFINE DOMAIN PERSONNEL
DFN>    USING CDD$TOP.DTR$USERS.CUVERDALE.PERSONNEL
DFN>    AT BIGVAX"CUVERDALE SESAME";
DTR>
```

The following example defines a network domain for a domain on a remote PDP-11 system:

```
DTR> DEFINE DOMAIN CDD$TOP.DEPT39.PERSONNEL USING PERSONNEL
DFN>    AT ELEVEN"*.USERNAME *.PASSWORD";
DTR>
```

'ou use the following format:

)EFINE DOMAIN path-name USING <------- (1)

remote-domain-name <---------- (2)

AT node-spec <---------- (3)

ee the *VAX DATATRIEVE Reference Manual* for the full syntax.

'he format:

1. Specifies a name for the network domain

   The path name you give the network domain can be a given name, a relative path name, or a full path name.

   The results of specifying each type of name and the consequences for extracting and moving the definition are the same for network domains as for any other domain.

2. Specifies the name of the domain at the remote node

   If the remote domain is on a VAX/VMS system, you can specify the remote domain name using either the given name or the full path name of that domain.

   If you use the full path name, your access to the remote domain is independent of the default dictionary directory used by the remote process running DATATRIEVE. If you use the given name, you access either the CDD default dictionary for that process, or a CDD dictionary identified in the login command file of the remote process that runs DATATRIEVE.

   If the remote domain is on a PDP-11 system, you specify only the domain's given name. Otherwise, the format for referring to remote domains is the same for both the VAX and the PDP-11 systems.

   Note that the remote domain you refer to in a network domain definition is no different from any other DATATRIEVE domain. It specifies the relationship of a particular record definition and a data file. (At remote VAX/VMS systems, a DBMS or Rdb domain can also be the access path to a DBMS or Rdb database.) The remote domain must be defined at the remote node during a DATATRIEVE session running on that remote node. A person or program local to that system can invoke DATATRIEVE to enter the domain definition, or a person or program running on that system as a remote terminal can enter the definition.

3. Specifies the network address

   The network address corresponds to node-spec in the preceding format.

   If the login procedure used by the remote process does not supply the necessary login information (user name, password, and, optionally, account name), either the person readying the network domain or the network domain definition must supply this information.

   You can use any of the following formats to specify the network address and to provide the best level of access security for your installation:

   - node-name"username password [account-name]"

     Examples of this format are:

     ```
     BIGVAX"WARTON KNOCKKNOCK DEPT32"

     ELEVEN"LINTE LETMEIN"
     ```

     When you specify the network address using this format, users do not have to supply login information when readying the network domain.

   - node-name"*.username-prompt *.password-prompt [*.account- promp

     Examples of this format are:

     ```
     WINKEN"*.USERNAME *.PASSWORD *.ACCOUNT"

     VAXTWO"*.'user name' *.'password'"

     PDPTWO"*.'user name' *.'password'"
     ```

     When you specify the network address using this format, users are prompted for login information when they ready the network domain. This method provides the best security.

   - node-name

     Two examples of this format are:

     ```
     BIGVAX

     ELEVEN
     ```

     When you specify the network address with this format, the account used by the remote process must provide login information automatically.

If you prefer, you can combine elements from the first two formats. For example, you can explicitly specify the user name and specify a prompting value expression for the password:

```
SNOOPY"CLARK *.PASSWORD"
```

6.1.2 Accessing Remote Domains

s you have seen, you can access a remote domain by either:

Readying a network domain (the domains you learned to define in the previous section)

Including the network address in a READY command

he next two sections show how to access a remote domain both ways.

5.1.2.1 Readying a Network Domain -- Readying a network domain makes stributed processing transparent to the DATATRIEVE user. Depending upon ow you specify login requirements in the network address of the domain defini-on, the user may have to enter a user name or password. The user need not be ncerned with the actual location of the domain, however, as that is already efined in the network domain definition.

ie following example readies the network domain REM_PERSONNEL, whose finition explicitly specifies user name and password in the network address:

```
R> READY REM_PERSONNEL

R>
R> SHOW REM_PERSONNEL
MAIN REM_PERSONNEL USING CDD$TOP.DEPT32.SMITH.PERSONNEL
 BIGVAX"SMITH PASSMETHROUGH";
```

ie following example readies the network domain REM_SALES whose definin specifies prompting value expressions for user name and password in the netrk address:

```
R> READY REM_SALES
ter USERNAME: GREEB
ter PASSWORD:
R>
R> SHOW REM_SALES
MAIN REM_SALES USING SALES AT ANODE"*.USERNAME *.PASSWORD";
R>
```

16.1.2.2 Readying a Remote Domain Directly -- When you ready a remote domain directly, you specify the name of the remote domain and its network address in the READY command. The formats and results for specifying a network address in the READY command are the same as those for including the network address in a network domain definition. See the third step, specifying a network address, in the section on defining network domains.

The following example readies a remote domain on a VAX/VMS system. The command specifies a full dictionary path name and assumes a default DECnet account in the network address:

```
DTR> READY CDD$TOP.DEPT32.PERSONNEL AT BIGVAX
DTR>
```

The next example readies a remote domain on a PDP-11 system. The network address specifies the login account assigned to the user VOJTEK. The prompting value expression for the password ensures that the password is not displayed on the terminal as it is entered:

```
DTR> READY PERSONNEL AT ELEVEN"VOJTEK *.PASSWORD"
DTR> Enter PASSWORD:
DTR>
```

16.1.3 Results of Accessing Remote Domains

There are some facts common to both ways of accessing remote domains.

Whichever method you choose, the remote process running DATATRIEVE executes a login command file, just as you do when you log in to your local system. Depending in how you specify a network address, the remote process can log in to a specific account or it can log in to a default DECnet account.

For example, the command READY PERSONNEL AT BIGVAX does not specify a user name. Therefore, it starts a remote process using a default DECnet account. (The guide for installing DATATRIEVE explains how to set up a default DECnet account for DATATRIEVE on a VAX/VMS system.) In this case, the login procedure executes the login command file for the default DECnet account.

On the other hand, the command READY PERSONNEL AT BIGVAX"SWAZY ITSME" starts a remote process using the login account assigned to user Swazy The login procedure executes the login command file for Swazy's account.

In addition to providing security information such as user name and password, you must make sure the remote process uses the correct dictionary and system directories when it invokes DATATRIEVE and readies the domain for you.

)n a VAX/VMS system, the login command file for the account used by the emote process can include commands that set any defaults not included in the ıetwork address and needed by the remote process running DATATRIEVE. Iowever, the login file should not execute commands that are appropriate for ıteractive mode and that might cause a network process to fail. (Assignments to 'T: can fall into this category.)

'he following example illustrates some helpful commands that you might want to ıclude in a login file for an account on a remote VAX/VMS system. The example ssumes that there are no commands preceding the first line that might cause a etwork process to fail.

```
IF F$MODE() .EQS. "NETWORK" THEN GOTO NETWORK_PROCESS

                .
                .
                .

NETWORK_PROCESS:
SET DEFAULT [HAMOND.PERSONNEL.DATA]
ASSIGN  "CDD$TOP.HAMOND.PERSONNEL"  CDD$DEFAULT
EXIT
```

he SET DEFAULT command moves the network process to the VAX/VMS irectory that contains the data file. This is necessary if the login directory does ot contain the data file and if the definition of the domain being readied does not ɔntain a full file specification for the data file. The ASSIGN command sets the ıgical name CDD$DEFAULT to the dictionary containing the definition of the ɔmain being readied. This is necessary if this information is not included in the ɛtwork address. The EXIT command exits the login command file so that subse- ıent commands inappropriate for a network process are not executed.

or a PDP-11 system, you can use a SET DICTIONARY command in a ATATRIEVE-11 QUERY.INI file to access the dictionary directory you ant to use when the remote process invokes DATATRIEVE. This is unneces- ıry, of course, if the dictionary file containing the domain definition is in the gin directory.

ɔ.1.4 Restrictions on Using Remote Domains

'hen you access data located on remote systems, note the following restrictions:

- Using a simple DATATRIEVE domain that includes a remote node name in the specification for the data file can slow DATATRIEVE response time. Avoid using this method to access data on other systems. Instead, use the

methods explained in this chapter to ensure optimum DATATRIEVE performance.

- The CONTAINING operator does not work with a prompt for remote domains. For example, the following PRINT statement does not work:

```
PRINT REMOTE_YACHTS WITH BUILDER CONTAINING *.BUILDER
```

  DATATRIEVE uses a fixed-length field to transmit the prompt value to the remote server. Trailing spaces are appended to any value shorter than the length of this field.

- You cannot use a Boolean expression containing the relational operators IN or ANY in a record selection expression with a remote domain or collection as its source.

- You can use remote domains or collections in a CROSS clause only if both of the following conditions are met:

  - The remote domains or collections reside on the same remote node.
  - You access the remote domains or collections with the same account, user name, and password.

- When validation checks are made for data entered in response to a prompt, distributed DATATRIEVE does not reprompt when validation errors occur. A validation error causes the statement being processed to abort.

- In a record selection expression with a remote domain as its source, you cannot use value expressions that require computations involving remote data.

- If a remote domain contains an elementary field that is also a list field, explicitly attempting to access it generates the error message:

```
[DDMF] field is not a list.
```

  This occurs if the field is defined using the following format:

```
06 ABSTRACT PIC X(80) OCCURS 0 TO 10 TIMES.
```

In this case, you need to change the format of the field definition. If the field is defined in the following way, you can explicitly access the field TEXT:

```
06 ABSTRACT OCCURS 0 TO 10 TIMES.
   08 TEXT PIC X(80).
```

You cannot ready a remote Rdb database by specifying a database path name in a READY command that uses the AT syntax. You must first create domain definitions on the remote node for each relation you want to access. You can then ready the database using these domains.

The following example does not work:

```
DTR> READY CDD$TOP.DTR$LIB.DEMO.RDB.PERSONNEL AT
[Looking for Node Specification]
CON> DEPT42"SMITH PASSWORD"
[DDMF] You can not READY a database with an alias.
[DDMF] Statement abandoned due to error.
```

If you define a DATATRIEVE domain on the remote node for each relation in PERSONNEL, you can ready and access each relation in the remote Rdb database PERSONNEL. In this example, the user defines a DATATRIEVE domain on the remote node DEPT42 for the Rdb relation SALARY_HISTORY in the database PERSONNEL.

```
DTR> DEFINE DOMAIN SALARY_HISTORY USING SALARY_HISTORY OF
DFN> DATABASE CDD$TOP.DTR$LIB.DEMO.RDB.PERSONNEL;
DTR>
```

Then the user can ready the remote domain from the local node:

```
DTR> READY SALARY_HISTORY AT DEPT42"SMITH PASSWORD"
```

Name Recognition and Single Record Context A

'hen you use a field name as a value expression and you display, modify, or erase ıe or more records, DATATRIEVE determines exactly which record or records e the targets of the action you propose.

ɔr each of these actions, DATATRIEVE must first determine the context within ıich the action occurs. The context is the set of conditions that govern the way ATATRIEVE recognizes field names and determines which records are the tar-ɪts of DATATRIEVE statements. Understanding the way DATATRIEVE man-ɣes context is especially important when you begin nesting DATATRIEVE atements.

.1 Establishing the Context for Name Recognition

ATATRIEVE does not require that every field name be unique. You can use the me name in several record definitions. You can even use the same name several nes in the same record definition, as long as the fields with identical names do t have the same level number in one group field.

ɪr example, both the YACHTS and OWNERS domains have group fields named 'PE, and both group fields contain elementary fields you can refer to with the mes BUILDER and MODEL. (In YACHTS, DATATRIEVE recognizes the ery name BUILDER as equivalent to MANUFACTURER. Other query names · YACHTS are SPECS, LOA, and DISP.) Figure A-1 shows the fields in both mains and points out the duplicate names.

hen you work with several record streams from the same domain, the field mes in all record streams are identical. Whether you form collections or record eams of records from the YACHTS domain, DATATRIEVE has a mechanism · identifying which record to act on when you want to retrieve or change data ɪm only one field of one record.

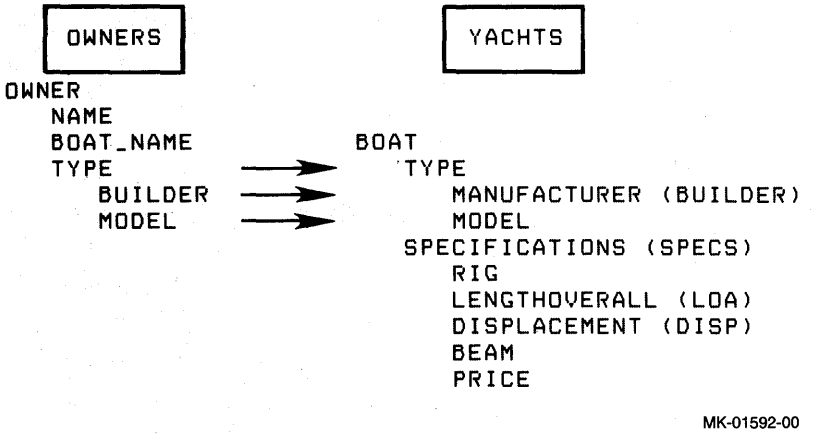


Figure A-1: Duplicate Field Names in YACHTS and OWNERS

When you understand the way DATATRIEVE establishes the context for recognizing names, you can use the names of domains, fields, collections, and variables to form the simple and the complex relationships DATATRIEVE provides. One of the keys to mastering the use of context is understanding the two DATATRIEVE context stacks.

A.1.1 The Right Context Stack

When you issue a statement, DATATRIEVE builds a **context stack**, a linked list that controls DATATRIEVE's search for names to match the ones you use in statements. The context stack consists of context blocks, or lists of names. These context blocks are linked together by pointers that control the sequence of DATATRIEVE's search for values to associate with the names you use in statements.

DATATRIEVE searches the right context stack for values to associate with names you use in print lists, Boolean expressions, and the right side of assignment statements such as x = y. See Section A.1.3 for a discussion of the left context stack.

A.1.1.1 The Content of a Context Block -- When you use a record selection expression, DATATRIEVE creates a context block to establish a context for name recognition. That context block contains, among other things, a list of names.

At the top of the list is a slot for the name of a context variable (see Section A.1.2.1). Next is the name of the domain referred to in the record selection expression. The rest of the list contains the names of fields in the record associated with that domain. Those field names are arranged according to the field tre associated with that record.

'he field tree contains the names of all the group fields, elementary fields, :OMPUTED BY fields, REDEFINES fields, and lists in the record and pre- erves the hierarchical relationships among them.

/hen DATATRIEVE searches for a name in the context stack, it looks for a alue to associate with that name. The search ends, and DATATRIEVE takes ıe associated value when it finds the first name that matches the one in your tatement.

. DATATRIEVE name can consist of several names joined together. (See Section ..1.2.2.) They resemble dictionary path names in form and function. To be recog- ized, these compound or qualified names you supply must represent a valid path ırough the hierarchy of a context block and the field tree it contains.

/hen DATATRIEVE encounters a name, it begins its search in the context block ı top of the stack. DATATRIEVE first looks at the slot in the context block :served for a collection name or the name of a context variable. For unnamed URRENT collections, this slot contains the name CURRENT. For named URRENT collections, the name CURRENT and the collection name are equiv- ent. Named collections that are not the CURRENT collection have the collec- on name in this slot.

the top block on the context stack refers to a record stream, this slot is empty ıless you use a context variable in the RSE that forms the record stream. The ıntext variable gives a record stream a temporary name; this name fills the first ot in the context block for these named record streams.

DATATRIEVE finds that the first segment of a qualified name matches the ıme in the collection name/context variable slot, it continues its search in that ock for a match for the rest of the name. If the name in your statement does not atch the name in the collection name/context variable slot, or if that slot is npty, DATATRIEVE continues to look through the first context block to find a atch.

ext in the context block is the name of the source of the records referred to by at block. For collections and record streams, that source can be the domain ıme or the name of a list for hierarchical records. The source can also be the ıme of a collection if you use the collection as the basis for a record stream in a)R statement and use a context variable.

the source name does not match the name in your statement, DATATRIEVE :xt looks for the name in the slot reserved for names.

ext DATATRIEVE looks at the name of the top-level (the 01 level) field name. no match occurs, DATATRIEVE looks at each succeeding field name in the der they are displayed when you enter a SHOW FIELDS command. That order n take you through the entire hierarchy of the field tree, traversing first the left anch and then the right wherever there is a branching point in the hierarchy.

If DATATRIEVE finds no match in the first block on the context stack, it goes t the next context block on the stack and begins its search there.

DATATRIEVE stops its search as soon as it finds an exact match for the name i your statement. Then it associates the value assigned to the name on the context stack with the name of the field in your statement.

If DATATRIEVE finds no match for the name in any of the context blocks, it dis plays a message on your terminal that the field name is either undefined or used out of context. The only remedies are to change the context so that the name in your statement resolves properly or to remove any ambiguity by qualifying the name further with group field names or context variables.

For the sake of clarity, the following description of the various types of context blocks starts with the bottom of the context stack, that is, with the context block that DATATRIEVE checks last.

A.1.1.2 Global Variables -- The bottom context block contains the names of any global variables you have established and have not released. This block is different from the others on the stack because its content is not determined by a record selection expression. Nevertheless, DATATRIEVE treats the name of a global variable as though it were the name of a field in a simple record. Just as DATATRIEVE associates the value of a field with the field name, DATATRIEVI associates the value of a global variable with its name.

DATATRIEVE looks at the global variables last when trying to find a name to match one in your statement. No two global variables can have the same name. When you issue a DECLARE statement at command level (indicated by the DTR> prompt), DATATRIEVE checks the names of the global variables you have declared. If it finds one with the same name, it releases the old variable and its value and replaces it with the new one. DATATRIEVE initializes the new variable with a default value, a missing value, a zero, or a space depending on the clauses you include in the DECLARE statement.

A.1.1.3 Collections -- The next higher set of blocks in the context stack refer to existing collections. Each collection with a block on the context stack must have one record singled out as a selected record. Although a collection can have a number of records in it, only one of those records can be used in the search for the context of a name. DATATRIEVE can assign only one value to the name. Consequently, that one value can come from only one of the records in the collection.

Remember, the reason for resolving the context of a name you use in a statemer is to assign to the name a value for use in the statement.

or an existing collection, you can designate one record at a time as the selected ecord for that collection. The SELECT statement lets you designate the selected ecord in a collection by relative reference (FIRST, NEXT, PRIOR, LAST, and /ITH Boolean) or by absolute reference to the position number of the record in ıe collection. A collection has a block on the context stack only if it has a elected record.

you have more than one existing collection with a selected record, the block nmediately above the one for global variables refers to a named collection with a elected record. That collection is the one you formed with a FIND statement efore you formed any of the other collections that have selected records.

he rest of the context blocks for the collections with selected records are ordered cording to the sequence in which you formed them, not the order in which you ıtered the SELECT statements to establish the selected records.

the CURRENT collection has a selected record, the context stack contains a ock referring to the CURRENT collection. That block is above the blocks of all her collections; that is, DATATRIEVE searches for names in the context block the CURRENT collection before it searches the context block of any other llection.

he key to understanding the way DATATRIEVE recognizes names is that cept for the global variables, the context stack is ordered on a "last-in, first-ıt" basis. The most recently formed context block is the one DATATRIEVE arches first.

ou do not have to rely on your memory to recall the order in which you ur existing collections. You need only issue a SHOW COLLECTIONS com-and. DATATRIEVE displays the most recently formed collection (always the URRENT collection, whether it has a name or not) at the top of the list and the ldest" one at the bottom.

ith the SHOW collection-name command, you can inspect each existing collec-n to see how many records are in the collection, whether it has a selected cord, and, if it does, what the position number of the selected record is in the llection.

DATATRIEVE searches the context stack and does not find a match for the ıme in your statement, it displays an error message that may seem puzzling ıless you understand the way DATATRIEVE forms the context stack:

```
eld "name" is undefined or used out of context
```

ou may know the name has been defined and that it is the name of a field in a cord associated with one or more existing collections. If, however, none of the llections containing that field have selected records, DATATRIEVE cannot tell the field is defined or not.

If a collection containing the named field has no selected record, that collection has no block on the context stack. Consequently, DATATRIEVE neither finds a match for the field name nor has a way of discovering from the search of the context stack if the field name is defined at all.

The order of context blocks at the higher levels of the context stack depends on the order in which DATATRIEVE encounters the elements containing names associated with values. The order of the following sections does not imply any relative position on the stack. Only the order DATATRIEVE encounters those elements determines their order on the stack.

A.1.1.4 Record Streams -- Before DATATRIEVE looks at the context block o the most recently formed collection with a selected record, it looks at the context blocks created explicitly in the statement. One type of context block created by a statement refers to the field names of a record stream formed by a statement.

Context blocks of record streams act differently from those of collections. The context block for a collection stays on the stack as long as the collection has a selected record. The context block of a collection is removed from the stack only you release the collection or remove its selected record with a DROP statement.

The context block for a record stream, however, stays on the stack only as long a the statement containing it is being executed. When DATATRIEVE finishes processing the statement, the block added to the stack is removed from the context stack and is not available when DATATRIEVE rebuilds the stack after it encounters the next statement.

Only three statements make lasting changes to the context stack:

- FIND

  The FIND statement can remove the CURRENT collection from the contex stack by forming a new CURRENT collection. The new CURRENT collection releases the old collection but does not put a block on the context stack because a newly formed collection has no selected record.

- SELECT

  The SELECT statement puts a collection on the context stack by establishing a selected record. SELECT cannot change the relative order of collections on the stack. That order is determined by the relative order in which you formed the collections with the FIND statement.

DROP

The DROP statement removes collections from the context stack by dropping the selected record from the collection. The SHOW collection-name command still notes the position number of the previously selected record, but a parenthetical note points out that the record has been dropped:

```
DTR> SHOW CURRENT
Collection CURRENT
    Domain: YACHTS
    Number of Records: 113
    Selected Record: 57 (Dropped)
DTR>
```

The selected record has been removed from the collection, and cannot be retrieved unless you form a new collection that contains it.

hese three statements, however, share a restriction that separates them from all ;her statements: you cannot use FIND, SELECT, or DROP statements in com-)und statements. They must be entered at command level by themselves. urthermore, these statements do not form temporary record streams; they 'fect only collections.

ou can, however, have several context blocks for record streams on the context ack at one time. The block for a record stream stays on the context stack until ATATRIEVE finishes the statement. Because you can nest statements in FOR ops, BEGIN-END blocks, IF-THEN-ELSE statements, THEN, and WHILE atements, the inner statements can form record streams before DATATRIEVE ıishes the outermost statement.

ATATRIEVE has to keep the context of outer statements separate from that of ner ones. It keeps them separate by putting a block on the context stack when encounters an element that requires one. DATATRIEVE begins processing ımpound statements with the outermost statement and works progressively ward the innermost one. The context blocks it forms for elements in the inner-ost statement are at the top of the stack when the innermost statement is being ocessed.

hen DATATRIEVE finishes processing the innermost statements, it removes e blocks created by that statement. DATATRIEVE works its way back out ward the outermost statement, removing blocks created by statements as soon it finishes processing the statement.

)r example, in the case of nested FOR loops, the context block for the inner-)st FOR loop is higher in the stack than the blocks for the outer loops.

When DATATRIEVE completes the execution of the innermost loop, it removes the context block of that FOR statement, leaving those of the outer FOR statements on the stack. As DATATRIEVE completes each loop, the context block for that loop is removed from the stack. This same pattern of events applies to statements in BEGIN-END blocks.

When a statement that forms a record stream is followed by a second statement that is not contained in the first, DATATRIEVE removes the context block created for the first statement from the stack and puts a context block for the second statement in its place.

For example, in a BEGIN-END block, one PRINT statement containing an OF rse clause follows another. The context block of the first statement is in effect only during the execution of that first statement. That block is replaced by the one for the second PRINT statement when DATATRIEVE begins processing the second statement.

DATATRIEVE handles the context block of a FOR loop the same way it handles statements containing an OF rse clause.

DATATRIEVE creates four other types of context bocks that affect the order of the context stack: those for local variables, VERIFY clauses, VALID IF clauses, and context variables.

A.1.1.5 Local Variables -- Local variables are variables defined in compound statements. A local variable and its effect on the context stack last only from the DECLARE statement that defines it until DATATRIEVE completes the execution of the statement containing the DECLARE statement.

A.1.1.6 VERIFY Clause in the STORE Statement -- Like the context for local variables, the context for resolving field names in a VERIFY clause of the STORE statement is short-lived. The STORE statement does not access or change any existing record. Consequently, for each STORE statement DATATRIEVE creates a context block to associate the field names with the values in the new record. DATATRIEVE executes the VERIFY clause after you have assigned values to all the fields prescribed by the syntax of the statement, but before DATATRIEVE stores the record in the data file.

A.1.1.7 VALID IF Clause in a Record Definition -- When you assign a value to a field name in either a STORE or MODIFY statement, DATATRIEVE looks in the appropriate record definition for a VALID IF clause. If the value is unacceptable according to the conditions specified in the VALID IF clause, DATATRIEVE displays a message on your terminal and reprompts you for an acceptable value. It uses the same context to associate the field name with your response to the reprompt.

'he context for resolving field names in the VALID IF clause is established in one f two ways:

By the context block set up for the STORE statement

By the context block set up for the the MODIFY statement

ı either case, the value associated with the field name is the one just assigned to by your response to a prompt or by an assignment statement in the USING ause of the STORE or MODIFY statement.

'ATATRIEVE executes the VERIFY clause only after the values you assign ıeet the conditions of VALID IF clauses in the record definition. As a result, ıere can be no conflict between the context established for these two clauses. he context for the VALID IF clause no longer exists when DATATRIEVE ɛecutes the VERIFY clause.

.1.2 Using Context Variables and Qualified Field Names

he ways of establishing context discussed to this point deal with resolving the ɔnnections between names and values by finding the first instance of a valid field ame or variable name. When several context blocks on the stack contain fields ith the same names, you need a way to skip over some instances of the name to ɛt to the field that contains the value you want to retrieve.

ATATRIEVE gives you two methods of forcing name recognition: context ıriables and qualified field names. Although they require different actions from ɔu, these two methods have an underlying similarity.

.1.2.1 Context Variables as Field Name Qualifiers -- A context variable is a ımmy variable specified in a record selection expression for the purpose of name cognition. When DATATRIEVE encounters a context variable, it puts a new ock on the context stack. That new block connects the name of the context vari- ɔle with the field names and values of the records identified by the record selec- ɔn expression.

ıe context established by the context variable lasts until DATATRIEVE com- ɛtes the execution of the statement containing the record selection expression which the context variable occurs. However, that context does not affect any ter loops or nesting statements that contain the statement in which you use the ntext variable.

context variable, however, does affect all inner statements nested in the state- ɛnt that contains the record selection expression in which the context variable curs.

You can use the context variable as a prefix for each field name of the records identified by the record selection expression. Citing a field name with a context variable prefix can make a field name unique, even when the domains and field trees of a record in a record stream are identical.

Putting a prefix on a field name produces a qualified field name. The context vari able must be the first prefix added to a field name.

A.1.2.2 Other Field Name Qualifiers -- Using other qualifiers as prefixes to field names is the second method of overriding DATATRIEVE's default mecha- nism of name recognition.

Although DATATRIEVE does not require that each field name be unique, each fully qualified field name must be unique. The fully qualified field name consists c the domain name, the top-level group field name, the names of any group field to which the elementary field belongs, and the elementary field name. You must sep arate each element of the fully qualified name from the next with a period.

For example, in the domain YACHTS, the fully qualified field name of MODEL is:

```
YACHTS.BOAT.TYPE.MODEL
```

You can use these elements in any combination that preserves their hierarchical order to distinguish the MODEL field in YACHTS from the MODEL field in another domain, such as OWNERS.

When DATATRIEVE encounters a qualified field name, it searches the context stack for the first match of the name you specify. For example, if you use BOAT.MODEL in a record selection expression, DATATRIEVE searches the context stack for the first valid occurrence of the name BOAT and searches the branches of the hierarchy under BOAT for the first valid occurrence of the name MODEL.

The success of the search is not jeopardized because you omit the group field name TYPE from the qualified name of MODEL. DATATRIEVE searches the entire hierarchy under BOAT until it finds the first valid occurrence of TYPE. When an intermediary group field name is omitted, DATATRIEVE searches the hierarchy according to the order in which the fields of the record were defined.

Fully qualified field names are adequate when working with two or more domain that share elementary or group field names, or both. However, when you work with two record streams from the same domain, you must further qualify the fie name with a context variable. This extra qualification is especially necessary wh dealing with lists in hierarchical records.

uppose you want to display information about all builders who build boats with
ıore than one type of rig. YACHT is the given name of the record associated
ith the domain YACHTS. The field tree of YACHT has the structure:

```
ACHTS
  01 BOAT
    03 TYPE
       06 MANUFACTURER
       06 MODEL
    03 SPECIFICATIONS
       06 RIG
       06 LENGTH_OVER_ALL
       06 DISPLACEMENT
       06 BEAM
       06 PRICE
```

ou can print the desired information with nested FOR loops. For each boat from
ıe outer FOR statement, you want DATATRIEVE to loop through all the boats
ıd find all the ones with the same builder. For each one it finds, you want it to
ɔmpare its rig with the rig of the boat from the outer loop. Then you want to sep-
ate the ones for which the rigs are not the same. At first, you might be tempted
› use the following statement to produce the desired list:

```
R> SET NO PROMPT
R> FOR YACHTS
N>          FOR YACHTS WITH BUILDER = BUILDER AND
N>                  RIG NE RIG
N>                  PRINT BUILDER, RIG, RIG
R>
```

fter a long search for records, DATATRIEVE displays no records. The problem
that the syntax above asks DATATRIEVE to look for a boat with a rig that is
ɔt equal to itself -- an obvious contradiction. Both of the fields named RIG
solve to the record stream formed by the second FOR statement. The name
UILDER also resolves to the same record stream.

hat happens when you enter the above statement is that DATATRIEVE takes
e first record from YACHTS but does not look at any of the values in its fields.
ıen it looks at every record in YACHTS and discovers that for every one of
em, the name of the builder equals itself, but that no rig is not equal to itself.
ıus every record in YACHTS fails to meet the condition set by the statement.

ATATRIEVE then takes the second record in YACHTS and once again goes
rough all the boats, finding that the two values are always equal to themselves
d thus fail to meet the impossible demands of the statement. And so it goes for
ch record: two comparisons for 113 times 113 records, and no records meet the
lf-contradictory conditions.

The problem is how to get DATATRIEVE to look at the builder and rig of the outer FOR statement when making the comparison. The context variable provides one solution:

```
DTR> FOR A IN YACHTS
CON>    FOR YACHTS WITH BUILDER = A.BUILDER AND RIG NE A.RIG
CON>            PRINT BUILDER, A.RIG, RIG

MANUFACTURER  RIG     RIG

 AMERICAN     SLOOP   MS
 AMERICAN     MS      SLOOP
 CHALLENGER   SLOOP   KETCH
                .
                .
                .
 PEARSON      KETCH   SLOOP
 PEARSON      KETCH   SLOOP

DTR>
```

In this case, the use of the context variable A forces DATATRIEVE to look to th record stream formed by the outer FOR statement. At the same time, DATATRIEVE recognizes the unqualified names, RIG and BUILDER, in the context established by the most recent RSE: the one in the second FOR statement. The conditions in the second FOR statement are no longer impossible, anc information from 62 records is displayed.

The way DATATRIEVE treats the unqualified names in this example illustrates another rule for context resolution: the left-hand member of a Boolean expressio must resolve to the record selection expression of which it is a part. If you start the Boolean in the second FOR statement with A.BUILDER, DATATRIEVE tells you that A.BUILDER is undefined or used out of context.

You can add a second context variable in the above example to make sure the re olution of the names is explicitly stated:

```
DTR> FOR A IN YACHTS
CON>   FOR B IN YACHTS WITH B.BUILDER = A.BUILDER AND B.RIG NE A.RIG
CON>     PRINT B.BUILDER, A.RIG, B.RIG
```

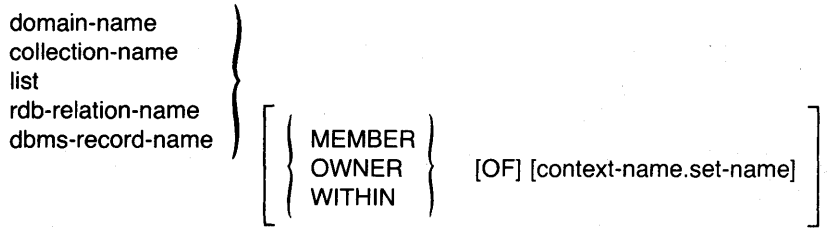
ou gain two advantages by specifying the second context variable: clarity of
presentation and the certainty of getting an error message from DATATRIEVE
you make a syntax error. Using the second context variable, however, does not
low you to violate the rule for resolving field names on the left side of Boolean
xpressions.

.1.2.3 The Effect of the CROSS Clause on Name Recognition -- You can use
ıe CROSS clause of the record selection expression to produce the same record
ream as the nested FOR statements in the previous example. The CROSS
ause, however, is not constrained by the rule for resolving field names on the
ft side of Boolean expressions.

ith the CROSS clause, you can establish more than one context variable (and,
nce, more than one context block) in a record selection expression. This is the
ntax of the CROSS clause:

ROSS [context-var IN] rse-source [OVER field-name]] [...]

ıe format for rse-source is:



\TATRIEVE creates a context block for each source in the CROSS clause. The
mes in all such context blocks resolve to the same record selection expression.
nsequently, adequately qualified names in the Booleans of the record selection
pression can appear on either the right-hand or left-hand side of any of the
oleans.

For example, any of the following statements produces the same result as the nested FOR statements of the previous example:

```
DTR> FOR A IN YACHTS CROSS B IN YACHTS WITH
CON>     B.BUILDER = A.BUILDER AND B.RIG NE A.RIG
CON>         PRINT B.BUILDER, A.RIG, B.RIG

DTR> FOR A IN YACHTS CROSS B IN YACHTS WITH
CON>     A.BUILDER = B.BUILDER AND A.RIG NE B.RIG
CON>         PRINT B.BUILDER, A.RIG, B.RIG

DTR> FOR A IN YACHTS CROSS YACHTS WITH
CON>     BUILDER = A.BUILDER AND RIG NE A.RIG
CON>         PRINT BUILDER, A.RIG, RIG

DTR> FOR A IN YACHTS CROSS YACHTS WITH
CON>     A.BUILDER = BUILDER AND A.RIG NE RIG
CON>         PRINT BUILDER, A.RIG, RIG
```

In cases where the sources specified in the CROSS clause share a field name, you can use the OVER clause to simplify the context specification. The field name specified in the OVER clause must exist in the records of all the sources specified in the CROSS clause. The following two statements are equivalent to the preceding ones:

```
DTR> FOR A IN YACHTS CROSS YACHTS OVER BUILDER WITH
CON>    RIG NE A.RIG
CON>            PRINT BUILDER, A.RIG, RIG

DTR> FOR A IN YACHTS CROSS YACHTS OVER BUILDER WITH
CON>    A.RIG NE RIG
CON>            PRINT BUILDER, A.RIG, RIG
```

To resolve field names in a record selection expression containing a CROSS clause, DATATRIEVE looks first at the context block for the last source specified in the CROSS clause. If that block contains no match for the field name, it begins looking at the context blocks for the other sources, working its way toward the block for the first source in the clause.

Consequently, when referring to fields from two or more identical sources, only those fields from the last source in the CROSS clause can remain unqualified. In such cases, you must use context variables to establish the appropriate context for fields from the other sources in the clause.

A.1.3 The Left Context Stack for Assignment Statements

When you make assignment statements at DATATRIEVE command level or as part of STORE or MODIFY statements, DATATRIEVE must assign values to

ıe field or variable you intend. It uses the left context stack to associate the val-
ɛs you supply with the fields and variables you want the values assigned to.
locks on the left context stack are for records and variables that you can update.

'henever DATATRIEVE begins to process a statement, the left context stack
ɔntains the global variables you have declared and not released. Any local
ıriables you declare in compound statements are also on the left context stack.
ıe local variables are removed when the statement in which you declared them
ıds.

ɔcal and global variables are on both stacks. Each type of variable has a value
at can be assigned to a field or another variable; hence, they are on the right
ntext stack. Both can be updated with new values you assign them; hence, they
e on the left context stack.

ɔntext blocks for a record you want to modify are also on both context stacks.
ıe record has a value you can use in Boolean expressions and assignment state-
ɛnts, and in a MODIFY statement you can update that value. Because a field is
both stacks at the same time, you can use the old value of the field to calculate
e new value. You can use the following form of assignment statement:

```
R> MODIFY USING PRICE = PRICE * 1.1
R>
```

\TATRIEVE retrieves the old value of PRICE associated with the name on the
ht context stack and multiplies the old PRICE by a constant. It then associates
ıt value with the name PRICE on the left context stack and updates the value
the PRICE field.

ıen you enter a STORE statement, the only context block for the new record is
the left context stack. No record exists yet, and, of course, no values are asso-
ted with its fields. The fields can only receive values.

ɪwever, as soon as DATATRIEVE associates a value with a field, you can move
ıt value to the right context stack and use it on the right side of assignment
tements. You can make this shift before you finish assigning values to all the
ıds of the new record. In fact, you can use the values of new fields to calculate
ɨ values DATATRIEVE stores in other new fields in the same record.

shift newly stored values to the right context stack, you include a context vari-
e with the domain name when you enter the STORE statement:

```
.> STORE A IN YACHTS USING . . .
```

Then, in the USING clause, you use the context variable to qualify the names of any field whose value you want to use on the right side of an assignment statement:

```
DTR> STORE A IN YACHTS USING
CON> BEGIN
CON>     F1 = value-expression
CON>     F2 = value-expression
CON>     F3 = A.F1 + A.F2
CON> END
DTR>
```

The context variable allows you to associate a field name on the right context stack with its new value as soon as you assign the value to the field. You cannot, however, use a field name on the right side of an assignment statement until you have assigned a value to the field.

You can combine STORE and MODIFY statements to keep an audit trail of changes made to records in a domain and to change statistical records when you store new records.

To form an audit trail, you need a domain for the audit records. This domain can use the same record definition as the original domain, but it must have its own domain definition and its own data file. Here is a simple example:

```
DTR> SHOW AUDIT_YACHTS
DOMAIN AUDIT_YACHTS USING
  YACHT ON AUD_YACHT;
DTR> FOR A IN YACHTS MODIFY USING
CON> BEGIN
CON>     BUILDER = *.BUILDER
CON>     MODEL =   *.MODEL
CON>     RIG =     *.RIG
CON>     LOA =     *.LOA
CON>     DISP =    *.WEIGHT
CON>     BEAM =    *.BEAM
CON>     PRICE =   *.PRICE
CON>     STORE B IN AUDIT_YACHTS USING
CON>        B.BOAT = A.BOAT
CON> END
Enter BUILDER:
```

If you have a VERIFY USING clause in the MODIFY statement, you should put the STORE statement as the last statement in the VERIFY clause. If you put the VERIFY clause after the STORE statement and the VERIFY clause aborts the change, you have a record of the change, but you have not changed the record.

ɔu can also embed a MODIFY statement in a STORE statement. In this exam-
ɛ, the embedded MODIFY statement updates a record of the last date a new
cord was added to the data file and records the TYPE field of the record stored.
ıe file LAST.DAT is a sequential file with one record in it.

```
R> SHOW LAST_ENTRY
MAIN LAST_ENTRY USING LAST_REC ON LAST.DAT;
R> SHOW LAST_REC
CORD LAST_REC USING
 TOP.
 LAST_DATE USAGE DATE.
 TYPE PIC X(20).

R> STORE A IN YACHTS USING
V> BEGIN
V>   BUILDER = *.BUILDER
V>   MODEL = *.MODEL
V>   RIG = *.RIG
V>   LOA = *.LOA
V>   DISP = *.DISP
V>   BEAM = *.BEAM
V>   PRICE = *.PRICE
V>   MODIFY B IN LAST_ENTRY USING
V>     BEGIN
V>       LAST_DATE = "TODAY"
V>       B.TYPE = A.TYPE
V>     END
V> END
ier BUILDER:
```

th the proper use of context variables, you can also store or change data in
ds shared by two or more domains.

2 Single Record Context

ɛ DATATRIEVE statements PRINT, MODIFY, and ERASE can act on one
ord at a time or on an entire record stream or collection. The records on which
y act are called target records. You can identify target records for these state-
nts in four ways:

A SELECT statement identifies one target record in a collection.

The keyword ALL in a statement, without an OF rse clause, makes all records in a collection the targets of the statement.

An OF rse clause in the statement forms a target record stream.

The RSE clause in a FOR statement forms a stream of target records for the statement contained in the FOR loop.

A.2.1 The SELECT Statement and the Single Record Context

Before discussing the SELECT statement and context, a short review of facts about collections is in order.

DATATRIEVE keeps a list of the collections you form with the FIND statement The most recent one formed is always at the top of the list and is called the CURRENT collection. The only other collections on the list are the ones to whic you assign a name when you form them. If you do not assign a name to the CURRENT collection, the next collection you form becomes the new CURREN collection. DATATRIEVE discards the old CURRENT collection unless you giv it a name when you form it.

With the RELEASE command, you can remove a collection from that list. If yo release the CURRENT collection, the next one on the list becomes the CURRENT collection.

No collection on this list, however, is represented by a block on the context stac unless you use the SELECT statement to single out one record in the collection When you select a record in a collection, DATATRIEVE puts a block for that collection on the context stack. If every existing collection has a selected record, then DATATRIEVE keeps a block on the context stack for each of thos collections.

The relative ages of the collections with selected records determine the order of context blocks for collections. The "oldest" collection with a selected record is t bottom of the context stack. Because the CURRENT collection is always the "youngest," its context block, if it has one, is nearest the top.

This order of context blocks for collections establishes the order DATATRIEVE uses not only for recognizing field names, as described above, but also for identi fying single target records. When you enter the most abbreviated forms of the PRINT, MODIFY, and ERASE statements, DATATRIEVE looks on the conte stack for the first valid single record context to carry out the specified action. I looks for the "youngest" collection with a selected record and either prints the record, erases it, or changes it.

The following sequence of examples illustrates the effect of the SELECT and DROP statements on single record context and the subsequent actions of the PRINT, MODIFY, and ERASE statements.

rm a collection of records from the YACHTS domain, call it BIGGIES, select
: third record as the target record, and display it:

```
t> READY YACHTS WRITE
t> FIND BIGGIES IN YACHTS WITH LOA > 40
 records found]
t> SELECT 3
t> PRINT

                                LENGTH
                                 OVER
IUFACTURER    MODEL      RIG     ALL    WEIGHT BEAM  PRICE

ILFSTAR     41          KETCH    41     22,000   12  $41,350

t>
```

)re a new record in the YACHTS domain and form a collection that consists of
ıt one record. Later, you can modify and erase this record:

```
:> STORE YACHTS
.er MANUFACTURER: HINKLEY
.er MODEL: BERMUDA 40
.er RIG: YAWL
.er LENGTH_OVER_ALL: 40
.er DISPLACEMENT: 20000
.er BEAM: 12
.er PRICE: 82,000
.> FIND YACHTS WITH BUILDER = "HINKLEY"
record found]
.>
```

ı now have two collections, CURRENT (the younger) and BIGGIES (the
er):

```
> SHOW COLLECTIONS
lections:
    CURRENT
    BIGGIES

> SHOW CURRENT
lection CURRENT
 Domain: YACHTS
 Number of Records: 1
 No Selected Record
> SHOW BIGGIES
lection BIGGIES
 Domain: YACHTS
 Number of Records: 8
 Selected Record: 3
>
```

The CURRENT collection has no selected record, but BIGGIES still does. Consequently, when you type PRINT and press the RETURN key again, DATATRIEVE prints the record in the first valid single record context, that is, the selected record in BIGGIES:

```
DTR> PRINT

                                  LENGTH
                                   OVER
MANUFACTURER    MODEL      RIG     ALL    WEIGHT BEAM  PRICE

 GULFSTAR     41          KETCH    41     22,000  12  $41,350

DTR>
```

When you type SELECT and press the RETURN key, DATATRIEVE selects tl first and only record in the CURRENT collection. Now when you type PRINT a: press the RETURN key, the single record context has changed. Now the selecte record in the CURRENT collection is the target record of the PRINT statement

```
DTR> SELECT
DTR> PRINT

                                  LENGTH
                                   OVER
MANUFACTURER    MODEL      RIG     ALL    WEIGHT BEAM  PRICE

 HINKLEY      BERMUDA 40 YAWL     40     20,000  12  $82,000

DTR> SHOW CURRENT
Collection CURRENT
    Domain: YACHTS
    Number of Records: 1
    Selected Record: 1
```

Now modify the price of the target record and display the result. The MODIFY and PRINT statements both act on the record in the first valid single record co text, that is, the selected record in the CURRENT collection:

```
DTR> MODIFY PRICE
Enter PRICE: 75,000
DTR> PRINT

                                  LENGTH
                                   OVER
MANUFACTURER    MODEL      RIG     ALL    WEIGHT BEAM  PRICE

 HINKLEY      BERMUDA 40 YAWL     40     20,000  12  $75,000

DTR>
```

ow type ERASE and press the RETURN key. The ERASE statement also acts
ı the record in the first valid single record context, and the record for the
INKLEY boat is removed from the data file YACHT.DAT. Even though you
ase the only record in the collection, DATATRIEVE does not discard the collec-
on. It takes note that you have erased the selected record and removes the con-
xt block for the CURRENT collection from the context stack. You can verify
ıe change in single record context by typing PRINT and pressing RETURN.
ıe selected record from BIGGIES is again in the first valid single record
ıntext:

```
R> ERASE
R> SHOW CURRENT
llection CURRENT
  Domain: YACHTS
  Number of Records: 1
  Selected Record: 1 (Erased)
R> PRINT

                              LENGTH
                               OVER
NUFACTURER    MODEL     RIG    ALL   WEIGHT BEAM  PRICE

ULFSTAR     41          KETCH  41    22,000  12  $41,350

R>
```

you type MODIFY or ERASE and press the RETURN key, and no existing
llection has a selected record, DATATRIEVE displays a message that there is
› target record for the action you propose:

```
R> ERASE
 target record for ERASE.
R> MODIFY
 target record for MODIFY.
R>
```

›wever, if you type PRINT and press the RETURN key, and no existing collec-
ın has a selected record, DATATRIEVE displays a message that there is no
lected record and then prints out the whole collection:

```
R> FIND YACHTS WITH BUILDER = "ALBIN"
 records found]
R> PRINT
 record selected, printing whole collection
```

(continued on next page)

```
                                    LENGTH
                                     OVER
MANUFACTURER    MODEL      RIG      ALL    WEIGHT BEAM  PRICE

 ALBIN          79         SLOOP    26      4,200  10   $17,900
 ALBIN          BALLAD     SLOOP    30      7,276  10   $27,500
 ALBIN          VEGA       SLOOP    27      5,070  08   $18,600

DTR>
```

You can change the single record context with the DROP statement. The DROP statement removes the selected record from a collection but does not erase the record from the data file. When you type DROP and press the RETURN key, an the CURRENT collection has no selected record, DATATRIEVE displays a message on your terminal:

```
DTR> FIND BIGGIES IN YACHTS WITH LOA > 40
[8 records found]
DTR> DROP
No collection with selected record for DROP.
```

If the CURRENT collection has a selected record, the DROP statement removes that record from the collection when you type DROP and press the RETURN key. If other collections have selected records, you must specify the collection name in the DROP statement.

The CURRENT collection is BIGGIES. Select and display the first record in BIGGIES and form a new CURRENT collection of boats built by Albin:

```
DTR> SELECT; PRINT

                                    LENGTH
                                     OVER
MANUFACTURER    MODEL      RIG      ALL    WEIGHT BEAM  PRICE

 CHALLENGER     41         KETCH    41     26,700  13   $51,228

DTR> FIND YACHTS WITH BUILDER = "ALBIN"
[3 records found]
DTR>
```

Now select, display, and drop the first record of the CURRENT collection. Then enter a SHOW CURRENT command to see how DATATRIEVE records the

sults of your actions. The SELECT creates a single record context for the cur-
nt collection, thus the target record of the PRINT statement is the selected
cord in the CURRENT collection, not in BIGGIES:

```
R> SELECT
R> PRINT

                                LENGTH
                                 OVER
NUFACTURER    MODEL      RIG     ALL    WEIGHT BEAM  PRICE

LBIN          79         SLOOP   26      4,200  10  $17,900

R> DROP
R> SHOW CURRENT
llection CURRENT
  Domain: YACHTS
  Number of Records: 3
  Selected Record: 1 (Dropped)
R>
```

hen you drop a selected record from a collection, you change the single record
ntext. The context block for that collection is removed from the context stack.

nsequently, when you type PRINT and press the RETURN key again,
ATATRIEVE displays the selected record in BIGGIES, the record in the first
lid single record context:

```
t> PRINT

                                LENGTH
                                 OVER
NUFACTURER    MODEL      RIG     ALL    WEIGHT BEAM  PRICE

IALLENGER  41            KETCH   41     26,700  13  $51,228

t>
```

like PRINT, MODIFY, and ERASE, the DROP statement does not act on the
ord in the first valid single record context. You have already dropped the
ected record in the CURRENT collection. When you type DROP and press the
TURN key again, DATATRIEVE displays a message on your terminal and
s not drop the selected record in BIGGIES. Because BIGGIES is not the
RRENT collection, you have to specify its name in the DROP statement:

```
> DROP
get record has already been dropped.
> DROP BIGGIES
> SHOW BIGGIES
lection BIGGIES
 Domain: YACHTS
 Number of Records: 8
 Selected Record: 1 (Dropped)
>
```

Now you have no valid single record context. When you type PRINT and press RETURN, DATATRIEVE displays the whole CURRENT collection because ther is no selected record in either of the two existing collections. Because you droppe one record from the CURRENT collection, it contains only two records now:

```
DTR> PRINT
No record selected, printing whole collection

                                LENGTH
                                 OVER
MANUFACTURER    MODEL      RIG   ALL   WEIGHT BEAM  PRICE

 ALBIN          BALLAD     SLOOP 30     7,276  10  $27,500
 ALBIN          VEGA       SLOOP 27     5,070  08  $18,600

DTR>
```

To show that you have not erased the record dropped from the CURRENT collec tion, form and display a new CURRENT collection of boats by Albin:

```
DTR> FIND YACHTS WITH BUILDER = "ALBIN"
[3 records found]
DTR> PRINT ALL

                                LENGTH
                                 OVER
MANUFACTURER    MODEL      RIG   ALL   WEIGHT BEAM  PRICE

 ALBIN          79         SLOOP 26     4,200  10  $17,900
 ALBIN          BALLAD     SLOOP 30     7,276  10  $27,500
 ALBIN          VEGA       SLOOP 27     5,070  08  $18,600

DTR>
```

A.2.2 The CURRENT Collection as Target Record Stream

The preceding example shows the effect of the keyword ALL on a PRINT state- ment that does not contain an OF rse clause.

Although DATATRIEVE acts on only one record at a time, you can identify mo than one record for a single DATATRIEVE statement to act on. With the keyword ALL, you can make every record in the CURRENT collection the targe of a single PRINT, MODIFY, or ERASE statement. Such a statement, howeve cannot also contain an OF rse clause.

If you have a CURRENT collection and type PRINT ALL and press the RETURN key, DATATRIEVE displays the whole CURRENT collection. If you

ave no CURRENT collection, DATATRIEVE displays a message on your termi-
al. To illustrate this effect, release all collections and enter the statement
RINT ALL:

```
TR> SHOW COLLECTIONS
ollections:
      CURRENT
      BIGGIES

TR> RELEASE CURRENT, BIGGIES
TR> SHOW COLLECTIONS
o established collections.
TR> PRINT ALL
 current collection has not been established.
TR>
```

ATATRIEVE displays the same message on your terminal when you have no URRENT collection and you enter either an ERASE ALL or MODIFY ALL atement.

hen you have a CURRENT collection and you enter an ERASE ALL state-ent, DATATRIEVE removes every record in the CURRENT collection from the ata file. Although frequently useful, this operation can jeopardize valuable data if ou use it carelessly.

he various forms of the MODIFY ALL statement change the data in each record the CURRENT collection. (See the article on the MODIFY statement in the *AX DATATRIEVE Reference Manual.*) Make a collection of the first three achts with no listed price. Display the CURRENT collection, modify the PRICE \$30,000, display the results of the change, and change the price back to zero sing a different form of the MODIFY ALL statement:

```
R> FIND FIRST 3 YACHTS WITH PRICE = 0
 records found]
R> PRINT ALL
```

NUFACTURER	MODEL	RIG	LENGTH OVER ALL	WEIGHT	BEAM	PRICE
LOCK I.	40	SLOOP	39	18,500	12	
UCCANEER	270	SLOOP	27	5,000	08	
UCCANEER	320	SLOOP	32	12,500	10	

```
DTR> MODIFY ALL PRICE
Enter PRICE: 30,000
DTR> PRINT ALL

                                LENGTH
                                 OVER
MANUFACTURER    MODEL     RIG    ALL    WEIGHT BEAM  PRICE

 BLOCK I.       40        SLOOP  39     18,500  12   $30,000
 BUCCANEER      270       SLOOP  27      5,000  08   $30,000
 BUCCANEER      320       SLOOP  32     12,500  10   $30,000

DTR> MODIFY ALL USING PRICE = 0; PRINT ALL

                                LENGTH
                                 OVER
MANUFACTURER    MODEL     RIG    ALL    WEIGHT BEAM  PRICE

 BLOCK I.       40        SLOOP  39     18,500  12
 BUCCANEER      270       SLOOP  27      5,000  08
 BUCCANEER      320       SLOOP  32     12,500  10

DTR>
```

If your collection contains many records and you mistakenly enter an ERASE ALL or MODIFY ALL statement, you can enter a CTRL/C to prevent all the records in the CURRENT collection from being erased or changed. How many records get erased or changed under such circumstances depends on the speed with which you enter the CTRL/C, the processing load on your system, and the priority of your process.

A.2.3 The OF rse Clause and Target Record Streams

The OF rse clause in a PRINT, ERASE, or MODIFY statement lets you create a new context for that statement. The OF rse clause specifies a target record stream that overrides any context established for your existing collections. For each such OF rse clause, DATATRIEVE puts a new block on the context stack. When DATATRIEVE completes execution of the statement, it removes that block from the context stack.

The following example contrasts the effect of PRINT, PRINT ALL, and PRINT OF rse. (When the PRINT statement does not include a list of fields, you can om the OF from the statement.) The record selection expression here is FIRST 3 YACHTS WITH PRICE = 0. This RSE identifies a new target record stream fo the PRINT statement that overrides the CURRENT collection as a target record stream. It also overrides the single record context of the selected record in the CURRENT collection:

```
DTR> FIND FIRST 3 YACHTS
[3 records found]
DTR> SELECT; PRINT
```

```
                                    LENGTH
                                     OVER
ANUFACTURER    MODEL      RIG       ALL    WEIGHT BEAM  PRICE

ALBERG         37 MK II   KETCH     37     20,000  12   $36,951

TR> PRINT ALL

                                    LENGTH
                                     OVER
ANUFACTURER    MODEL      RIG       ALL    WEIGHT BEAM  PRICE

ALBERG         37 MK II   KETCH     37     20,000  12   $36,951
ALBIN          79         SLOOP     26      4,200  10   $17,900
ALBIN          BALLAD     SLOOP     30      7,276  10   $27,500

TR> PRINT FIRST 3 YACHTS WITH PRICE = 0

                                    LENGTH
                                     OVER
ANUFACTURER    MODEL      RIG       ALL    WEIGHT BEAM  PRICE

BLOCK I.       40         SLOOP     39     18,500  12
BUCCANEER      270        SLOOP     27      5,000  08
BUCCANEER      320        SLOOP     32     12,500  10

'R>
```

o reduce the risk to your data, DATATRIEVE forces you to include both eywords ALL and OF when using the OF rse clause in MODIFY and ERASE atements. Although the results are not shown here, you must type MODIFY id ERASE statements to resemble the following examples. The record selection xpression used in these statements is PHONES WITH DEPT = "32T":

```
IDIFY ALL OF PHONES WITH DEPT = "32T"

DIFY ALL DEPT OF PHONES WITH DEPT = "32T"

DIFY ALL USING DEPT = *."NEW DEPT" OF PHONES WITH DEPT = "32T"

ASE ALL OF PHONES WITH DEPT = "32T"
```

nless you include assignment statements in the USING clause of a MODIFY atement, DATATRIEVE prompts you once to supply a value for each elemen- ry field specified or implied in the statement. After you respond to the last of e prompts, DATATRIEVE begins to change each of the records in the URRENT collection to correspond to the values you supplied to the prompts. ou can prevent any changes from taking effect by entering CTRL/Z when sponding to any of the prompts.

A.2.4 FOR Statements and Target Record Streams

You can use FOR statements to create target record streams for the DATATRIEVE statements that use single record context. Using FOR loops has an advantage over using target record streams formed by OF rse clause and the target record stream formed of the CURRENT collection by the keyword ALL. The FOR statement lets you work with each record individually; you do not have to perform the same operation on all target records. By putting STORE and MODIFY statements and prompting value expressions in a FOR loop, you can ac on each member of a record stream or collection one at a time.

When you put a MODIFY statement in a FOR statement, DATATRIEVE prompts you once for each field in the record if you do not specify a field list or a USING clause in the MODIFY statement.

This FOR statement creates a record stream of boats that have no price listed. The MODIFY statement prompts you to supply a price for each record in the record stream. You can put a unique value in the PRICE field for each boat:

```
DTR> READY YACHTS MODIFY
DTR> FOR YACHTS WITH PRICE MISSING MODIFY PRICE
Enter PRICE:  12900
Enter PRICE:  15600
Enter PRICE:
```

Another valuable feature of FOR loops is the complex relationships you create between record streams when you include one FOR loop inside another. Each FOR statement puts a block on the context stack. As a result, you can use the context mechanism to transfer values between records.

By putting a MODIFY statement inside two FOR statements, you can automatically update master records with the data from periodic transaction records:

```
DTR>  FOR A IN DAILY_TRANSACTIONS
CON>    FOR B IN MASTER_DATA WITH B.ACCOUNT = A.ACCOUNT
CON>      MODIFY USING
CON>        BEGIN
CON>          MASTER_BAL    = MASTER_BAL    - WITHDRAW + DEPOSIT
CON>          TOT_WITHDRAW = TOT_WITHDRAW + WITHDRAW
CON>          TOT_DEPOSIT  = TOT_DEPOSIT  + DEPOSIT
CON>        END
DTR>
```

The Boolean expression in this example limits the record stream for the inner FOR statement to one record.

You can also create nested FOR statements in which DATATRIEVE executes a series of statements at each level of nesting. For each owner record in the next example, DATATRIEVE asks you if you want to modify the SPECS field of evei boat in the YACHTS inventory built by the manufacturer of the owner's boat.

'he third time through the outer loop, DATATRIEVE again begins the cycle of rompting for the boats by Albin, because the third person in the OWNERS omain also owns a boat by Albin. Notice that the record changed during the sec- nd loop appears during the third:

```
TR>FOR OWNERS
)N>BEGIN
)N>     PRINT SKIP, BUILDER, SKIP
)N>     FOR YACHTS WITH BOAT.BUILDER = OWNER.BUILDER
)N>         BEGIN
)N>            PRINT SPECS
)N>            IF *."DO YOU WANT TO CHANGE THIS" CONT "Y"
)N>               THEN MODIFY SPECS
)N>         END
)N>END

)UILDER

.BERG

      LENGTH
       OVER
:IG    ALL   WEIGHT BEAM  PRICE

:TCH   37    20,000  12  $36,000
ter DO YOU WANT TO CHANGE THIS:  N

.BIN

.OOP   26     4,200  10  $17,900
ter DO YOU WANT TO CHANGE THIS:  N
.OOP   30     7,276  10  $27,500
ter DO YOU WANT TO CHANGE THIS:  N
OOP   27     5,070  08  $18,600
ter DO YOU WANT TO CHANGE THIS:  Y
ter RIG:  KETCH
ter LENGTH_OVER_ALL:  35
ter DISPLACEMENT:  17000
ter BEAM:  12
ter PRICE:  33000

BIN

OOP   26     4,200  10  $17,900
ter DO YOU WANT TO CHANGE THIS:  N
OOP   30     7,276  10  $27,500
ter DO YOU WANT TO CHANGE THIS:  N
TCH   35    17,000  12  $33,000
ter DO YOU WANT TO CHANGE THIS:  N

C

)OP   31     8,650  09
ter DO YOU WANT TO CHANGE THIS:  ^Z
ecution terminated by operator
R>
```

Sample Database Definitions and Procedures B

his appendix contains:

Record and domain definitions used in the sample databases that come with the VAX DATATRIEVE software.

Table and view definitions that use or supplement those domains.

Procedures that use the sample databases.

Rdb and DBMS data definitions and procedures used in the DATATRIEVE UETP, the User Environment Test Package.

.1 RMS Data Definitions and Procedures

```
CORD ANNUAL_REC
 DATA.
 03 DATE DATE PIC YYYY.
 03 EQUIPMENT_SALES REAL      EDIT_STRING ZZZ9.9.
 03 SERVICES REAL     EDIT_STRING ZZ9.9.
 03 REVENUE COMPUTED BY EQUIPMENT_SALES + SERVICES    EDIT_STRING ZZZ9.9.
 03 NET_INCOME REAL   EDIT_STRING ZZ9.9.
 03 NET_INCOME_PER_SHARE REAL EDIT_STRING ZZ9.9.
 03 RESEARCH REAL QUERY_NAME DEVELOPMENT      EDIT_STRING ZZ9.9.
 03 INVENTORIES REAL  EDIT_STRING ZZ9.9.
 03 EMPLOYEES REAL EDIT_STRING IS ZZ,ZZZ.
 03 FILLER PIC X(68).


MAIN ANNUAL_REPORT USING ANNUAL_REC ON DTR$LIBRARY:ANNUAL.DAT;
```

```
!
!
PROCEDURE BILL_PAID
READY PAYABLES SHARED READ
REPORT PAYABLES WITH BILL_PAID MISSING AND
   ITEMS_RECEIVED NOT MISSING AND
   INVOICE_DUE NOT MISSING SORTED BY INVOICE_DUE
SET REPORT_NAME = "Accounts Payable"
SET COLUMNS_PAGE = 65
PRINT RUNNING COUNT ("COUNT"), MANUFACTURER,
   ITEMS_RECEIVED,INVOICE_DUE,BILL_PAID,WHSLE_PRICE,
   COL 55,RUNNING TOTAL WHSLE_PRICE ("TOTAL"/"OWNED") USING $$$,$$$
END_REPORT
END-PROCEDURE

!
!
PROCEDURE BILL_PAID_1
READY PAYABLES SHARED READ
REPORT PAYABLES WITH BILL_PAID MISSING AND
   ITEMS_RECEIVED NOT MISSING AND
   INVOICE_DUE NOT MISSING SORTED BY INVOICE_DUE
SET REPORT_NAME = "Accounts Payable"
SET COLUMNS_PAGE = 65
PRINT RUNNING COUNT ("COUNT"), MANUFACTURER,
   ITEMS_RECEIVED,INVOICE_DUE,BILL_PAID,WHSLE_PRICE,
   RUNNING TOTAL WHSLE_PRICE ("TOTAL"/"OWNED") USING $$$,$$$
END_REPORT
END-PROCEDURE

!
!
DOMAIN FAMILIES
 USING FAMILY_REC ON DTR$LIBRARY:FAMILY.DAT;

!
!
RECORD FAMILY_REC
01 FAMILY.
   03 PARENTS.
      06 FATHER PIC X(10).
      06 MOTHER PIC X(10).
   03 NUMBER_KIDS PIC 99 EDIT_STRING IS Z9.
   03 KIDS OCCURS 0 TO 10 TIMES DEPENDING ON NUMBER_KIDS.
      06 EACH_KID.
         09 KID_NAME PIC X(10) QUERY_NAME IS KID.
         09 AGE PIC 99 EDIT_STRING IS Z9.
;

!
!
DOMAIN KETCHES
 OF YACHTS BY
01 KETCH OCCURS FOR YACHTS WITH RIG EQ "KETCH".
  03 TYPE FROM YACHTS.
  03 LOA FROM YACHTS.
  03 PRICE FROM YACHTS.
;
```

```
:OCEDURE LOA_REPORT
:PORT ON *.FILE
 SET REPORT_NAME="JIM'S VERY OWN LISTING"/"OF"/"INTERESTING SAILBOATS"/
     "(BY LENGTH)"
 SET LINES_PAGE=55, COLUMNS_PAGE=72
 AT TOP OF LOA PRINT LOA("LENGTH")
 PRINT TYPE, RIG, DISP, BEAM USING Z9 , PRICE
 AT BOTTOM OF LOA PRINT SKIP, COL 32, "*** AVERAGE ***",
    AVERAGE DISP, AVERAGE BEAM, AVERAGE PRICE
 AT BOTTOM OF REPORT PRINT SKIP, "REPORT AVERAGES",
    AVERAGE DISP, AVERAGE BEAM, AVERAGE PRICE
 AT BOTTOM OF PAGE PRINT SKIP, COL 20,
    """ANOTHER SERVICE OF QUERY ENTERPRISES"""
D_REPORT
D-PROCEDURE


MAIN OWNERS
SING OWNER_RECORD ON OWNER.DAT;


MAIN OWNERS_SEQUENTIAL USING OWNER_RECORD ON DTR$LIBRARY:OWNER.SEQ;


CORD OWNER_RECORD
 OWNER.
 03 NAME PIC X(10) QUERY_HEADER IS "OWNER"/"NAME"
    EDIT_STRING IS X(5).
 03 BOAT_NAME PIC X(17) QUERY_HEADER IS "BOAT NAME".
 03 TYPE.
    06 BUILDER PIC X(10).
    06 MODEL PIC X(10).


IAIN PAYABLES USING PAYABLES_REC ON DTR$LIBRARY:PAYABLES.DAT;
```

```
!
!
RECORD PAYABLES_REC USING
     01 PAYABLE.
          05 ORDER_NUM PIC 9(7).
          05 TYPE.
              10 MANUFACTURER    PIC IS X(10)
                 QUERY_NAME IS BUILDER
                 QUERY_HEADER IS "VENDOR".
              10 MODEL           PIC IS X(10)
                 QUERY_HEADER IS "ITEM_TYPE".
          05 WHSLE_PRICE   PIC 9(5)
                EDIT_STRING IS $$$,$$$.
          05 ITEMS_RECEIVED      USAGE IS DATE
                MISSING VALUE IS 010101
                EDIT_STRING IS NN/DD/YY?"NO GOODS".
          05 INVOICE_DUE         USAGE IS DATE
                MISSING VALUE IS 010101
                EDIT_STRING IS NN/DD/YY?"NO INVCE".
          05 BILL_PAID           USAGE IS DATE
                MISSING VALUE IS 010101
                EDIT_STRING IS NN/DD/YY?"NOT PAID".
          05 AGE COMPUTED BY FN$FLOOR(("TODAY" - INVOICE_DUE)/30)
                EDIT_STRING IS Z9.
;

!
!
DOMAIN PERSONNEL USING PERSONNEL_REC ON DTR$LIBRARY:PERSON.DAT;

!
!
RECORD PERSONNEL_REC USING
01 PERSON.
   05 ID                          PIC IS 9(5).
   05 EMPLOYEE_STATUS             PIC IS X(11)
                                  QUERY_NAME IS STATUS
                                  QUERY_HEADER IS "STATUS"
                                  VALID IF STATUS EQ "TRAINEE","EXPERIENCE
   05 EMPLOYEE_NAME               QUERY_NAME IS NAME.
      10 FIRST_NAME               PIC IS X(10)
                                  QUERY_NAME IS F_NAME.
      10 LAST_NAME                PIC IS X(10)
                                  QUERY_NAME IS L_NAME.
   05 DEPT                        PIC IS XXX.
   05 START_DATE                  USAGE IS DATE
                                  DEFAULT VALUE IS "TODAY".
   05 SALARY                      PIC IS 9(5)
                                  EDIT_STRING IS $$$,$$$.
   05 SUP_ID
                                  PIC IS 9(5)
                                  MISSING VALUE IS 0.
;

!
!
DOMAIN PETS USING PET_REC ON DTR$LIBRARY:PET.DAT;
```

```
ECORD PET_REC
L FAMILY.
  03 PARENTS.
     06 FATHER PIC X(10).
     06 MOTHER PIC X(10).
  03 NUMBER_KIDS PIC 99 EDIT_STRING IS Z9.
  03 KIDS OCCURS 0 TO 10 TIMES DEPENDING ON NUMBER_KIDS.
     06 EACH_KID.
        09 KID_NAME PIC X(10) QUERY_NAME IS KID.
        09 KID_AGE PIC 99 EDIT_STRING IS Z9.
        09 PET OCCURS 2 TIMES.
           13  PET_NAME  PIC X(10).
           13  PET_AGE   PIC 99.



ABLE PRICES FROM YACHTS USING
TPE : PRICE
.SE 0
ID_TABLE



LOCEDURE PRICE_PER_POUND
LICE/DISP ("PRICE"/"PER"/"POUND") USING $$.99
ID-PROCEDURE



IMAIN PROJECTS USING PROJECT_REC ON DTR$LIBRARY:PROJECT.DAT;



ICORD PROJECT_REC USING
  PROJECT_REC.
   03  PROJ_CODE     PIC 9(3)  QUERY_NAME IS CODE.
   03  PROJ_NAME     PIC X(10) QUERY_NAME IS NAME.
   03  MANAGER_NUM   PIC 9(5)  QUERY_NAME IS NUM.



ABLE RIG_TABLE
SLOOP" : "ONE MAST",
ETCH" : "TWO MASTS, BIG ONE IN FRONT",
'AWL"  : "SIMILAR TO KETCH",
I/S"   : "SAILS AND BIG MOTOR",
.SE "SOMETHING ELSE"
ID_TABLE
```

```
!
!
DOMAIN SAILBOATS
 OF YACHTS, OWNERS BY
01 SAILBOAT OCCURS FOR YACHTS.
   03 BOAT FROM YACHTS.
   03 SKIPPERS OCCURS FOR OWNERS WITH TYPE EQ BOAT.TYPE.
      05 NAME FROM OWNERS.
;

!
!
DOMAIN SALES USING SALES_REC
        ON DTR$LIBRARY:SALES.DAT;

!
!
RECORD SALES_REC USING
01 SALESREC.
      05 SALES_NAME    PIC IS X(20).
      05 START_DATE    USAGE DATE.
      05 MONTHS_EMP    COMPUTED BY ("TODAY" - START_DATE)/30
                       EDIT_STRING IS ZZ9.
      05 AMOUNT        PIC IS 9(5)V99
                       EDIT_STRING IS $$$,$$$.99.
      05 COMM_PCT       COMPUTED BY
                       CHOICE
                          (MONTHS_EMP LE 6 AND AMOUNT > 5000) THEN 10
                          (MONTHS_EMP LE 6) THEN 05
                          (AMOUNT > 10000) THEN 12
                          ELSE 07
                       END_CHOICE
                       EDIT_STRING IS Z9%.
      05 RATING        COMPUTED BY
                       CHOICE
                          (MONTHS_EMP LE 6 AND AMOUNT > 5000) THEN "ABOVE
                          (AMOUNT > 10000) THEN "ABOVE QUOTA"
                          ELSE "BELOW QUOTA"
                       END_CHOICE.
;

!
!
PROCEDURE VERIFY
VERIFY USING
  BEGIN
     RIG = FN$UPCASE (RIG)
     PRINT
     DISPLAY "IF RECORD IS OK, CONFIRM WITH Y"
     IF *.CONFIRM NOT CONTAINING "Y" THEN ABORT "UPDATE ABORTED"
  END
END-PROCEDURE
```

```
ECORD YACHT USING
1 BOAT.
 03 TYPE.
    06 MANUFACTURER PIC X(10)
       QUERY_NAME IS BUILDER.
    06 MODEL PIC X(10).
 03 SPECIFICATIONS
    QUERY_NAME SPECS.
    06 RIG PIC X(6)
       VALID IF RIG CONT "SLOOP","KETCH","MS","YAWL".
    06 LENGTH_OVER_ALL PIC XXX
       VALID IF LOA BETWEEN 15 AND 50
       QUERY_NAME IS LOA.
    06 DISPLACEMENT PIC 99999
       QUERY_HEADER IS "WEIGHT"
       EDIT_STRING IS ZZ,ZZ9
       QUERY_NAME IS DISP.
    06 BEAM PIC 99 MISSING VALUE IS 0.
    06 PRICE PIC 99999
       MISSING VALUE IS 0
       VALID IF PRICE>DISP*1.3 OR PRICE EQ 0
       EDIT_STRING IS $$$,$$$.



)MAIN YACHTS USING YACHT ON YACHT.DAT;


)MAIN YACHTS_SEQUENTIAL USING YACHT ON DTR$LIBRARY:YACHT.SEQ;
```

.2 DBMS Data Definitions and Procedures

```
:OCEDURE BOM_LIST
)R FIRST 5 PARTS WITH PART_STATUS = "G"
  BEGIN
  PRINT "Part :" ||| PART_ID ||| "(" ||| PART_DESC ||| ")"
  FOR CLASSES OWNER CLASS_PART
     PRINT "    Class :" ||| CLASS_DESC
  PRINT "    Cost  :", PART_COST (-) USING $$$$$,$$9.999
  PRINT "    Price :", PART_PRICE (-) USING $$$$$,$$9.999
  PRINT "    Bill of Materials :"
  FOR COMPONENTS MEMBER OF PART_USES
     FOR PARTS OWNER PART_USED_ON
        PRINT "        Quantity : ", COMP_QUANTITY (-) USING ZZZ9,
           "    Part :" ||| PART_ID ||| "(" ||| PART_DESC ||| ")"
  PRINT SKIP
  END
 D-PROCEDURE
```

```
!
!
DOMAIN CLASSES
 USING CLASS
  OF DATABASE PARTS_DB;

!
!
DOMAIN CLASS_VIEW OF
 CLASSES, PART_S, EMPLOYEES, COMPONENTS, SUPPLIES USING
01  CLASS_RECORD OCCURS FOR CLASSES.
    03  CLASS_CODE FROM CLASSES.
    03  PART_GROUP OCCURS FOR PART_S MEMBER OF CLASS_PART.
        05  PART_DESC FROM PART_S.
        05  EMPLOYEE_GROUP OCCURS FOR EMPLOYEES OWNER RESPONSIBLE_FOR.
            07  EMP_ID FROM EMPLOYEES.
        05  COMPONENT_GROUP OCCURS FOR COMPONENTS MEMBER OF PART_USES.
            07  COMP_SUB_PART FROM COMPONENTS.
        05  SUPPLY_GROUP OCCURS FOR SUPPLIES MEMBER OF PART_INFO.
            07  SUP_TYPE FROM SUPPLIES.
;

!
!
DOMAIN COMPONENTS
 USING COMPONENT
  OF DATABASE PARTS_DB;

!
!
DOMAIN DIVISIONS
 USING DIVISION
  OF DATABASE PARTS_DB;

!
!
DOMAIN EMPLOYEES
 USING EMPLOYEE
  OF DATABASE PARTS_DB;

!
!
DATABASE PARTS_DB
 USING SUBSCHEMA DTR_SUBSCHEMA
  OF SCHEMA PARTS
 ON DTR$LIBRARY:DTRPARTDB;

!
!
DOMAIN PART_S
 USING PART
  OF DATABASE PARTS_DB;
```

```
ROCEDURE PRINT_NEW_DIVISION
)R DIVISIONS WITH DIV_NAME CONTAINING "Firmware"
   PRINT ALL DIV_NAME,
         ALL COL 30, EMP_LAST_NAME ("Manager"/"_________________") OF
                     EMPLOYEES OWNER MANAGES,
         ALL COL 60, EMP_LAST_NAME ("Personpower"/"_________________") OF
                     EMPLOYEES MEMBER OF CONSISTS_OF
ID-PROCEDURE


)MAIN QUOTES
)SING QUOTE
 OF DATABASE PARTS_DB;


:OCEDURE READY_PARTS
:ADY CLASSES, PART_S AS PARTS, COMPONENTS, VENDORS, SUPPLIES, QUOTES,
IPLOYEES, DIVISIONS
ID-PROCEDURE


:OCEDURE READY_PARTS_WRITE
:ADY CLASSES WRITE, PART_S AS PARTS WRITE, COMPONENTS WRITE, VENDORS WRITE,
PPLIES WRITE, QUOTES WRITE, EMPLOYEES WRITE, DIVISIONS WRITE
D-PROCEDURE
```

```
!
!
PROCEDURE RESPONSIBLE_LIST
FOR FIRST 5 OVERSEER IN EMPLOYEES
    BEGIN
    DECLARE TEXT PICTURE X(1000).
    TEXT = EMP_FIRST_NAME ||| EMP_LAST_NAME
    IF ANY PARTS MEMBER RESPONSIBLE_FOR
        THEN BEGIN
            TEXT = TEXT ||| "oversees production of"
            TEXT = TEXT ||| COUNT OF PARTS MEMBER RESPONSIBLE_FOR
            TEXT = TEXT ||| "part"
            IF COUNT OF PARTS MEMBER RESPONSIBLE_FOR NOT EQUAL 1
                THEN TEXT = TEXT || "s"
            TEXT = TEXT || "."
            TEXT = TEXT ||| "He/She"
            END
    FOR DIVISIONS OWNER CONSISTS_OF
        BEGIN
        FOR MANAGER IN EMPLOYEES OWNER MANAGES
            BEGIN
            DECLARE COHORT_COUNT PICTURE 999.
            DECLARE COHORT_NUMBER PICTURE 9.
            COHORT_COUNT =
             COUNT OF EMPLOYEES MEMBER OF OVERSEER.CONSISTS_OF WITH
                        EMP_ID NOT EQUAL OVERSEER.EMP_ID AND
                        EMP_ID NOT EQUAL MANAGER.EMP_ID
            IF COHORT_COUNT > 0
                THEN TEXT = TEXT ||| "and his/her cohort"
            IF COHORT_COUNT > 1
                THEN TEXT = TEXT || "s"
            COHORT_NUMBER = 0
            FOR FIRST 3 EMPLOYEES MEMBER OF OVERSEER.CONSISTS_OF WITH
                                EMP_ID NOT EQUAL OVERSEER.EMP_ID AND
                                EMP_ID NOT EQUAL MANAGER.EMP_ID
               BEGIN
               COHORT_NUMBER = COHORT_NUMBER + 1
               IF COHORT_NUMBER > 1
                  THEN TEXT = TEXT || ","
               IF COHORT_COUNT = COHORT_NUMBER AND COHORT_COUNT > 1
                  THEN TEXT = TEXT ||| "and"
               TEXT = TEXT ||| EMP_FIRST_NAME ||| EMP_LAST_NAME
               END
            IF COHORT_COUNT > 3
               THEN TEXT = TEXT || ", and others too numerous to mentio
            TEXT = TEXT ||| "report"
            IF COHORT_COUNT = 0
               THEN TEXT = TEXT || "s"
            TEXT = TEXT ||| "to"
            TEXT = TEXT ||| EMP_FIRST_NAME ||| EMP_LAST_NAME
            TEXT = TEXT || ", manager of the"
            END
        TEXT = TEXT ||| DIV_NAME
        TEXT = TEXT ||| "division."
        END
    PRINT SKIP, TEXT (-) USING T(70)
    END
END-PROCEDURE
```

```
)OMAIN SUPPLIES
USING SUPPLY
 OF DATABASE PARTS_DB;


OMAIN VENDORS
USING VENDOR
 OF DATABASE PARTS_DB;
```

3.3 Rdb Data Definitions and Procedures

```
)MAIN COLLEGES
       USING COLLEGES OF DATABASE PERSONNEL;


)MAIN DEGREES
       USING DEGREES OF DATABASE PERSONNEL;


)MAIN DEPARTMENTS
       USING DEPARTMENTS OF DATABASE PERSONNEL;


)MAIN DEPARTMENT_STAFF OF DEPARTMENTS, EMPLOYEES, JOB_HISTORY
 TOP OCCURS FOR DEPARTMENTS CROSS JOB_HISTORY OVER DEPARTMENT_CODE CROSS
       EMPLOYEES OVER EMPLOYEE_ID WITH JOB_END MISSING.
       03 DEPARTMENT_CODE FROM DEPARTMENTS.
       03 DEPARTMENT_NAME FROM DEPARTMENTS.
       03 EMPLOYEE_ID FROM EMPLOYEES.
       03 FIRST_NAME FROM EMPLOYEES.
       03 LAST_NAME FROM EMPLOYEES.


BLE DEPARTMENT_TABLE FROM DEPARTMENTS USING
PARTMENT_CODE : DEPARTMENT_NAME
SE "No department"
D_TABLE


MAIN EMPLOYEES
       USING EMPLOYEES OF DATABASE PERSONNEL;
```

```
!
!
DOMAIN EMPLOYEE_EDUCATION OF EMPLOYEES, COLLEGES, DEGREES USING
01 TOP OCCURS FOR EMPLOYEES.
        03 LAST_NAME FROM EMPLOYEES.
        03 DEG OCCURS FOR DEGREES WITH EMPLOYEE_ID EQ EMPLOYEES.EMPLOYEE_
                05 DEGREE FROM DEGREES.
                05 DEGREE_FIELD FROM DEGREES.
                05 COLL OCCURS FOR COLLEGES WITH
                        COLLEGE_CODE EQ DEGREES.COLLEGE_CODE.
                        07 COLLEGE_NAME FROM COLLEGES.
;

!
!
PROCEDURE EMPLOYEE_INFO
BEGIN
FOR FIRST 1 EMPLOYEES
    BEGIN
    PRINT NEW_PAGE, SKIP 2
    PRINT COL 30, "Employee Profile", SKIP 2
    FOR WORK_STATUS WITH STATUS_CODE EQ EMPLOYEES.STATUS_CODE
    PRINT "Id:", COL 15, EMPLOYEE_ID(-) USING X(10),
                COL 50, STATUS_NAME|||STATUS_TYPE, SKIP
    PRINT "Name:", COL 15, FIRST_NAME|||MIDDLE_INITIAL||"."|||LAST_NAME
    IF ADDRESS_DATA NE " " THEN
        BEGIN
        PRINT "Address: ", COL 15, ADDRESS_DATA(-),
           COL 15, STREET(-)
        END ELSE  PRINT "Address: ", COL 15, STREET(-)
    PRINT COL 15, TOWN||",",|||STATE||"    "|ZIP
    PRINT SKIP, "Job History:"
    FOR JOB_HISTORY WITH
        EMPLOYEE_ID EQ EMPLOYEES.EMPLOYEE_ID SORTED BY
          DESCENDING JOB_START
        BEGIN
        FOR JOBS WITH JOB_CODE EQ JOB_HISTORY.JOB_CODE
            BEGIN
            PRINT (DEPARTMENT_CODE VIA DEPARTMENT_TABLE)("DEPT") USING 1
            JOB_TITLE,
            WAGE_CLASS,
            MINIMUM_SALARY USING $$$,$$$,
            MAXIMUM_SALARY USING $$$,$$$,
            JOB_START USING NN/DD/YY,
            JOB_END USING NN/DD/YY
            END
        END
     PRINT "-------------------------------------------------------",
            SKIP, "Salary History:"
    FOR SALARY_HISTORY WITH EMPLOYEE_ID EQ EMPLOYEES.EMPLOYEE_ID SORTED
        DESCENDING SALARY_START
        PRINT SALARY_START USING NN/DD/YY,
        SALARY_END USING NN/DD/YY,
        SALARY_AMOUNT USING $$$,$$$
    IF ANY DEGREES WITH EMPLOYEE_ID EQ EMPLOYEES.EMPLOYEE_ID THEN
     PRINT "-------------------------------------------------------",
        SKIP, "Education:", SKIP
```

(continued on next page)

```
  FOR DEGREES WITH EMPLOYEE_ID = EMPLOYEES.EMPLOYEE_ID
      FOR COLLEGES WITH COLLEGE_CODE = DEGREES.COLLEGE_CODE
      BEGIN
      PRINT COLLEGE_NAME, YEAR_GIVEN, DEGREE,
          DEGREE_FIELD
      END
   PRINT "------------------------------------------------------------",
  END
  PRINT NEW_PAGE
D
D-PROCEDURE


MAIN JOBS
      USING JOBS OF DATABASE PERSONNEL;


MAIN JOB_HISTORY
      USING JOB_HISTORY OF DATABASE PERSONNEL;


BLE NAME_TABLE FROM EMPLOYEES USING
PLOYEE_ID : LAST_NAME
SE " "
D_TABLE


)CEDURE READY_PERSONNEL
ADY COLLEGES SHARED READ, DEGREES SHARED READ, -
    DEPARTMENTS SHARED READ, EMPLOYEES SHARED READ, -
    JOBS SHARED READ, JOB_HISTORY SHARED READ, -
    SALARY_HISTORY SHARED READ, WORK_STATUS SHARED READ
)-PROCEDURE


)CEDURE READY_PERSONNEL_WRITE
ADY COLLEGES WRITE, DEGREES WRITE, DEPARTMENTS WRITE,-
    EMPLOYEES WRITE, JOBS WRITE, JOB_HISTORY WRITE,-
    SALARY_HISTORY WRITE, WORK_STATUS WRITE
)-PROCEDURE


AIN SALARY_HISTORY
      USING SALARY_HISTORY OF DATABASE PERSONNEL;
```

```
!
!
PROCEDURE SALARY_REPORT
FIND FIRST 3 EMPLOYEES
FIND CURRENT CROSS SALARY_HISTORY OVER EMPLOYEE_ID
FIND CURRENT CROSS JOB_HISTORY OVER EMPLOYEE_ID WITH
        JOB_START EQ SALARY_HISTORY.SALARY_START
FIND CURRENT CROSS JOBS WITH JOB_CODE EQ JOB_HISTORY.JOB_CODE
REPORT CURRENT SORTED BY LAST_NAME, SALARY_AMOUNT
AT TOP OF EMPLOYEE_ID PRINT SKIP, COL 1,
        EMPLOYEE_ID|||FIRST_NAME|||LAST_NAME, SKIP 2
PRINT COL 1, JOB_START USING NN/DD/YY, JOB_TITLE,
        MINIMUM_SALARY, SALARY_AMOUNT,MAXIMUM_SALARY
AT BOTTOM OF EMPLOYEE_ID PRINT
"------------------------------------------------------------"
END_REPORT
END-PROCEDURE

!
!
DOMAIN WORK_STATUS
        USING WORK_STATUS OF DATABASE PERSONNEL;
```

Index

this index, a page number followed
a "t" indicates a table reference.

B

C

E

F

M

N

O

P

S

HOW TO ORDER ADDITIONAL DOCUMENTATION

DIRECT TELEPHONE ORDERS

In Continental USA and Puerto Rico call **800–258–1710**

In Canada call **800–267–6146**

In New Hampshire, Alaska or Hawaii call **603–884–6660**

DIRECT MAIL ORDERS (U.S. and Puerto Rico*)

DIGITAL EQUIPMENT CORPORATION
P.O. Box CS2008
Nashua, New Hampshire 03061

DIRECT MAIL ORDERS (Canada)

DIGITAL EQUIPMENT OF CANADA LTD.
940 Belfast Road
Ottawa, Ontario, Canada K1G 4C2
Attn: P&SG Business Manager

INTERNATIONAL

DIGITAL EQUIPMENT CORPORATION
P&SG Business Manager
c/o Digital's local subsidiary
or approved distributor

Internal orders should be placed through the Software Distribution Center (SDC), Digital Equipment Corporation, Northboro, Massachusetts 01532

*Any prepaid order from Puerto Rico must be placed with the Local Digital Subsidiary:
809–754–7575

VAX DATATRIEVE
User's Guide
AA-K080D-TE

Reader's Comments

Note: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement. _____

_____

_____

_____

_____

_____

_____

Did you find errors in this manual? If so, specify the error and the page number. _____

_____

_____

_____

_____

_____

_____

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify) _____

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code or Country _____

Do Not Tear - Fold Here and Tape

digital

No Postage Necessary if Mailed in the United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

ATTN: DISG Documentation ZKO2–2/N53
DIGITAL EQUIPMENT CORPORATION
110 SPIT BROOK ROAD
NASHUA, N.H. 03062

Do Not Tear - Fold Here and Tape