Section II:

In this section, we develop the use of expressions to modify dBASE II commands. This is may be the most important part of learning how to use dBASE II effectively.

The dBASE II commands can be learned fairly easily because they are English-like, and learning another command is a matter of increasing your vocabulary (and your repertoire) by another word.

Expressions, combined with the commands, give you the fine control you need to manipulate your data to perform specific tasks. Once you have learned how to handle expressions, you will only have to learn two more things about programming to be able to write effective applications command files. (These are how to make decisions and how to repeat a sequence of commands, covered in Section III).

Using expressions for selection and control

We gave you a brief introduction to expressions that can be used with dBASE II commands in Section I.

As you saw, they are a powerful way to extend the commands and manipulate your data quickly and easily. If you check the index of commands in Section VI, you'll see that many dBASE II commands can be modified in the form:

```
^<COMMAND> [FOR <expression>]^
```

This extended power gives you a flexibility that you simply do not get with other database management systems. We've been told by experienced programmers that they can write a program (a dBASE II command file) for an application in as little as one-tenth the time it would take them using BASIC or even higher level languages such as COBOL, FORTRAN and PL/1.

But to take advantage of this power, you need to understand how to work with expressions and operators, then how to combine the modified commands into command files that will perform the same tasks again and again.

The next few pages will get you started. Ultimately, experience is going to be the best teacher.

Reminder: as we introduce commands through the text, we try to explain a particular aspect of the command that will allow you to a few more things with your database. This means that we do not cover the entire command at one time. To find out all that a command can do, use the summary at the end of Part I and the definitions of Part II.

Note: If, after you've finished this Section, you are still uncertain about how to write expressions that make the dBASE II commands do exactly what you want done, you may want to look at some beginning programming texts at your local library. Most of them discuss expressions within the first two chapters or so.

Constants and variables (STORE)

Expressions in dBASE II are used to help select and manipulate the data in your database (see ^DISPLAY^). The quantity that you manipulate may be either a **constant** or a **variable**.

Constants are data items that do not change, no matter where they appear in a database or within the computer. They are literal values because they are exactly what they represent. Examples are numerals such as 3 and the logical values T and F.

Characters and character strings (all the printable characters plus spaces) can also be constants, but must be handled a bit differently.

"Strings" are simply a collection of characters (including spaces, digits and symbols) handled, modified, manipulated and otherwise used as data. A "substring" is a portion of any specific string.

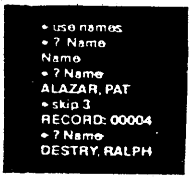
If a character or collection of characters is to be treated as a string constant, it must be enclosed in single or double quotes or in square brackets so the computer understands that it is to deal with the characters as characters. To see what we mean, get dBASE II up on your computer and USE <Names>. Type:

```
^dBASE^
^USE Names^
^? 'Name'^
^? Name^
```

In response to the first "What is..." (the ^?^ command), the computer responded with NAME because that was the value of the constant. When you eliminated the single quotes, the computer first checked to see if the word was a command. It wasn't, so it then checked to see if it was the name of a variable.

Variables are data items that can change. Frequently they are the names of database fields whose contents can change. In this case, the computer found that our database had a field called <Name> so it gave us the data that was in that field at that time. Type the following:

```
^SKIP 3^
^? Name^
```


```
• use names
• ? Name
Name
• ? Name
ALAZAR, PAT
• skip 3
RECORD: 00004
• ? Name
DESTRY, RALPH
```

Now type ^USE^. Since we do not specify a file n. the computer simply closes all files.

If we type ^? **Name**^ again, the computer tells us that we made an error. In this case, we tried to use a variable that did not exist because we were no longer USing a file with a matching field name.

The variables can also be **memory variables** rather than field names. dBASE II reserves an area of memory for storing up to 64 variables, each with a maximum length of 254 characters, but with a maximum total of 1536 characters for all the variables.

You might want to think of this as a series of 64 pigeon-holes available for you to tuck data into temporarily while working out a problem.

Variable names can be any legal dBASE II identifier (start with a letter, up to ten characters long, optional embedded colon and numbers, no spaces).

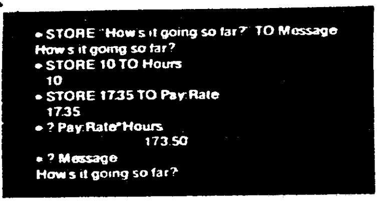
You can use a memory variable for storing temporary data or for keeping input data separate from field variables. In one session, for example, we might "tuck" the date into a pigeon-hole (variable) called <Date>. During the session, we could get it by asking for <Date>, then place it into any date field in any database without having to re-enter it (see GetDate.CMD in Section VI).

To get data (character, numeric or logical) into a memory variable, you can use the ^STORE^ command. The full form is:

^STORE <expression> TO <memory variable>^

Type the following:

```
^STORE "How's it going so far?" TO Message
^STORE 10 TO Hours^
^STORE 17.35 TO Pay:Rate^
^? Pay:Rate*Hours^
^? Message^
```

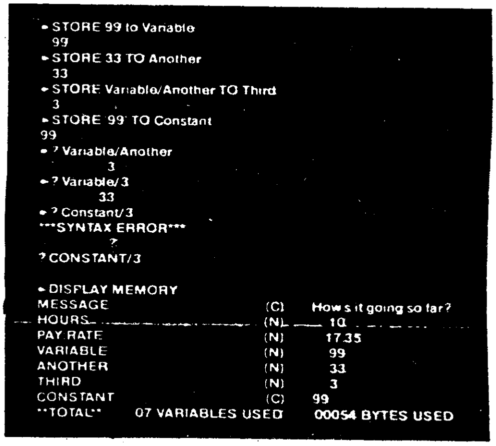


```
. STORE "How's it going so far?" TO Message
How's it going so far?
. STORE 10 TO Hours
  10
. STORE 17.35 TO Pay:Rate
  17.35
. ? Pay:Rate*Hours
          173.50
. ? Message
How's it going so far?
```

Notice that we used double quotes around the character string (a constant) in the first line because we wanted to use the single quote as an apostrophé inside the string.

If this isn't clear yet, try experimenting with and without the quotes to get the distinction between constants and variables. To start you off, type the following:

```
^STORE 99 TO Variable^
^STORE 33 TO Another^
^STORE Variable/Another TO Third^
^STORE '99' TO Constant^
^? Variable/Another^
^? Variable/3^
^? Constant/3^
^DISPLAY MEMORY
```



```
• STORE 99 to Variable
  99
• STORE 33 TO Another
  33
• STORE Variable/Another TO Third
  3
• STORE '99' TO Constant
99
• ? Variable/Another
         3
• ? Variable/3
        33
• ? Constant/3
***SYNTAX ERROR***
         ?
? CONSTANT/3

• DISPLAY MEMORY
MESSAGE                      (C)   How's it going so far?
HOURS                        (N)    10
PAY RATE                     (N)    17.35
VARIABLE                     (N)    99
ANOTHER                      (N)    33
THIRD                        (N)    3
CONSTANT                     (C)   99
**TOTAL**     07 VARIABLES USED    00054 BYTES USED
```

Entering a value into a variable automatically tells dBASE II what the data type is. From then on, you cannot mix data types (by trying to divide a character string by a number, for instance.)

RULES: Character strings that appear in expressions must be enclosed in <u>matching</u> single or double quote marks or square brackets. Character strings may contain any of the printable characters (including the space) . If you want to use the ampersand (&) as a character, it <u>must</u> be between two spaces because it is also used for the dBASE II macro function (described later).

The last command in the previous screen representation is another form of ^DISPLAY^ that you'll find useful. (You can also ^LIST MEMORY^.)

You can eliminate a memory variable by typing ^RELEASE <name>^, or you can get rid of all the memory variables by typing ^RELEASE ALL^.

Type the following (you may want to ^ERASE^ the screen first):

```
^DISPLAY MEMORY^
^RELEASE Another^
^DISPLAY MEMORY^
^RELEASE ALL^
^DISPLAY MEMORY^
```

Tip: When naming any variables, try to use as many characters as necessary to make the name meaningful to humans.

Another tip: If you use only nine characters for database field names, when you want to use the name as a memory variable, you can do so by putting an "M" in front of it. What it stands for will be clearer when you come back to clean up your programs later than if you invented a completely new and different name.

dBASE II operators

Operators are manipulations that dBASE II performs on your data. Some of them will be familiar; others may take a bit of practice.

Arithmetic operators should be tne most familiar. They generate arithmetic results.

```
() : parentheses for grouping
*  : multiplication
/  : division
+  : addition
-  : subtraction
```

The **arithmetic operators** are evaluated in a sequence of precedence. The order is: parentheses; multiply and divide; add and subtract. When the operators have equal precedence, they are evaluated from left to right. Here are some examples:

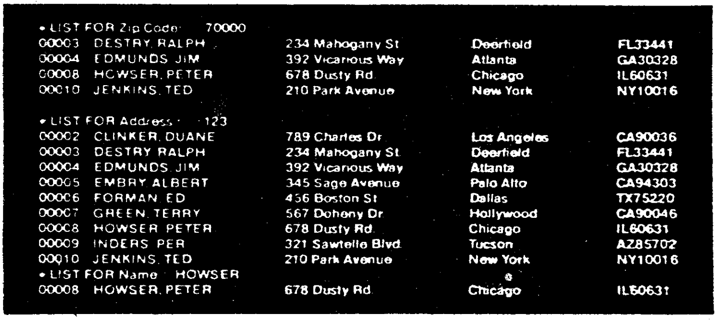
```
17/33*72 + 8 = 45.09     (divide, multiply then add)
17/(33*72 +8) = 0.00644  (multiply,add then divide)
17/33*(72 +8) = 41.21    (divide, add then multiply)
```

Relational operators make comparisons, then generate logical results. They take action based on whether the comparison is True or False.

```
<  : less than
>  : greater than
=  : equal to
<> : not equal to
<= : less than or equal to
>= : greater than or equal to
```

Type the following:

```
^USE Names^
^LIST FOR Zip:Code <= '70000'^
^LIST FOR Address <> '123'^
^LIST FOR Name = 'HOWSER'^
```



```
. LIST FOR Zip:Code<=70000
00003  DESTRY, RALPH      234 Mahogany St       Deerfield     FL33441
00004  EDMUNDS, JIM       392 Vicarious Way     Atlanta       GA30328
00008  HOWSER, PETER      678 Dusty Rd.         Chicago       IL60631
00010  JENKINS, TED       210 Park Avenue       New York      NY10016

. LIST FOR Address<>123
00002  CLINKER, DUANE     789 Charles Dr        Los Angeles   CA90036
00003  DESTRY RALPH       234 Mahogany St.      Deerfield     FL33441
00004  EDMUNDS, JIM       392 Vicarious Way     Atlanta       GA30328
00005  EMBRY ALBERT       345 Sage Avenue       Palo Alto     CA94303
00006  FORMAN, ED         456 Boston St         Dallas        TX75220
00007  GREEN, TERRY       567 Doheny Dr.        Hollywood     CA90046
00008  HOWSER PETER       678 Dusty Rd.         Chicago       IL60631
00009  INDERS PER         321 Sawtelle Blvd.    Tucson        AZ85702
00010  JENKINS, TED       210 Park Avenue       New York      NY10016
. LIST FOR Name  HOWSER
00008  HOWSER, PETER      678 Dusty Rd.         Chicago       IL60631
```

The logical operators greatly expand the ability to refine data and manipulate records and databases. Explaining them in depth is beyond the scope of this manual, but if you are not familiar with them, most computer texts have a chapter very near the beginning that explains their use. They generate logical results (True or False). They are listed below in the order of precedence within an expression (.NOT. is applied before .AND., etc.):

()	: parentheses for grouping
.NOT.	: boolean not (unary operator)
.AND.	: boolean and
.OR.	: boolean or
$	: substring logical operator (substring search)

```
^LIST FOR (JobNumber=730 .OR. JobNumber=731);
          AND. (Bill:Date >= '791001' .AND.;
                Bill:Date <= '791031')^
```

displays all the October, 1979 records for costs billed against job numbers 730 and 731 (notice how the command line was extended with the semi-colons).

If you're not familiar with logical operators, start with the basic fact that these operators will give results that are True or False. In our example, dBASE II asks the following questions about each record:

1) Is JobNumber equal to 730 (T or F)?
2) Is JobNumber equal to 731 (T or F)?
3) Is Bill:Date greater than or equal to '791001' (T or F)?
4) Is Bill:Date less than or equal to '791031' (T or F)?

dBASE II then performs three logical tests (.OR., .AND., .AND.) before deciding whether the record should be displayed or not.

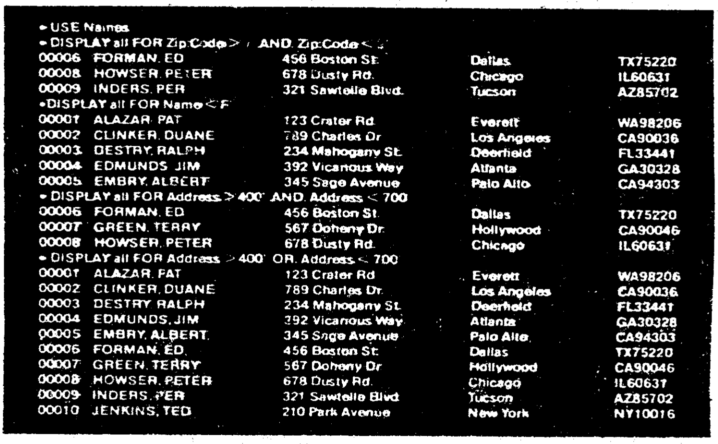
Parentheses are used as they would be in an arithmetic expression to clarify operations and relations. Because of the first .AND., dBASE II will display records only when the conditions in both parenthetical statements are true.

Evaluating the first expression, it first checks the <Job:Number> field. If the value in the field is 730 or 731, this sub-expression is set to True. If the field contains some other value, this sub-expression is False and the record will not be displayed.

If the first sub-expression is true, dBASE II must still check the contents of the <Bill:Date> field to evaluate the second sub-expression. If the contents of the field are between '791001' and '791031, inclusive, this expression is true, too, and the record will be displayed. Otherwise, the complete expression is false and dBASE II will skip to the next record, where it proceeds through the same evaluation.

Let's try some of this with <Names.DBF>. Type the following:

```
^USE Names^
^DISPLAY all FOR Zip:Code > '5' .AND. Zip:Code < 9^
^DISPLAY all FOR Name < 'F'^
^DISPLAY all FOR Address > '400' .AND. Address < '700'^
^DISPLAY all FOR Address > '400' .OR. Address < '700'^
```



```
• USE Names
• DISPLAY all FOR Zip:Code > ' AND. Zip:Code < '
00006  FORMAN, ED       456 Boston St.       Dallas        TX75220
00008  HOWSER, PETER    678 Dusty Rd.        Chicago       IL60631
00009  INDERS, PER      321 Sawtelle Blvd.   Tucson        AZ85702
• DISPLAY all FOR Name < F
00001  ALAZAR, PAT      123 Crater Rd.       Everett       WA98206
00002  CLINKER, DUANE   789 Charles Dr       Los Angeles   CA90036
00003  DESTRY, RALPH    234 Mahogany St.     Deerfield     FL33441
00004  EDMUNDS, JIM     392 Vicarious Way    Atlanta       GA30328
00005  EMBRY, ALBERT    345 Sage Avenue      Palo Alto     CA94303
• DISPLAY all FOR Address > 400' AND. Address < 700
00006  FORMAN, ED       456 Boston St.       Dallas        TX75220
00007  GREEN, TERRY     567 Doheny Dr.       Hollywood     CA90046
00008  HOWSER, PETER    678 Dusty Rd.        Chicago       IL60631
• DISPLAY all FOR Address > 400' OR. Address < 700
00001  ALAZAR, PAT      123 Crater Rd        Everett       WA98206
00002  CLINKER, DUANE   789 Charles Dr.      Los Angeles   CA90036
00003  DESTRY, RALPH    234 Mahogany St.     Deerfield     FL33441
00004  EDMUNDS, JIM     392 Vicarious Way    Atlanta       GA30328
00005  EMBRY, ALBERT    345 Sage Avenue      Palo Alto     CA94303
00006  FORMAN, ED       456 Boston St.       Dallas        TX75220
00007  GREEN, TERRY     567 Doheny Dr        Hollywood     CA90046
00008  HOWSER, PETER    678 Dusty Rd.        Chicago       IL60631
00009  INDERS, PER      321 Sawtelle Blvd    Tucson        AZ85702
00010  JENKINS, TED     210 Park Avenue      New York      NY10016
```

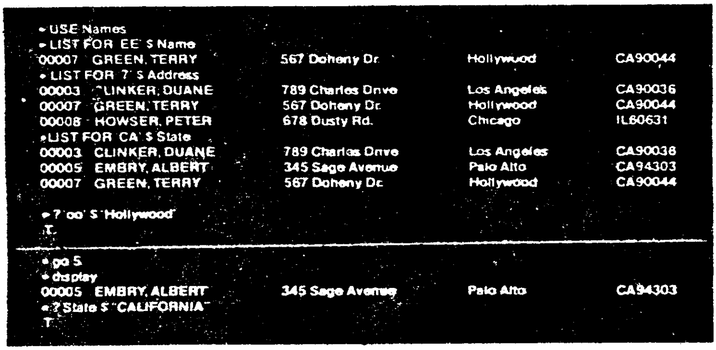
Notice what happened with the last command: all the records were displayed. If you're not familiar with logical operators, this kind of non-selective "selection" will have to guarded against.

The $ substring logical operator is extremely useful because of its powerful search capabilities. The format is:

```
^<substring> $ <string>^
```

This operator searches for the substring on the left within the string on the right. Either or both terms may be string variables as well as string constants. To see how this works, type the following:

```
^USE Names^
^LIST FOR 'EE' $ Name^
^LIST FOR '7' $ Address^
^LIST FOR 'CA' $ State^
^? 'oo' $ 'Hollywood'^
^. GO 5^
^: DISPLAY^
^? State $ "CALIFORNIA"^
```



```
• USE Names
• LIST FOR 'EE' $ Name
00007   GREEN, TERRY        567 Doheny Dr.       Hollywood       CA90044
• LIST FOR '7' $ Address
00003   CLINKER, DUANE      789 Charles Drive    Los Angeles     CA90036
00007   GREEN, TERRY        567 Doheny Dr.       Hollywood       CA90044
00008   HOWSER, PETER       678 Dusty Rd.        Chicago         IL60631
• LIST FOR 'CA' $ State
00003   CLINKER, DUANE      789 Charles Drive    Los Angeles     CA90036
00005   EMBRY, ALBERT       345 Sage Avenue      Palo Alto       CA94303
00007   GREEN, TERRY        567 Doheny Dr.       Hollywood       CA90044

• ? 'oo' $ 'Hollywood'
.T.
• go 5
• display
00005   EMBRY, ALBERT       345 Sage Avenue      Palo Alto       CA94303
• ? State $ "CALIFORNIA"
.T.
```

With this function we could have, for example, simplified the structure of our mailing list names file. The states could have been entered as part of the address. To call out names within a specific state, we could have simply typed the following, where XX is the abbreviation for the state we want:

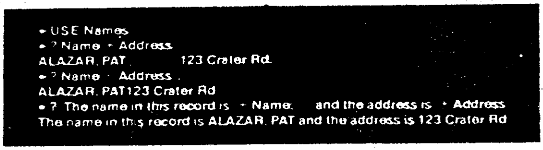
```
^<COMMAND> FOR 'XX' $ Address^
```

<u>String operators</u> generate string results.

+ = string concatenation (exact)
- = string concatenation (moves blanks)

Concatenation is just another one of those fancy computer buzzwords. All it really means is that one character string is stuck on to the end of another one. Type the following:

```
^USE Names^
^? Name + Address^
^? Name - Address^
^? 'The name in this record is ' + Name;
  - ' and the address is '+ Address^
```



```
• USE Names
• ? Name + Address
ALAZAR, PAT           123 Crater Rd.
• ? Name - Address .
ALAZAR, PAT123 Crater Rd
• ? The name in this record is  + Name      and the address is  + Address
The name in this record is ALAZAR, PAT and the address is 123 Crater Rd
```

The ^+^ and ^-^ both join two strings. The "plus" sign joins the string exactly as they are found. The "minus" sign moves the trailing blanks in a string to the end of the string. They are not eliminated, but for many purposes this is enough, as they do not show up between the strings being joined.

If you want to eliminate the trailing blanks, you can use the ^TRIM^ function. This is used by typing ^**STORE TRIM(<variable>) TO <variable>**^. As an example, we could have typed:^STORE TRIM(**Name**) TO (**Name**)^ to eliminate the blanks following the characters of the name.

To eliminate all of the trailing blanks in our example, we could have typed: ^STORE TRIM(Name - Address) TO Example^.

Now that we've introduced you to expressions and dBASE II operators, we'll continue with other dBASE II commands. We'll be giving you some practice in using expressions and operators as we work our way up to develpping command files.

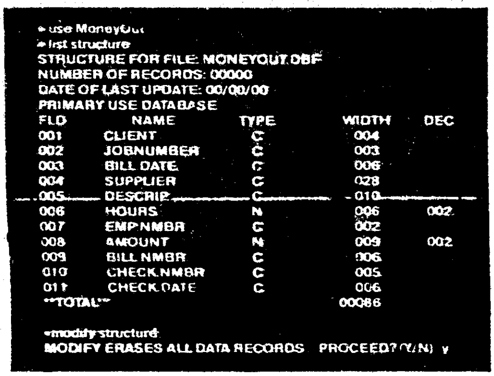
Changing an empty database structure (MODIFY)

WARNING: the ^MODIFY^ command will destroy your database. Please follow instructions carefully.

When there is no data in your database, the **^MODIFY^** command is the fastest and easiest way to add, delete, rename, resize or otherwise change the database structure. This destroys any data in the database so don't use it after you've entered data. (Later we'll show you a way to do so, safely.)

<MoneyOut.DBF> has no data in it yet, so we'll work with it. A useful change would be to rename <JobNumber> to <Job:Nmbr> so that the abbreviation is consistent with <Emp:Nmbr> and <Bill:Nmbr>. Type the following:

^USE MoneyOut^
^LIST STRUCTURE^ (page 22)
^MODIFY STRUCTURE^
^y^ (in response to the question)



```
• use MoneyOut
• list structure
STRUCTURE FOR FILE: MONEYOUT.DBF
NUMBER OF RECORDS: 00000
DATE OF LAST UPDATE: 00/00/00
PRIMARY USE DATABASE
FLD        NAME        TYPE      WIDTH    DEC
001      CLIENT         C         004
002      JOBNUMBER      C         003
003      BILL DATE      C         006
004      SUPPLIER       C         028
005      DESCRIP        C         010
006      HOURS          N         006      002
007      EMP:NMBR       C         002
008      AMOUNT         N         009      002
009      BILL:NMBR      C         006
010      CHECK:NMBR     C         005
011      CHECK:DATE     C         006
**TOTAL**                        00086

•modify structure
MODIFY ERASES ALL DATA RECORDS   PROCEED? (Y/N) y
```

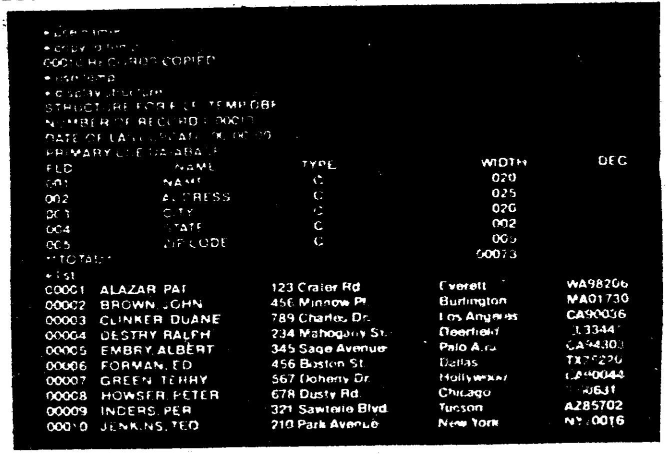
dBASE II erases the screen and lists the first 16 (or fewer) fields in the database. Use ^Ctl-X^ to move down one field. Just type in the new field name over the old one (use a space to blank out the extra letter).

You can exit ^MODIFY^ in either of two ways: **ctl-W** changes the structure on disk, then resumes normal dBASE II operation (^ctl-O^ for Superbrain). **ctl-Q** quits and returns to normal dBASE II operation without making the changes. This actually gets you back without destroying the database, but play it safe and have a backup file (see next page).

Duplicating databases and structures (COPY)

Duplicating a file without going back to your computer operating system is straightforward. Type the following:

^USE Names^
^COPY TO Temp^
^USE Temp^
^DISPLAY STRUCTURE^
^LIST^



```
• use names
• copy to temp
00010 RECORDS COPIED
• use temp
• display structure
STRUCTURE FOR FILE  TEMP.DBF
NUMBER OF RECORDS  00010
DATE OF LAST UPDATE  00/00/00
PRIMARY USE DATABASE
FLD          NAME          TYPE               WIDTH        DEC
001          NAME           C                  020
002          ADDRESS        C                  025
003          CITY           C                  020
004          STATE          C                  002
005          ZIP CODE       C                  005
** TOTAL **                                   00073
• list
00001  ALAZAR PAT        123 Crater Rd        Everett        WA98206
00002  BROWN, JOHN       456 Minnow Pl.       Burlington     MA01730
00003  CLINKER DUANE     789 Charles Dr.      Los Angeles    CA90036
00004  DESTRY RALPH      234 Mahogany St.     Deerfield      [illegible]13344
00005  EMBRY, ALBERT     345 Sage Avenue      Palo Alto      CA94303
00006  FORMAN, ED        456 Boston St.       Dallas         TX75220
00007  GREEN TERRY       567 Doheny Dr.       Hollywood      CA90044
00008  HOWSER, PETER     678 Dusty Rd.        Chicago        [illegible]60631
00009  INDERS, PER       321 Sawtelle Blvd.   Tucson         AZ85702
00010  JENKINS, TED      210 Park Avenue      New York       NY10016
```

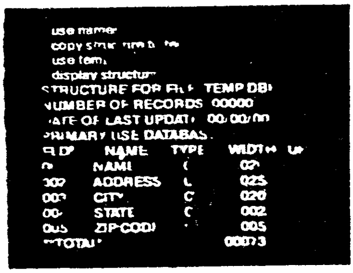
Warning: When you ^COPY^ to an existing filename, the file is written over and the old data is destroyed.

^COPY TO TEMP^ created a new database called <Temp.DBF>. It is identical to the <Names.DBF>, with the same structure and the same data. The command can be expanded even further:

^COPY TO <filename> [STRUCTURE] [FIELD list]

With this command, you can copy <u>only</u> the structure or <u>some</u> of the structure to another file. Type the following:

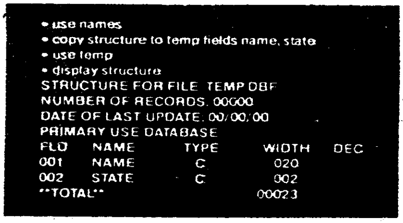
^USE Names^
^COPY TO Temp STRUCTURE^
^USE Temp^
^DISPLAY STRUCTURE^



```
 use names
 copy structure to temp
 use temp
 display structure
STRUCTURE FOR FILE  TEMP.DBF
NUMBER OF RECORDS  00000
DATE OF LAST UPDATE  00/00/00
PRIMARY USE DATABASE
FLD       NAME       TYPE     WIDTH   DEC
001       NAME        C        020
002       ADDRESS     C        025
003       CITY        C        020
004       STATE       C        002
005       ZIP CODE    C        005
** TOTAL **                   00073
```

We can copy a portion of the structure by listing only the fields we want in the new database. Type:

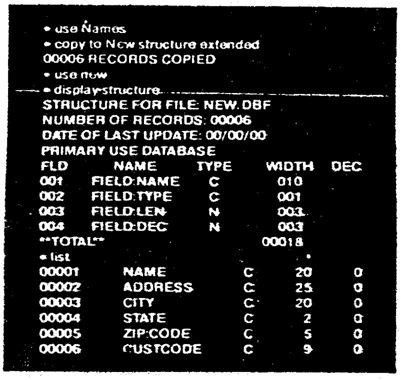
^USE Names^
^COPY TO Temp STRUCTURE FIELDS Name, State^
^USE Temp^
^DISPLAY STRUCTURE^



```
• use names
• copy structure to temp fields name, state
• use temp
• display structure
STRUCTURE FOR FILE  TEMP.DBF
NUMBER OF RECORDS: 00000
DATE OF LAST UPDATE: 00/00/00
PRIMARY USE DATABASE
FLD      NAME        TYPE       WIDTH     DEC
001      NAME          C         020
002      STATE         C         002
**TOTAL**                      00023
```

FOR ADVANCED PROGRAMMERS: COPY can also be used to give your program access to a database structure. Type:

^USE Names^
^COPY TO New STRUCTURE EXTENDED^
^USE New^
^LIST^



```
• use Names
• copy to New structure extended
00006 RECORDS COPIED
• use new
• display structure
STRUCTURE FOR FILE: NEW.DBF
NUMBER OF RECORDS: 00006
DATE OF LAST UPDATE: 00/00/00
PRIMARY USE DATABASE
FLD       NAME        TYPE      WIDTH    DEC
001    FIELD:NAME       C        010
002    FIELD:TYPE       C        001
003    FIELD:LEN        N        003
004    FIELD:DEC        N        003
**TOTAL**                       00018
• list
00001        NAME             C      20      0
00002        ADDRESS          C      25      0
00003        CITY             C      20      0
00004        STATE            C       2      0
00005        ZIP:CODE         C       5      0
00006        CUSTCODE         C       9      0
```

The <New.DBF> database records describe the <Names> database structure, and an application program has direct access to this information (see Review.CMD, Section VI).

Alternatively, a file with the same structure as <New.DBF> could be embedded in a program so that the operator could enter the structure for a file without learning dBASE II. The program would then create the database for him with the following command:

^CREATE <datafile> FROM <structurefile>^

Adding and deleting fields with data in the database

As you expand the applications for dBASE II, you'll probably want to add or delete fields in your databases.

^**MODIFY STRUCTURE**^ alone would destroy all the data in your database, but used with ^**COPY**^ and ^**APPEND**^, it lets you add and delete fields at will.

The strategy consists of copying the structure of the database you want to change to a temporary file, then making your modifications on that file. After that is done, you bring in the data from the old file into the new modified structure.

As an example, we'll use our <Names> file and our <Orders> file. At some point, it would be useful to list the orders placed by a given customer. This could be done easily by adding a customer number field to <Names> file to match the field in the <Orders> file. To do so without destroying the records we already have, type the following:

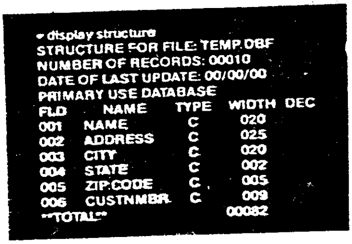
```
^USE Names^
^COPY TO Temp STRUCTURE^
^USE Temp^
^MODIFY STRUCTURE^
^y^                      (in answer to the prompt.
```

Use the Full Screen Editing features to move down to the first blank field and type in the changes in the appropriate columns (name is "CustNmbr", data type is "C", length is 9). Now type ^ctl-W^ (^ctl-O^ with Superbrain) to save the changes and exit to the dBASE II dot prompt.

^**DISPLAY STRUCTURE**^ to make sure that it's right. If it is we can add the data from <Names> by typing:

```
^APPEND FROM Names^
```

We could also have changed field sizes: the ^**APPEND**^ command transfers data to fields with matching names.



```
• display structure
STRUCTURE FOR FILE: TEMP.DBF
NUMBER OF RECORDS: 00010
DATE OF LAST UPDATE: 00/00/00
PRIMARY USE DATABASE
FLD      NAME        TYPE   WIDTH  DEC
001    NAME            C      020
002    ADDRESS         C      025
003    CITY            C      020
004    STATE           C      002
005    ZIP:CODE        C      005
006    CUSTNMBR        C      009
**TOTAL**                    00082
```

Our new file <Temp> should now have the new field we wanted to add and all of the old data. ^DISPLAY STRUCTURE^ then ^LIST^ to make sure that a power line glitch or a bad spot on the floppy hasn't messed anything up.

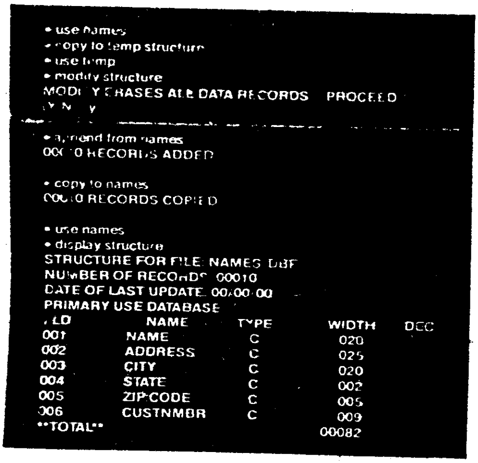
If the data got transferred correctly, we can finish up by typing:

```
^COPY TO Names
^USE Names^
```

The ^COPY^ command writes over the old structure and data. After displaying and listing the new <Names> file, you can ^DELETE FILE Temp^.

To summarize, the procedure can be used to add or delete fields in a database in the following sequence:

```
^USE <oldfile>^
^COPY TO <newfile> STRUCTURE^
^USE <newfile>^
^MODIFY STRUCTURE^
^APPEND FROM <oldfile>^
^COPY TO <oldfile>^
```



```
• use names
• copy to temp structure
• use temp
• modify structure
MODIFY ERASES ALL DATA RECORDS    PROCEED
(Y/N)  y

• append from names
00010 RECORDS ADDED

• copy to names
00010 RECORDS COPIED

• use names
• display structure
STRUCTURE FOR FILE: NAMES.DBF
NUMBER OF RECORDS: 00010
DATE OF LAST UPDATE: 00/00/00
PRIMARY USE DATABASE
FLD        NAME       TYPE     WIDTH    DEC
001       NAME         C        020
002       ADDRESS      C        025
003       CITY         C        020
004       STATE        C        002
005       ZIP:CODE     C        005
006       CUSTNMBR     C        009
**TOTAL**                     00082
```

Dealing with CP/M and other "foreign" data files (more on COPY and APPEND)

dBASE II information can be changed into a form that is compatible with other processors and systems (BASIC, PASCAL, FORTRAN, PL/1, etc.). dBASE II can also read data files that have been created by these processors.

With CP/M, the standard data format (abbreviated as SDF in dBASE II) includes a carriage return and line feed after every line of text. To create a compatible data file (for wordprocessing, for example) from one of your databases, you use another form of the ^COPY^ command. Type:

```
^USE Names^
^COPY TO SysData SDF^
```

This command creates a file called <SysData.TXT>. Now ^QUIT^ dBASE II and use your word processor to look at the file. You'll find that you can work with it exactly as if you had created it under CP/M.

The Standard Data Format also allows dBASE II to work with data from CP/M files. However, the data must match the structure of the database that will be using it.

If we had used a wordprocessor to create a file called <NewData.TXT>, we could add it to the <Names.DBF> file with this command. NOTE: the spacing of the data must match the structure of the database. If the <NewData.TXT> file contained the following information:

```
FREITAG, JEAN          854 Munchkin Ave.          Houston                TX77006
GOULD, NICOLE          73 Radnor Way              Radnor                 PA19089
PETERS, ALICE          676 Wacker Dr.             Chicago                IL60606
GREEN, FRANK           441 Spicer Ave.            Tampa                  FL33622

      (20)                   (25)                  (20)                  (2) (5)
```

we would add it to the <Names> file by typing the following:

```
^USE Names^
^APPEND FROM NewData.TXT SDF^
```

Adding data to an existing file from a system file takes only seconds.

The procedure is similar if your "foreign" files use different delimiters. A common data file format uses commas between fields and single quotes around strings to delimit the data. To create or use these types of data files, use the word **DELIMITED** instead of **SDF**. To see how this works, type:

^COPY TO Temp DELIMITED^

then go back to your operating system to look at your data.

If your system has a different delimiter, you can specify it in the command: **^DELIMITED [WITH <delimiter>]^** (do NOT type the "<" and ">" symbols). If your system uses only commas and nothing around strings, use: **^DELIMITED WITH ,^**.

The full forms of **^COPY^** and **^APPEND^** for working with system data files are:

```
                                                [SDF       ]
COPY [scope] TO <filename> [FIELD list] [STRUCTURE ]  [FOR <expression>]
                                                [DELIMITED [WITH <delimiter>]]

APPEND FROM <filename.TXT> [SI          ]  [FOR <expression>]
                           [DELIMITED [WITH <delimiter>]]
```

Both commands can be made selective by using a conditional expression, and the scope of ^COPY^ can be specified as for other dBASE II commands.

NOTE: While dBASE II automatically generates extensions for files it creates, you must specify the ".TXT" filename extension when APPENDing from a system data file.

NOTE: With the APPEND command, any fields used in the <expression> must exist in the database to which the data is being transferred.

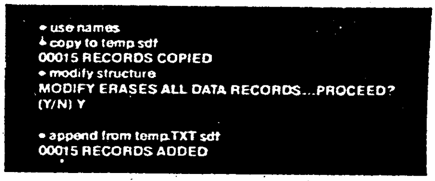
Renaming database fields with COPY and APPEND

As we sad earlier, ^**APPEND**^ transfers data from one file to another for matching fields. If a field name in the FROM file is not in the file in USE, the data in that field will not be transferred.

However, the full form does allow you to transfer only data, and we can use this feature to rename the fields in a database. If we wanted to rename <CustNmbr> to <CustCode> in <Names.DBF>, we would type:

```
^USE Names^
^COPY TO Temp SDF^              (data only to Temp.TXT)
^MODIFY STRUCTURE^
^APPEND FROM Temp.TXT SDF^      (after changing field name)
```

Now when you ^**DISPLAY STRUCTURE**^, the last field will be called <CustCode>. Don't forget to change the name of the <CustNmbr> field in our <Orders> database so that the fields match.



```
• use names
• copy to temp sdf
00015 RECORDS COPIED
• modify structure
MODIFY ERASES ALL DATA RECORDS...PROCEED?
(Y/N) Y

• append from temp.TXT sdf
00015 RECORDS ADDED
```

Data in a <.TXT> file created by using the **SDF** (or **DELIMITED**) option is kept in columns that are spaced like the fields were in the original file. While you can edit a <.TXT> file with your word processor, this can be dangerous:

<u>Warning</u>: Do not change **field positions or sizes**: the data you saved is saved by <u>position</u>, not by name! If you change the field sizes when you modify the structure, you will destroy your database when you bring the saved data back into it.

When you ^**COPY**^ data to a <.TXT> file, you can use the full command to specify the <u>scope</u>, <u>fields</u> and <u>conditions</u> (see earlier explanation).

Modifying data rapidly (REPLACE, CHANGE)

Changes can be made rapidly to any or all of the records using the following command:

^REPLACE [scope] <field> WITH <data> [, <field> WITH <data>,...]
[FOR <expression>]^

This is an extremely powerful command because it REPLACES a "<field-that-you-name> WITH <whatever-you-write-in-here>". You can REPLACE more than one field by using a comma after the first combination, then listing the new fields and data as shown in the center brackets.

The "data" can be specific new information (including blanks), or it could be an operation, such as deducting state sales tax from all your bills because you have a resale number (REPLACE all Amount WITH Amount/1.06).

You can also make this replacement conditional by using the FOR and specifying your conditions as an expression.

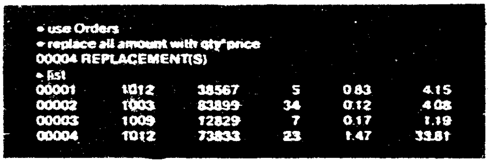
To show you how this works, we need to add some data to both the <Names> and <Orders> database files.

First, **^USE Name^** then type **^EDIT 1^**. Now enter a ^1001^ in the <CustCode> field, using the full screen editing features to get into position. Use **^ctl-C^** to move on to the next record when you are finished customer codes should be entered as four-digit numbers, with the record number as the last two digits (1001, 1002, 1003, etc.)

Now ^USE Orders^ and **^APPEND^** the following order information (do not type the column headings):

```
(Cust) (Item)   (Qty)  (Price)^
^1012   38567      5      .83^
^1003   83899     34      .12^
^1009   12829      7      .17^
^1012   73833     23     1.47^
```

^USE Orders^
^REPLACE All Amount WITH Qty*Price
^LIST^



```
• use Orders
• replace all amount with qty*price
00004 REPLACEMENT(S)
• list
00001      1012      38567      5      0.83      4.15
00002      1003      83899     34      0.12      4.08
00003      1009      12829      7      0.17      1.19
00004      1012      73833     23      1.47     33.81
```

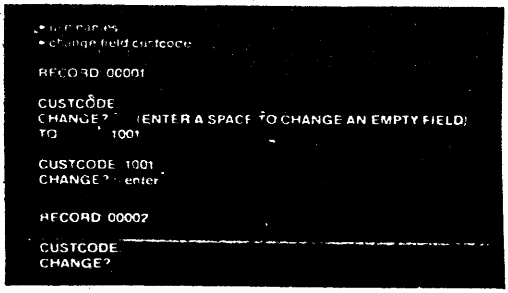
You'll also find ^REPLACE^ useful in command files to fill in a blank record that you have appended to a file. Data from memory variables in your program is frequently used to fill in the blank fields.

Changes to a few fields in a large number of records can also be made rapidly by using:

^CHANGE [scope] FIELD <list> [FOR <expression>]^

The "scope" is the same as for other dBASE II commands. At least one field must be named, but several field names can be listed if separated by commas. This command finds the first record that meets the conditions in the "expression", then displays the record name and contents with a prompt. To change the data in the field, type in the new information. To leave it the way it was, hit <enter>. If the field is blank and you want to add data, type a space.

Once you have looked at all the listed fields within a record, you are presented with the first field of the next record that meets the conditions you set. To return to dBASE II, hit the ^ESCAPE^ key.



```
• use names
• change field custcode

RECORD 00001

CUSTCODE
CHANGE?      (ENTER A SPACE TO CHANGE AN EMPTY FIELD)
TO          1001

CUSTCODE  1001
CHANGE?   enter

RECORD 00002

CUSTCODE
CHANGE?
```

Organizing your databases (SORT, INDEX)

Data is frequently entered randomly, as it was in our <Names> database. This not necessarily the way you want it, so dBASE II includes tools to help you organize your databases by **SORTING** and **INDEXING** it.

INDEXED files allow you locate records quickly (typically within two seconds even with floppy disks).

Files can be sorted in ascending or descending order. The full command is:

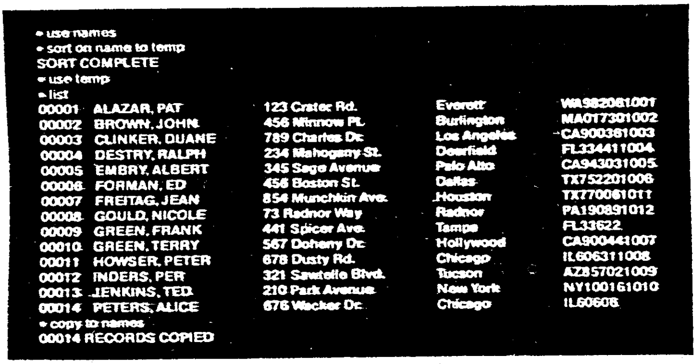
```
^SORT ON <fieldname> TO <filename> [DESCENDING]
```

The <fieldname> specifies the key on which the file is sorted and may be character or numeric (not logical). The sort defaults to ascending order, but you can over-ride this by specifying the descending option.

To sort on several keys, start with the least important key, then use a series of sorts leading up to the major key. During sorting, dBASE II will move only as many records as it must.

To sort our <Names> file so that the customers are in alphabetical order, type:

```
^USE Names^
^SORT ON Name TO Temp^
^USE Temp^
^LIST^
^COPY TO Names^
```



```
• use names
• sort on name to temp
SORT COMPLETE
• use temp
• list
00001  ALAZAR, PAT        123 Crater Rd.       Everett       WA982081001
00002  BROWN, JOHN        456 Minnow Pl.       Burlington    MA017301002
00003  CLINKER, DUANE     789 Charles Dr.      Los Angeles   CA900381003
00004  DESTRY, RALPH      234 Mahogany St.     Deerfield     FL334411004
00005  EMBRY, ALBERT      345 Sage Avenue      Palo Alto     CA943031005
00006  FORMAN, ED         456 Boston St.       Dallas        TX752201006
00007  FREITAG, JEAN      854 Munchkin Ave.    Houston       TX770061011
00008  GOULD, NICOLE      73 Radnor Way        Radnor        PA190891012
00009  GREEN, FRANK       441 Spicer Ave.      Tampa         FL33622
00010  GREEN, TERRY       567 Doheny Dr.       Hollywood     CA900441007
00011  HOWSER, PETER      678 Dusty Rd.        Chicago       IL606311008
00012  INDERS, PER        321 Sawtelle Blvd.   Tucson        AZ857021009
00013  JENKINS, TED       210 Park Avenue      New York      NY100161010
00014  PETERS, ALICE      676 Wacker Dr.       Chicago       IL60606
• copy to names
00014 RECORDS COPIED
```

WARNING: Do not SORT a database to itself. A power line "glitch" could destroy your entire database if it came along at the wrong moment.

Instead, sort to a temporary file, then ^COPY^ it back to the original file name after you've confirmed the data.

A database can also be INDEXED so that it appears to be sorted. The form of the ^INDEX^ command is:

```
^INDEX ON <key (variable/expression)> TO <index filename>^
```

This creates a file with the new name and the extension <.NDX>. Only the data within the "key" is sorted, although it appears that the entire database has been sorted. The key may be a variable name or a complex expression up to 100 characters long. It cannot be a logical field. To organize our customer database by ZIP code, type:

```
^USE Names^
^INDEX ON Zip:Code TO Zips^
^USE Names INDEX Zips^
^LIST^
```

We could also index our database on three keys by typing:

```
^INDEX ON Name + CustCode + State TO Compound^
```

Numeric fields used in this manner must be converted to character type. If CustCode were a numeric field with 5 positions and 2 decimal places, ^STR^ function (described later) performs the conversion like this:

```
^INDEX ON Name + STR(CustCode,5,2) + State TO Compound^
```

To take advantage of the speed built into an INDEX file, you have to specify it as part of the ^USE^ command:

```
^USE <database name> INDEX <index filename>^
```

Positioning commands (GO, GO BOTTOM, etc.) given with an INDEX file in use move you to positions on the index, rather than the database. ^GO BOTTOM^, for example, will position you at the last record in the index rather than the last record in the database.

Changes made to key fields when you ^APPEND^, ^EDIT^, ^REPLACE^ or ^PACK^ the database, are reflected in the index file in USE.

Other index files for your database can be updated by typing: ^SET INDEX TO <index File 1>, <index File 2>, ...<index File n>^. Then perform your ^APPEND^, ^EDIT^, etc. All named index files will now be current.

A major benefit of an INDEXED file is that it allows you to use the ^FIND^ command (described next) to locate records in seconds, even with large databases.

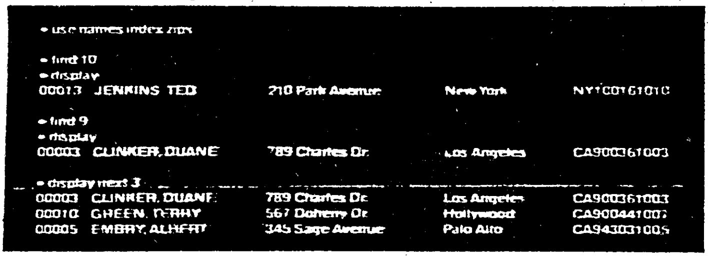
Finding the information you want (FIND, LOCATE)

If you know what data you are looking for, you can use the FIND command (but only when your database is indexed, and the index file is in USE). A typical FIND time is two seconds with a floppy disk system.

Simply type FIND <character string> (without quote marks), where the "character string" is all or part of the contents of a field.

This string can be as short as you like, but should be long enough to make it unique. "th", for example, occurs in a large number of words; "theatr" is much more limited. Type the following:

```
^USE Names INDEX Zips^
^FIND 10^
^DISPLAY^
^FIND 9^
^DISPLAY^
^DISPLAY Next 3^
```



```
• use names index zips

• find 10
• display
00013  JENKINS TED        210 Park Avenue      New York      NY1C0161010

• find 9
• display
00003  CLINKER, DUANE     789 Charles Dr.      Los Angeles   CA900361003

• display next 3
00003  CLINKER, DUANE     789 Charles Dr.      Los Angeles   CA900361003
00010  GREEN, TERRY       567 Doheny Dr.       Hollywood     CA900441007
00005  EMBRY, ALBERT      345 Sage Avenue      Palo Alto     CA943031005
```

If the key is not unique, dBASE II finds the first record that meets your specifications. This may or may not be the one you're looking for. If no record exists with the identical key that you are looking for, dBASE II displays NO FIND.

^FIND^ can also be used with files that have been INDEXED on multiple keys. The disadvantage of a compound key (which may not be a disadvantage in your application) is that it must be used from the left when you access the data. That is, you can access the data by using the FIND command and just the Name, or the Name and CustCode, or all three fields, but could not access it using the State or CustCode alone. To do that, you would either have to use the LOCATE command (next), or have another file indexed on the State field as the primary key.

When looking for specific kinds of data, use

^LOCATE [scope] [FOR <expression>]^

This command is used when you are looking for specific data in a file that is not indexed on the key you are interested in (file is indexed on zip codes, but you're interested in states, etc.)

If you want to search the entire database, you do not have to specify the scope, as ^LOCATE^ starts on the first record. To search part of a file, use ^LOCATE **Next <number>**^. The search will start at the record the pointer is on and look at the next "number" of records. If this would move the pointer past the end of the file, **LOCATE** examines every record from the pointer position to the end of the file.

If you are looking for data in a character field, the data should be enclosed in single quotes. Type the following:

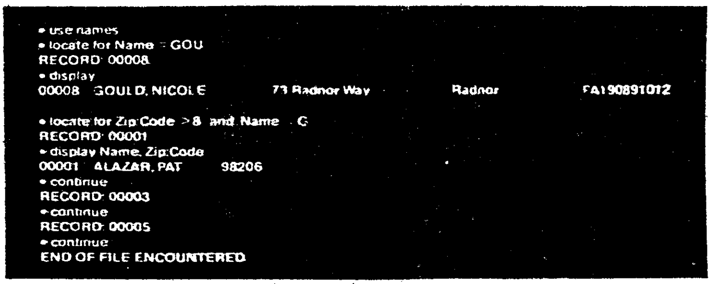
```
^USE Names^
^LOCATE FOR Name='GOU'^
^DISPLAY^
^LOCATE FOR Zip:Code>'8' .AND.  Name < 'G'^
^DISPLAY Name, Zip:Code^
```

If a record is found that meets the conditions in your expression, dBASE II signals you with: **RECORD n.** You can display or edit the record once it is located.

If there may be more than one record that meets your conditions, type **CONTINUE** to get the next record number.

```
^CONTINUE^
^CONTINUE^
^CONTINUE^
```

If dBASE II cannot find your record within the "scope" that you defined, it will display: **END OF LOCATE** or **END OF FILE ENCOUNTERED.**



```
• use names
• locate for Name = GOU
RECORD 00008
• display
00008   GOULD, NICOLE          73 Radnor Way          Radnor          PA190891012

• locate for Zip:Code >8  and Name  C
RECORD 00001
• display Name, Zip:Code
00001   ALAZAR, PAT       98206
• continue
RECORD 00003
• continue
RECORD 00005
• continue
END OF FILE ENCOUNTERED
```

Getting information out of all that data (the REPORT command)

FIND and **LOCATE** are fine for locating individual records and data items, but in most applications you will want data summaries that include many records that meet certain specifications. The ^**REPORT**^ command lets you do this quickly and easily.

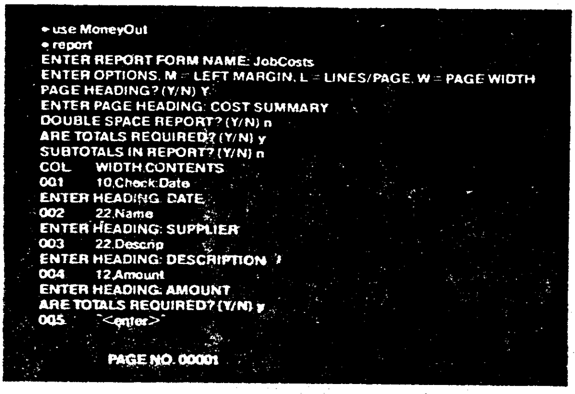
If you are using single sheets of paper in your printer, first type ^**SET EJECT OFF**^ to turn the initial formfeed off. Now select the database you want the report from and create your custom report format by typing:

```
^SET EJECT OFF^
^USE <database>^
^REPORT^
```

dBASE II then leads you through a series of prompts to create a custom format for the report. You specify which fields from the database you want, report and column headings, which columns should be totalled, etc. The standard defaults are 8 columns from the left edge of the paper for the page offset, 56 lines per page, and a page width of 80 characters.

You can try this with the files you've created on the demonstration disk, but the <Names> and <Orders> databases that we've used as examples so far don't have enough data in them to really show you how powerful dBASE II can be. For our examples from here on we will be using <MoneyOut.DBF> and other databases that are part of an existing business system. (The entire system is in Section VI, including database structures and the command files that run it.)

This would be a good time for you to create a database structure that you would actually use in your business. Enter data in it, then substitute it for <MoneyOut> in our examples.



```
• use MoneyOut
• report
ENTER REPORT FORM NAME: JobCosts
ENTER OPTIONS, M = LEFT MARGIN, L = LINES/PAGE, W = PAGE WIDTH
PAGE HEADING? (Y/N) Y
ENTER PAGE HEADING: COST SUMMARY
DOUBLE SPACE REPORT? (Y/N) n
ARE TOTALS REQUIRED? (Y/N) y
SUBTOTALS IN REPORT? (Y/N) n
COL     WIDTH,CONTENTS
001     10,Check:Date
ENTER HEADING: DATE
002     22,Name
ENTER HEADING: SUPPLIER
003     22,Descrip
ENTER HEADING: DESCRIPTION
004     12,Amount
ENTER HEADING: AMOUNT
ARE TOTALS REQUIRED? (Y/N) y
005     ^<enter>^

        PAGE NO. 00001
```

When you have defined all the contents of the report, hit <enter> when prompted with the next field number. dBASE II immediately starts the report to show you what you have specified, and will go through the entire database if you let it. To stop the report, hit the <escape> key.

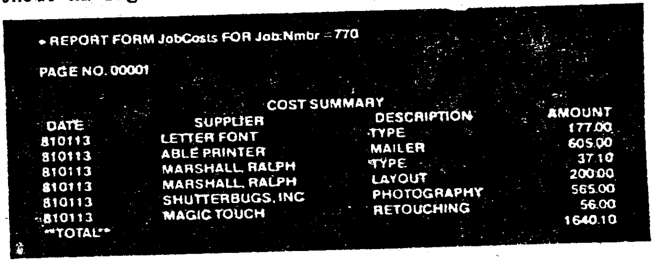
At the same time, dBASE II saves the format in a file with the extension .FRM, so that you can use it without having to go through the dialog again. The full form of the command is:

^REPORT FORM <formname> [scope] [FOR <expression>] [TO PRINT]^

By typing

^REPORT FORM JobCosts FOR Job:Nmbr='770'^

we can get a listing of all the job costs for job number 770 without having to redefine the format.



```
• REPORT FORM JobCosts FOR Job:Nmbr =770

PAGE NO. 00001

                          COST SUMMARY
 DATE          SUPPLIER              DESCRIPTION          AMOUNT
810113      LETTER FONT              TYPE                  177.00
810113      ABLE PRINTER             MAILER                605.00
810113      MARSHALL, RALPH          TYPE                   37.10
810113      MARSHALL, RALPH          LAYOUT                200.00
810113      SHUTTERBUGS, INC         PHOTOGRAPHY           565.00
810113      MAGIC TOUCH              RETOUCHING             56.00
 **TOTAL**                                                1640.10
```

You can change the information in the heading by typing ^**SET HEADING TO character string**^ (up to 60 characters and spaces, no quote marks). The "scope" defaults to "all" when not specified.

The expression could have been expanded with other conditions, and the entire report could have been prepared as a hardcopy by adding TO PRINT at the end of the command.

This report capability can be used for just about any business report, from accounts payable (FOR Check:Nmbr=' '), to auto expenses (FOR Job:Nmbr='4 ') to anything else you need.

Automatic counting and summing (COUNT, SUM)

In some applications, you won't need to see the actual records, but will want to know how many meet certain conditions, or what the total is for some specified condition (How many widgets do we have in stock? How many are on back order? What is the total of our accounts payable?)

For counting use:

^COUNT [scope] [FOR conditions] [TO memory variable]^

This command can be used with none, some or all of the modifiers.

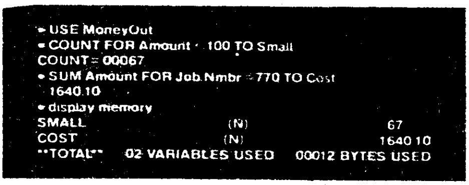
Unqualified, it counts all the records in the database. The "scope" can be limited to one or a specified number of records, and the "condition" can be any complex logical expression (see earlier section on expressions). The result of the count can be stored in a memory variable, which is created when the command is executed if it did not exist.

To get totals, use:

^SUM field(s)[scope][FOR condition][TO memory variable(s)]^

You can list up to 5 numeric fields to total in the database in USE. If more than one field is to be totaled, the field names are separated by commas. The records totaled can be limited by using the "scope" and/or conditional expressions after the FOR (Client <> 'SEM' .AND. Amount > 10...).

If memory variables are used (separated by commas), remember that totals are stored based on position. If you don't want to store the last fields in memory variables but do want to see what the amounts are, there's no problem: simply name the first few variables that you want. If there's a gap (you want to save the first, third and fourth field totals out of six), name memory variables for the first four fields then RELEASE the second one after the SUM is done.



```
• USE MoneyOut
• COUNT FOR Amount < 100 TO Small
COUNT = 00067
• SUM Amount FOR Job.Nmbr = 770 TO Cost
 1640.10
• display memory
SMALL                  (N)                    67
COST                   (N)               1640.10
**TOTAL**   02 VARIABLES USED    00012 BYTES USED
```

Summarizing data and eliminating details (TOTAL)

^TOTAL^ works similarly to the sub-total capability in the REPORT command except that the results are placed in a database rather than being printed out:

```
TOTAL ON <key> TO <database> [FIELDS list] [FOR conditions]
```

NOTE: The database that the information is coming from must be presorted or indexed on the key that is used in this command.

This command is particularly useful for eliminating detail and providing summaries. The screen shows what happens with our <MoneyOut> database:

```
^USE MoneyOut^
^INDEX ON Job:Nmbr TO Jobs^
^USE MoneyOut INDEX Jobs^
^TOTAL ON Job:Nmbr TO Temp FIELDS Amount FOR Job:Nmbr >699;
                                          .AND. Job:NMbr < 800^

^USE Temp^
^LIST^
```

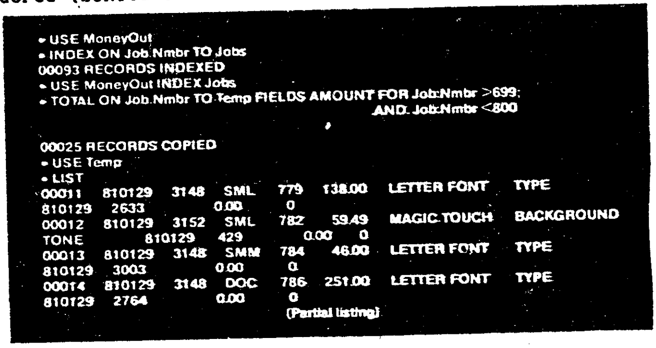
The new database has one entry for each job number, and a total for all the costs against that job number in our <MoneyOut> database. One problem with the new database, however, is that only two of the fields contain useful information.

This can be handled with one more command line. ^TOTAL^ transfers all the fields if the database named did not exist, but uses the structure of an existing database. In the commands above, we could have limited the fields in the new database by creating it first, before we used the ^TOTAL^ command:

```
^COPY TO Temp FIELDS Job:Nmbr, Amount^
```

Now when we ^TOTAL^ to <Temp>, the new database will contain only the job numbers and totals. Try it with your database.

This same technique can be used to summarize quantities of parts, accounts receivable or any other ordered (SORTed or INDEXed) information.



```
• USE MoneyOut
• INDEX ON Job.Nmbr TO Jobs
00093 RECORDS INDEXED
• USE MoneyOut INDEX Jobs
• TOTAL ON Job.Nmbr TO Temp FIELDS AMOUNT FOR Job:Nmbr >699;
                                        .AND. Job:Nmbr <800

00025 RECORDS COPIED
• USE Temp
• LIST
00011   810129   3148   SML    779   138.00   LETTER FONT    TYPE
810129   2633          0.00     0
00012   810129   3152   SML    782    59.49   MAGIC TOUCH    BACKGROUND
TONE         810129   429          0.00   0
00013   810129   3148   SMM    784    46.00   LETTER FONT    TYPE
810129   3003          0.00     0
00014   810129   3148   DOC    786   251.00   LETTER FONT    TYPE
810129   2764          0.00     0
                                (Partial listing)
```

Section II Summary

This section has broadened the scope of what you can now do with dBASE II.

We have shown you how different operators (arithmetic, relational and string) can be used to modify dBASE II commands to give you a greater degree of control over your data than is possible with other database management systems.

Since data structures are the basis of database systems, we have covered a number of different ways in which you can alter the these structures, with or without data in the database.

We have also shown you how to enter, alter and find the specific information you may be looking for. We have also introduced new global commands that make it possible for you to turn all that data into information with a single command (COUNT, SUM, REPORT, TOTAL).

In the next section, we will show you how to set up dBASE II command files (programs), so that you can automate your information processes.