Some notes on dBASE

John R Hudson

6th June 2023

Introduction

dBASE was invented in the 1970s by Wayne Ratliff who worked at NASA's Jet Propulsion Lab. He sold it through small ads in the computer press as Vulcan until Ashton-Tate began to market the program as dBASE II.

It took the market by storm but its successors, dBASE III (written in C rather than assembler), dBASE III Plus and dBASE IV, never achieved their promise and competitors like FoxBASE and Clipper began to eat into the market. Wayne Ratliff had left by now and, unlike Wordstar which brought back the inventors when the company ran into trouble, Ashton-Tate struggled on alone.

They lost lawsuits for copyright infringement because they had failed to disclose that it was based on JPLDIS and were taken over by Borland. Subsequently Microsoft and Computer Associates swallowed up its two main rivals, Fox and Nantucket, producers of FoxBASE and Clipper respectively.

dBASE has always attracted ambivalent feelings — Jerry Pournelle, one of its earliest reviewers, described it in *BYTE* as 'infuriatingly excellent.'

Overview

dBASEII was one of the first, if not the first, program to offer an integrated development environment (IDE). When you started dBASE, you could manage a database, creating tables, adding or importing data, searching and manipulating the data and exporting it in a variety of formats. But you could also write programs to automate operations or to provide a user-friendly interface for users with no understanding of databases; you could edit text files, do many file management tasks and make calls to the operating system, all without exiting dBASE.

For example, when CP/M Plus was brought out, the place where the date was held in memory was changed. But using the dBASE PEEK function, you could read the date in CP/M Plus and use a dBASE program to convert it into dBASE format and use the date in dBASE.

Elements

dBASE originally had seven elements, six types of file:

- data tables, which dBASE calls databases,
- index files,
- memory files,
- report forms,
- screen format files, and
- program files

and a command line interface for which the prompt is the famous `.` dot prompt.

The screen format files were removed once it acquired a GUI in the 1990s as layout was now handled by the GUI and not through the CLI.

Data tables

dBASE data tables, always ending in `.dbf`, are relational, consisting of columns, or fields, and rows, or records.

The first row of each column contains the structure of the fields in CSV format: FIELDNAME,TYPE,WIDTH,DECIMAL PLACES, where, in dBASEII, TYPE may be:

C character

L logical (i.e. Boolean)

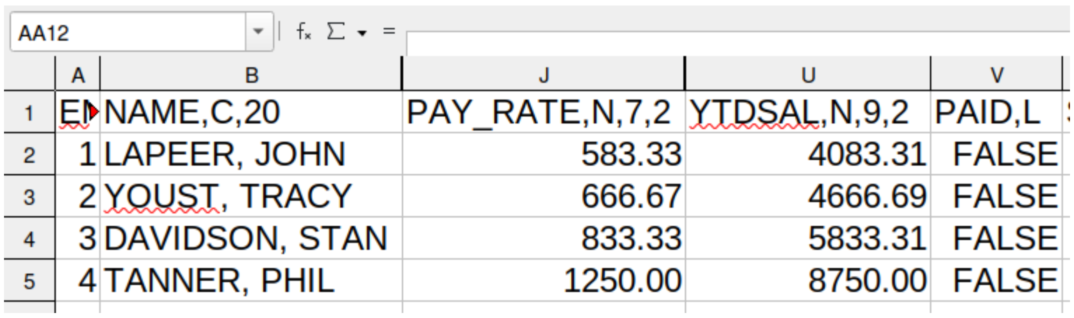
N numerical

Various other TYPEs, including **D** for date in dBASEIII, were added in later versions.

WIDTH may be up to 254 characters in dBASEII and the sum total of all WIDTHs in dBASEII must not exceed 1000, raised to 4000 in dBASEIV

DECIMAL PLACES only appears in numerical fields

You could have 65535 rows and 32 fields in a table and as many data tables as you want but, in dBASEII, only two tables open at a time; so theoretically you could have access to 128MB of data except that the maximum filesize under CP/M was 32MB!



	A	B	J	U	V
1	EI	NAME,C,20	PAY_RATE,N,7,2	YTDSAL,N,9,2	PAID,L
2	1	LAPEER, JOHN	583.33	4083.31	FALSE
3	2	YOUST, TRACY	666.67	4666.69	FALSE
4	3	DAVIDSON, STAN	833.33	5833.31	FALSE
5	4	TANNER, PHIL	1250.00	8750.00	FALSE

The first row, showing the CSV format table definition, shows up when a data table is imported into a spreadsheet but is not visible in dBASE.

You can extract the CSV information to create a separate data table containing the structure of the data table and then use that to create a another data table.

By combining the data table structure tables you can create a data dictionary of all the fieldnames and their types and widths which you use; you can then create new data tables by extracting the relevant data rather than laboriously going through the process of defining the structure of each new table by itself. This ensures that all the fields you use in all the tables you have are consistent and well-defined.

Index files

You can both SORT and INDEX the records in a data table. When you sort the records, dBASE creates a new file with all the records in a new order. If you index a file, dBASE creates a separate file with a name ending in `.ndx` which contains a list of the order in which you would like the records to appear. The actual order of the records in the database file does not change, only the order in which they will be processed or appear on your screen or printer. Unlike some other database programs, the index file is not linked to the data table unless you make the link. So you can create multiple indexes for the same data table and choose which one to use knowing that none of the others will interfere with the order set by the one you have chosen.

You can index on parts of a field or parts of multiple fields, such as LASTNAME and FIRSTNAME. While an index can have up to 99 characters, indexing on parts of fields, such as the first six characters of LASTNAME and the first four characters of FIRSTNAME, reduces the size of the index files while indexing on the last, middle and first parts of a character based DD/MM/YYYY date in dBASEII allows you to put the dates in their chronological sequence.

Memory files

Like most programming languages, dBASE allows temporary variables to be created to handle both calculations and the manipulation of text. You can have up to 64 in dBASEII and you can `SAVE` all or some of these temporary variables to disk in a file with a name ending in `.mem`. For example, you can save the next cheque number to be used and `RESTORE` it from the `.mem` file when you pay the next cheque. If you save them all, when you start again, you can resume where you left off.

You could also store print formatting commands in a temporary variable and send the contents to a printer, for example, to print in landscape at 17 cpi — useful when there were no printer configuration files.

Numeric variables can contain 16 significant figures but, while dBASEII would always use all 16, only the first 10 were guaranteed to be accurate after they had been used in a calculation.

Temporary variables are not typed; so you can, for example, extract numerical data from a character field and store it as numerical data or *vice-versa*.

Report forms

You can LIST or DISPLAY the information in a data table in a variety of ways but creating a report form ending in `.frm` allows you to specify how you want information from the data table to appear on the screen or on a printer. For example, you may want

- only certain fields

- the fields to appear in a different order from the one in the data table
- to restrict the width of each field, for example, if the total of the field widths exceeds the size of your screen or your printer
- to create a report in landscape orientation for your printer.

Report forms can also be used to sub-divide a report and to provide sub-totals and totals based on the subdivision, for example, monthly sub-totals and a yearly total. However, in dBASEII report forms are limited to 24 fields.

Screen format files

Screen format files, ending in `.fmt`, contain commands for displaying input and output on a text screen. In dBASEII the screen is assumed to be 24 rows by 80 columns (which was embedded in the code; so you needed to edit the program if your screen size was different).

```
@ 5,10 SAY 'This is a custom entry screen for the NAMES file.'
@ 7,5 SAY 'The first name is ' GET First_Name
@ 7,45 SAY 'The last name is ' GET Last_Name
@ 9,8 SAY 'The address is ' GET Address
@ 11,5 SAY 'City ' GET City
@ 11,45 SAY 'State ' GET State
@ 11,60 SAY 'ZIP code ' GET ZIP
```

`@ n, n` specifies the row and column where dBASE should display the text following SAY on the screen. GET creates a text entry in which the user can enter the data and the variable after GET specifies the name of the memory variable which will hold that data, i.e. `First_Name`, `Last_Name`, etc.

If `SAY <variable name>` is followed by `USING` or `GET <variable name>` is followed by `PICTURE`, the `USING or PICTURE` restricts the possible entry, for example:

```
GET date PICTURE "99/99/9999"
```

restricts the user to numeric entries and automatically puts the `/`s in the right place.

```
SAY amount USING "$$$$$.99"
```

displays the amount to the right with two decimal places.

```
GET posttown PICTURE "!!!!!!!!!!!!!!"
```

changes any lowercase characters to uppercase.

While modern HTML allows you to restrict the format of certain form entries, some modern database software cannot offer this functionality without special programming.

No actual programming takes place on any of the above entries; this must be done via the command line interface or a program which `READ`s the memory variables and then does what is needed.

Program files

Program files, sometimes called command files because they ended in `.cmd` on CP/M but `.prg` in MS-DOS as `cmd` was already in use in MS-DOS/PC-DOS, contain commands, some of which

```
DO WHILE .T.
CLEAR
@ 2, 0 SAY '* * * * * * B I L L S & T I M E S H E E T S * * * * * *'
@ 4,15 SAY ' 1> ENTER EMPLOYEE TIME SHEETS'
@ 6,15 SAY ' 2> ENTER SUPPLIER BILLS'
@ 8,15 SAY ' 3> UPDATE the Costbase'
@ 10,15 SAY ' 4> EDIT the Postfile'
@ 12, 0 SAY '* * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * *'
@ 14,15 SAY 'Choose a number or press <ENTER> to exit.'
@ 15, 0
WAIT ' ' TO Costing
DO CASE
CASE Costing = '1'
@ ROW(),0 SAY 'Preparing Employee Time Sheets'
DO Costtime
CASE Costing = '2'
@ ROW(),0 SAY 'Preparing Supplier Bills'
SELECT 8
USE Invoices
DO Costbill
CASE Costing = '3'
@ ROW(),0 SAY 'Preparing to UPDATE Costbase'
SELECT 3
USE Costbase INDEX Costname,Costjobs
DO Costupda
CASE Costing = '4'
** multiple lines omitted **
ENDCASE Costing
ENDDO
```

Figure 1: Extracts from a program file

can be used on the command line, some of which you will find in screen format files and others of which can only be used within a program file.

Program files are best used for routine and repeated operations or to create a menu based interface for users without programming skills. Figures 1 and 2 contain extracts from a much longer program file to illustrate some of the commands available in dBASE.

Many programs will start with

```
DO WHILE .T.
CLEAR
```

The first command prevents the program from exiting after the last command in it; the second (`ERASE` in dBASEII) clears the screen.

The next few lines in figure 1 set up the options on the screen.

```
WAIT ' ' TO Costing
```

```
PROCEDURE Indexing
USE Costbase
GO BOTTOM
Temp = STR(RECNO(),5)
SET TALK ON
CLEAR
@ 5,0 SAY 'Indexing ' + Temp + ' Costbase records'
?  'by JOB NUMBER to Costjobs.ndx:'
ERASE Costjobs.ndx
INDEX ON Job_Nmbr TO Costjobs
?
?
?  'Indexing ' + Temp + ' Costbase records'
?  'by SUPPLIER NAME to Costname.ndx:'
ERASE Costname.ndx
INDEX ON Name to Costname
** multiple lines omitted **
USE
SET TALK OFF
RETURN
```

Figure 2: Extracts from a program file

waits for the user to press a key and then places the value of the key in the memory variable `Costing`.

Much later in this dBASEIII program there is an exit option if the user does not press a valid key. Unfortunately, one of the features of dBASEII which went missing with the move to dBASEIII was an SQL style LIKE operator allowing you to check if a value was contained anywhere in another, longer, value.

In dBASEII, rather than using WAIT, you would have a test as to whether the key pressed was like `'1234'` and only move on if it was one of those. In practice, I would often add `Qq` options to allow people to quit if they did not want any of the options offered.

Once the key press has been stored, there is a `DO CASE` which, in the first three cases, passes control to another program with the command `DO`.

In CASE 2, there is also a change in the data table being used to `Invoices`. Because dBASE considers each data table as a database, `USE` has the same function as in SQL except that it applies to a table (in SQL terms) rather than to a database (in SQL terms).

IN CASE 3, there is a change in both the data table being used and the indexes being used to control the order in which the program will act on each record.

The program ends (after a lot of lines which I have omitted) with

```
ENDCASE Costing
ENDDO
```

Figure 2 illustrates re-indexing data tables which have changed. Indexes will be updated if they are active while the data table is being amended but, particularly in CP/M, this can slow things down. So it can be more efficient to postpone re-indexing until the data tables are not being used.

In this example, we first re-index `Costjobs.ndx` against the `Costbase` data table.

```
GO BOTTOM
```

moves the pointer to the last record (guess what `GO TOP` does!).

```
Temp = STR(RECNO(),5)
```

creates the memory variable `Temp` which holds the record number of the last record as a character string 5 characters long, illustrating how a numeric memory variable can be converted into a character string. As the last record is 66535, a five character string can hold any record number.

```
SET TALK ON
```

tells dBASE to say what it is doing, in this case, list each record that it has indexed in turn. On CP/M one would try to avoid this as filling the screen with messages slows down the indexing!

```
@ 5,0 SAY 'Indexing ' + Temp + ' Costbase records'
```

allows you to insert the character string `Temp` into an expression. Using `-` rather than `+` removes any spaces round the variable in the expression

```
ERASE Costjobs.ndx
```

erases the existing index file (dBASEII would have said `DELETE FILE Costjobs.ndx`)

```
INDEX ON Job_Nmbr TO Costjobs
```

creates a new index `Costjobs` using the `Job_Nmbr` field while

```
INDEX ON Name to Costname
```

creates a new index `Costname` using the `Name` field.

Several more indexes (omitted) are re-indexed until

```
USE
SET TALK OFF
RETURN
```

closes any open data tables, stops the list of messages and instructs the program to return to the program from which it was called.

Because dBASE data tables on disk are accessed directly, `USE <data table>` creates access to the file on disk which can then be changed very easily. So it is important to use `USE` at the end of the program to prevent any accidental access to the file once the program has stopped running.

Reading and writing data

Once you have opened a data table with `USE <data table>`, you simply use the fieldnames to access the contents of the fields as in any relational database. By convention, if you wish to manipulate the contents of a field, you place it in a memory variable with `m` at the start of the fieldname, for example, `mcity` to hold the contents of `CITY`.[1]

[1]Though the example screen format file on page 4 came from a dBASEIII disk of utilities, it did not follow this convention!

When you have manipulated the data in the memory variables or obtained more data using `GET <memory variable>`, you overwrite the contents of the field in the data table with a series of `REPLACE` commands for each field, such as

```
REPLACE CITY WITH mcity
```

Other features of dBASE

Many dBASE commands do exactly what you would expect them to: `COPY COUNT CREATE EDIT FIND HELP LOCATE RETURN SKIP`

Preferences are set by using the `SET` command whether in the command line or in the `.prg/.cmd` file.

`DISPLAY` and `LIST` display the status of lots of different things; in the context of a data table `DISPLAY` displays the current record; `LIST` lists all the records.

`SELECT` allows you to choose which of the two data tables (in dBASEII — more in dBASEIII) you want to use.

`BROWSE` allows you to view the data table on disk; it should generally be avoided because of the risk of an accidental insertion/deletion.

`APPEND` allows you to add a fresh record to a data table or to import them from another data table or from a CSV file; `INSERT` allows you to insert a record in the middle of a data table. If you `SET CARRY ON`, it will also import the entries from the previous record.

The `DELETE/ERASE` command simply marks a record for deletion. It is only deleted following a `PACK` command.

`COUNT` counts the number of records which satisfy a criterion while `SUM` and `TOTAL` allow you to do mathematical calculations on the contents of fields.

`JOIN` allows you to use two data tables as if they were one while `UPDATE` allows you to update data in one data table by reference to another.

`QUIT TO` allows you to specify another program to load. This is particularly useful if you have set up a user friendly menu driven program as it prevents the user from exiting to the dot prompt.

Postscript

After Wayne Ratliff left Ashton-Tate in 1988, he wrote Emerald Bay of which he was later to say in an interview with Visual Pro Magazine in 2007:

> I have to tell you the guys that designed Access and Emerald Bay were either reading the same book or one followed from the other. There are an awful lot of things in Access that, while they weren't big inventions, I thought I was the first one to espouse the desirability for them. I don't want to accuse anybody of anything, but it was striking to me.