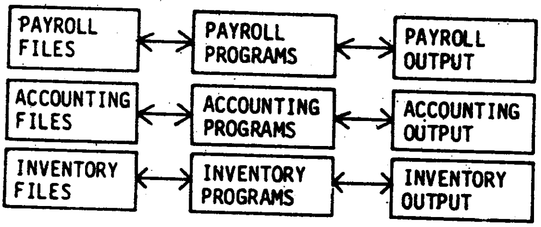
Section V:

Database Basics

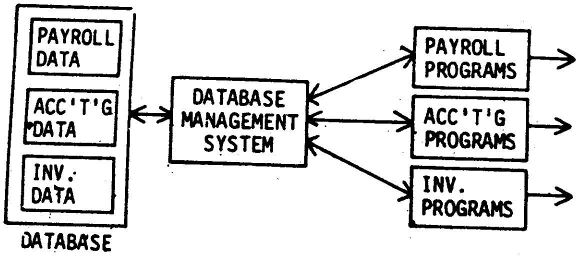
A database management system (DBMS) like dBASE II is considerably different from a file handling system.

A file handling system is usually configured like this:



The payroll programs process the payroll files. The accounting programs process the accounting files. And the inventory programs process the inventory files. To get reports that combine data from different files, a new program would have to be written and it wouldn't necessarily work: the data may be incompatible from file to file, or may be buried so deeply within the other programs that getting it out is more trouble than it's worth.

A database management system integrates the data and makes it much easier to get useful information from your records, rather than just reams of data. Conceptually, a DBMS looks something like this:



Data is monitored and manipulated by the DBMS, not the individual applications programs. All of the applications systems have access to all of the data. In a file handling system, this would require a great deal of duplicated data. Aside from the potential for entry errors, data integrity is extremely hard to maintain when the same data is supposed to be duplicated in different files: it never is.

To generate a new processing system in a file handling system, a new program and new files must be set up. Using a DBMS, a new access program is written, but the data does not have to be restructured: the DBMS takes care of it.

If a new kind of data is added to a record (salary history in a personnel file, for example), file handling programs have to be modified. With a DBMS, additions and changes have no effect on the programs that don't need to use the new information: they don't see it and don't know that it's there.

Database management systems come in two flavors: hierarchal and relational. These terms refer to how the DBMS keeps track of data.

A hierarchal system tends to get extremely complex and difficult to maintain because the relationships between the data elements are maintained with sets, linked lists, and pointers telling the system where to go next. Very quickly, you can end up with lists of lists of lists and pointers to pointers to pointers.

A relational database management system like dBASE II is a great deal simpler. Data is represented as it is, and the relation between data elements can be considered a two-dimensional table like this one:

Col.1	Col.2	Col.3	Col.4	Col.5
Invoice Number	Supplier	Description	Amount	Number
2386	Graphic Process	Prints	23.00	BBQ-747
78622	Brown Engraving	Litho plates	397.42	TFS-901
M1883	Air Feeight Inc.	Shipping	97.00	SPT-233

Each row going across the table is called a record. Each column is called a field of the record. Each entry in the table must be a single value (no arrays, no sets, etc.) All the entries in a column must be of the same type. Each record (row) is unique, and the order of records (rows) doesn't matter.

When we show you more realistic examples later, you'll see that records don't get any more complicated, just larger.

A brief introduction to database organization

Once you've got your database set up, you'll want to access your data in an orderly, ordered manner.

With some databases, the order in which you enter the data will be the order in which you want to get your information out. In most cases, however, you'll want it organized differently.

With dBASE II you can organize data using the **SORT** command or the **INDEX** command. (Both of these are described in more detail in Section II: Organizing your databases.)

The **SORT** command moves entire records around to set up your database in ascending or descending order on any field that you specify (name, ZIP code, etc.). This field is called the key.

One drawback of sorting is that you may want to access the database on one field for one application, on another field for a different application. Another drawback is that any new records added are not in order, and would require a sort every time you entered data if you wanted to maintain the order.

Finding data is also relatively slow, since the sorted database must be searched sequentially.

INDEXING is a way around these problems.

Indexing is a method of setting up a file using only the keys that you are interested in, rather than the entire databases. A key is a database field (or combination of fields) that make up the "subject" of the record. In an inventory system, the part number might be the subject, and the amount-on-hand, cost, location, etc. the descriptive fields. In a personnel database, names or employee numbers would probably make the best keys.

With an indexed database, the keys alone are organized with pointers to the record to which they belong. dBASE II uses a structure called B*-trees for indexes. This is similar to a binary tree, but uses storage much more efficiently and is a great deal faster. A FIND command (described in Section II) typically takes 2 seconds with a medium to large database.

If you need your data organized on several different fields for different applications, you can set up several index files (one for each of the fields) and use the appropriate index file whenever required. You could have index files ordered by supplier name, by customer number, by ZIP code or any other key, all for a single database.

New entries to a database are automatically added to the index file being used.

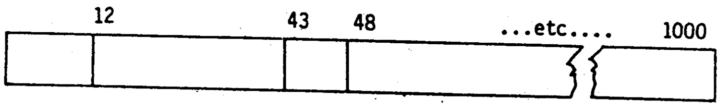
Another advantage of indexed databases is the rapid location of data that you are interested in.

dBASE II Records, Files and Data Types

dBASE II was designed to run on your micro so its scope stops short of infinity, but you'll find that you'll have to work at figuring out how to get to its maximums.

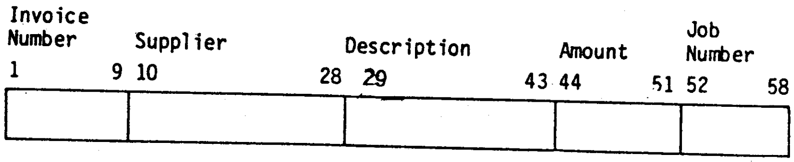
dBASE II limits you to 65,535 records per file, but with the memory and even "mass storage" limitations of a micro, this is really no limitation at all.

A dBASE II record can be as large as 32 fields and 1000 characters long (whichever comes first):



You might want to think of this as a 1000 character long strip that you can segment any way you want to up to the maximums, or shorten if you don't need to use it all. You can have four fields that use the full 1000 characters (254 characters per field maximum). Or a record one character (and field) long. Or anything in between.

In our previous example, each record had five fields and the total record length was 58 characters:



Data Types

As we said earlier, each field must contain a single type of data, and in dBASE II these are:

Character: all the printable ASCII characters, including the integers, symbols and spaces.

Numeric: positive and negative numbers as large as 1.8 x 10<63> down to numbers as small as 1.0 x 10<-63>. Accuracy is to ten digits, or down to the penny for dollar amounts as high as $99,999,999.99.

Logical: these are true/false (yes/no) values that occupy a field one character long. dBASE II recognizes T, t, Y and y as **TRUE**, while F, f, N and n are recognized as **FALSE**.

Field Names

Each field has a name so that dBASE II can recognize it when you want to find it. Field names can be up to 10 characters (no spaces) long, and must start with a letter, but can include digits and an embedded colon:

A	(valid)
A123456789	(valid)
Job:Number	(valid: upper and lowercase okay)
A123,B456	(illegal comma)
Reading:	(illegal: colon not embedded)

Tip: Use as many characters as it takes to make the name meaningful. 'Job:Nmbr' is a lot better than 'No.' and infinitely better than 'J'. Using a maximum of nine characters will make handling memory variables much easier (discussed later).

Another tip: Once you get into setting up Command files, you'll find it useful to use capital letters for words that dBASE II understands and upper and lowercase for fields, variables and other items that you control. You'll appreciate this the first time you go back into a command file to make changes.

dBASE II File Types

File names are limited to 8 characters and a 3 character extension after a period. You can use the colon in the file name, but then you'll only be able to manipulate the files through dBASE II: CP/M will store the files and get the names right, but won't recognize them if you ask it to perform a function like PIP. Ten character long filenames aren't a problem: CP/M simply chops them down to eight. If you use upper and lower case letters to name your files, CP/M will change them to capitals, but they'll still show up better in your command files.

A dBASE II file is simply a collection of information of a similar type under a single name, something like a giant file folder. dBASE II operates with the six different file types described below.

DBF Database files: This is where all your data is kept and the extension is automatically assigned by dBASE II when you **CREATE** a new file. Each .DBF file can store up to 65,535 records. Do not use a word processor on these files.

.FRM Report form files: These files are automatically created by dBASE II when you go through the **REPORT** dialog. They contain headings, totals, column contents, etc. They can be modified using a word processor or text editor, but we definitely recommend against this practice: make your changes using dBASE II.

.CMD Command files: These files contain a sequence of dBASE II statements to perform functions that you use frequently, and can be a complex as a complete payroll system. These are created using a text editor or word processor.

.NDX Index files: These are automatically created by the **INDEX** command. Indexing provides very rapid location of data in larger databases.

.MEM Memory files: These are automatically created when you **SAVE** the results of computations, constants or variables that you will want later. You can **SAVE** up to 64 items, each up to 254 characters long, then **RESTORE** them the next time you need them.

.TXT Text output files: This file is created when you use the **SET ALTERNATE** command to store everything that goes to the CRT on your disk, too. This feature can be used as a system logging function, and the information can later be edited, printed, and/or saved. They are also created when you **COPY...SDF**.

dBASE II OPERATIONS SUMMARY

Arithmetic operators (generate arithmetic results: p.37)

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

Relational Operators (generate logical results: p. 37)

<	:	less than
>	:	greater than
=	:	equal
<>	:	not equal
<+	:	less than or equal
>=	:	greater than or equal

Logical Operators (generate T/F logical results: p. 38)

( )	:	parentheses for grouping
.NOT.	:	Boolean not (unary operator)
.AND.	:	boolean and
.OR.	:	boolean or
$	:	substring logical operator (p. 40) (is string1 in string2?)

String operators (generate string results: p. 41)

+	:	string concatenation (joining)
-	:	string concatenation with blank squash

dBASE II FUNCTION SUMMARY

#	record number (p. 76)
*****	deleted record (p. 76)
EOF	end of file (p.76)
!(<variable/string>)	convert to uppercase (p. 80)
TYPE(<expression>)	data type (p. 80)
INT(<variable/expression>)	integer function (p. 80)
VAL(<variable/string/substring>)	string to integer (p.81)
STR(<expression/variable/number>, <length>, <decimals>)	integer to string (p.81)
LEN(<variable/string>)	string length (p. 81)
$(<expression/variable/string>, <start>, <length>)	substring select (p. 82)
@(<variable1/string1>, <variable2/string2>)	substring search (p. 82)
CHR(<number>)	number to ASCII (p. 82)
&	macro substitution (p. 83)
FILE(<"filename"/var/exp>)	file exists? (p. 83)
TRIM	trailing blanks (p. 83)

dBASE II COMMAND SUMMARY

The following abbreviations are used in this summary:

```
  <exp> = expression
  <var> = variable
  <str> = string
<coord> = coordinates
```

The symbols <..> bracket items that are to be specified by the user. Square brackets [..] enclose optional items. In some cases, options are nested (themselves have other options).

```
? <exp [,list]>
```
Display an expression (or list separated by commas) (p. 25)

```
@ <coord> [SAY <exp> [USING 'picture']] [GET <var> [PICTURE 'picture']]
```
Format console screen or printer output (p. 88)

```
ACCEPT ['prompt'] TO <var>
```
Input a character string from the console, no quotes (p. 68)

```
APPEND [BLANK]
APPEND FROM <filename> [SDF]     [FOR <exp>]
                       [DELIMITED [WITH delimiter]]
```
Add data to a database (pp. 26, 47, 70)

```
CANCEL
```
Abort a command file execution

```
CHANGE [scope] FIELD <list> [FOR <exp>]
```
Make multiple changes to a database (p. 51)

```
CLEAR
```
Reset dBASE file and memory variable environment (p. 76)

```
CONTINUE
```
Continue a LOCATE command (p. 55)

```
                             [SDF]
COPY [scope] TO <filename>  [STRUCTURE] [FIELD <list>] [FOR <exp>]
                             [DELIMITED [WITH delimiter]]
```
Copy data from a database to another file (pp. 43, 47, 49)

```
COPY TO <filename> STRUCTURE EXTENDED
```
Creates a new file whose records define the structure of the old file. (see also CREATE <newfile> FROM <oldfile>)

```
COUNT [scope] [FOR <exp>] [TO <var>]
```
Counts records that satisfy some condition (p. 58)

CREATE
Make a new database (p. 14)

CREATE <newfile> FROM <oldfile>
Creates <newfile> with structure determined by the data in the records of <oldfile>. (see also COPY STRUCTURE EXTENDED)

DELETE [scope] [FOR <exp>]
Mark specified records for deletion (p. 28)

DELETE FILE <filename>
Erase a file from the system (p. 28)

DISPLAY [scope] [FOR <exp>] [OFF]
Show data based upon request (pp. 20, 23)

DISPLAY [scope] [<field> [,list]]
Shows only the selected field(s)

DISPLAY STRUCTURE
Show structure of the database in USE (p. 23)

DISPLAY MEMORY
Show the contents of the memory variables (p. 35)

DISPLAY FILES [ON disk drive]
Show a disk directory (p. 23)

DO <filename>
Execute a command file (p. 63)

DO WHILE <exp>
Perform a group of commands repeatedly (p. 66)

EDIT
Alter the data in a database (p. 18)

EDIT [number]
Presents a specific record for editing (p. 18)

EJECT
Do a form feed on the printer

ELSE
Alternate execution path in an IF command (p. 64)

ENDDO
Terminator for DO WHILE command

ENDIF
Terminator for an IF command

ERASE
Clear console screen

FIND <key>
Locate a record in an indexed database based upon key value (no quotes needed for character keys) (p. 54)

GO or GOTO [RECORD], or [TOP], or [BOTTOM], n
Position to a given place in a database (p. 24)

IF <exp>
Conditional execution command (p. 64)

INDEX ON <key> TO <filename>
Create an index file for the database in USE (p. 52)

INPUT ['prompt'] TO <var>
Accept user inputs into memory variables. User prompt string is optional (p. 68)

INSERT [BEFORE]
[BLANK]
Add a new record to a database among other records (p. 26)

JOIN TO <filename> ON <exp> [FIELDS <list>]
Create a database composed of matching records from two other databases (p. 87)

LIST
Show data records (pp. 20, 21)

LOCATE [scope] [FOR <exp>]
Find the record that matches a condition (p. 54)

LOOP
Escape mechanism for DO WHILE groups (p. 66)

NOTE or *
A command file comment that is not displayed when the command file is run

MODIFY COMMAND <filename>
Permits modification of a file directly from dBASE II (p. 76)

MODIFY STRUCTURE
Alter the structure of a database. Destroys all data in the database (p. 42)

PACK
Eliminates records marked for deletion (p. 28)

QUIT [TO list of CP/M level commands or .COM files]
Terminate dBASE and execute a program chain. Each command must be in quote marks, and commands must be separated by commas (p. 76)

READ
Enter full screen editing of a formatted screen. Accepts data into GET commands (p. 70)

RECALL [scope] [FOR <exp>]
Unmark records that have been marked for deletion (p. 28)

RELEASE [<var> [,list]] or [ALL]
Eliminate unwanted memory variables (p. 36)

REMARK
A comment that is shown on the screen when the command file is run

RENAME <oldfile> TO <newfile>
Give a file a new name (p. 76)

REPLACE [scope] <field> WITH <exp> [,<field> WITH <exp>...]
Alter data in a database. Make sure that you have a backup, because dBASE II will do precisely what you ask it to do, even if it's not exactly what you had in mind (p. 50)

REPORT [scope] [FORM <filename>] [TO PRINT] [FOR <exp>]
Generate a report (p. 56)

RESET
Tell CP/M that a diskette swap may have occurred

RESTORE FROM <filename>
Remember SAVEd memory variables. Destroys all existing memory variables

RETURN
Terminate a command file and return to calling file

SAVE TO <filename>
Write memory variables to a file for future use

SELECT [PRIMARY] or [SECONDARY]
Switch working areas (p. 75)

SET parameter [ON], or [OFF]
Dynamically reconfigure dBASE operation (p. 84)

SKIP ±<exp/number>
Move forward or backwards in the database (p. 24)

SORT ON <key> TO <filename> [ASCENDING]
[DESCENDING]
    Generate a database that is sorted on a field (p. 52)

STORE <exp> TO <var>
    Place a value into a memory variable (p.33)

SUM [scope] <field [,list]> [TO <var [,list]> [FOR <exp>]
    Total fields in a database (p. 58)

TOTAL TO <filename> ON <key> [FIELDS <field [,list]>
    Generate a database with sub-totals for records (p. 59)

UPDATE FROM <filename> ON <key> [ADD <field [,list]>]
[REPLACE <field [,list]>]
    Modify a database with data from another database (p. 86)

USE <filename> [INDEX <filename>]
    Open a database file for future operations (p. 20)

USE
    Close a previously opened database file

WAIT [TO <var>]
    Pause in program operation [for input] (p. 68)

dBASE II commands grouped functionally

FILE STRUCTURE:

CREATE   defines an entirely new file structure

CREATE <newfile> FROM <oldfile> creates a new file whose structure is described in the records of the old file.

```
USE <oldfile>
COPY TO <newfile> STRUCTURE
     These two commands combined create a new file with
     the same structure as an old file
```

```
USE <oldfile>
COPY TO <newfile> STRUCTURE EXTENDED
     Create a new file that contains the structure of
     the old file as data
```

```
CREATE <newfile> FROM <oldfile>
     Creates a new file whose structure is defined by the
     records in the old file.
```

```
DISPLAY STRUCTURE
LIST STRUCTURE
     Both show the structure of the file in USE
```

```
MODIFY STRUCTURE changes file names, sizes, and overall
     structure, but destroys data in the database
```

To change structure with data in the database:

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

To rename fields with data in the database:

```
USE <oldfile>
COPY TO <newfile> SDF
MODIFY STRUCTURE
APPEND FROM <newfile>.TXT SDF
DELETE FILE <newfile>
```

FILE OPERATIONS:

USE <filename> opens a file

USE <newfile> closes the old file

USE closes all files

RENAME <oldname> TO <newname>
Must NOT rename an open file

COPY TO <filename> creates a backup copy

CLEAR closes all files and erase all memory variables

SELECT [PRIMARY][SECONDARY]
allows two files to be independently open at the same time. Data can be transferred with P. and S. prefixes

DISPLAY FILES [ON <d>] lists databases on logged-in drive (or drive specified), can use LIST instead

DISPLAY FILES LIKE <wildcard> [ON <d>] shows other types of files on drives

QUIT closes both active areas, all files, terminates dBASE II operation

ORGANIZING DATABASES:

SORT ON <key> TO <newfile>
INDEX ON <key> TO <newfile>
Can use multiple keys for both commands

COMBINING DATABASES

COPY TO <newfile> creates a duplicate of the file in USE

APPEND FROM <otherfile> adds records to the file in USE

UPDATE FROM <otherfile> ON <key> adds to totals or replaces data in the file in USE. Both files must be sorted on the <key>.

JOIN creates a third file from two other files

EDITING, UPDATING, CHANGING DATA:

DISPLAY, LIST, BROWSE let you examine the records

DELETE marks record so it is not used

RECALL unmarks record

PACK erases deleted records

EDIT lets you make changes to specific records

REPLACE <field WITH data> global replacement of data
in fields, can be conditional as with most dBASE II
commands

CHANGE..FIELD edit based on field, rather than record

@ <coord> GET <var>
READ displays the variable, lets you change it

INSERT [BEFORE][BLANK] inserts a record in a database

UPDATE FROM <otherfile> ON <key> adds to totals or
replaces data in file in USE from another file

MODIFY COMMAND <filename> allows changes to your command
files without having to go through your text editor

USING VARIABLES:

(Allowed up to 64 memory variables plus any number of
field names.)

LIST MEMORY, DISPLAY MEMORY both show the variables,
their
data types and their contents

& returns the contents of a character memory variable
(i. e., provides a literal character string)

STORE <value> TO <var> sets up or changes variables

RELEASE <var> cancels the named variable

SAVE MEMORY TO **<filename>** stores memory variables to
the named file (with .MEM extension)

RESTORE **FROM** <filename> reads memory variables back into
memory (destroys any other existing memory variables)

INTERACTIVE INPUT:

WAIT stops screen scrolling, continues with any key

WAIT TO **<var>** accepts character to memory variable

INPUT ['prompt'] TO <var> accepts any data type to a memory variable (creates it if it did not exist), character input must be in quotes

ACCEPT ['prompt'] TO <var> same as INPUT, but no quotes around character input

@ <coord> SAY ['prompt'] GET <var> [PICTURE]
READ
displays memory variable, replaces it with new input

SEARCHING:

SKIP [±<exp>] moves forward or backward a specific number of records

GO[TO] <number>, GO TOP, GO BOTTOM
move you to a specific record, the first record, or the last record in the database

FIND <str> works with indexed file in USE, very fast

LOCATE FOR <exp>
CONTINUE
Searches entire database

OUTPUT:

?, DISPLAY, LIST show expressions, records, variables, structures

REPORT [FORM <formname>] creates a custom format for for output, then presents data in that form when called

@ <coord> SAY <var/exp/str> formats output to screen or to printer ([USING <format>] can be added to provide PICTURE format for the printer)

PROGRAMMING:

(Programs stored in COMMAND FILES with .CMD extension.)

DO <filename> starts the program

```
IF <conditions>                      Makes choices, single or
   perform commands                  multiple (when nested)
ELSE
  perform other commands
ENDIF


DO WHILE <conditions>                <Conditions> must be
   perform commands                  changed by something
ENDDO                                in the loop eventually
```

(This page intentionally left blank.)