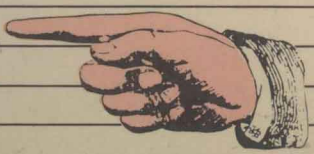
A BYTE BOOK

INTRODUCING dBase II™

Lan Barnes
Microtrend, Inc.



Introducing dBase II™

by

Lan Barnes
Microtrend Inc.

McGraw-Hill Book Company
New York St. Louis San Francisco Auckland Bogotá
Hamburg Johannesburg London Madrid
Mexico Montreal New Delhi Panama
Paris São Paulo Singapore
Sydney Tokyo Toronto

Library of Congress Cataloging in Publication Data

Barnes, Lan.
Introducing dBase II.

Includes index.
1. dBASE II (Computer program) 2. Business – Data processing. I. Title.
HF5548.4.D22B37 1985 650'.028'5425 84-26170
ISBN 0-07-041807-1

234567890 DOCDOC 898765

ISBN 0-07-041807-1

Printed and bound by R.R. Donnelley and Sons

Contents

Introduction

People have never lived without data and data bases. Archeologists tell us that cuneiform on clay tablets, hieroglyphics on papyrus, and runes on runestones were all used to record how much inventory was in stock, how much wine was in the wine cellar, and who owed whom what. It's comforting, although a little disappointing, to see that people 3,000 years ago led lives as conventional as ours.

Record keeping pervades civilized life. Every one of us leaves a trail of official records behind over the years—birth certificates, report cards and transcripts, military records, job histories, bank accounts and credit references, social security and FICA accounts; the list is endless.

Until very recently, all of these records were either managed by secretaries, clerks, and bureaucrats, or were managed by large computers too expensive to be used by anyone except the government or the retail giants of the business world.

Today, a true technological revolution is taking place. The advent of inexpensive microcomputers, or "personal" computers, has made it possible for almost anybody to have access to powerful electronic data management tools—housewives, executives, students, hobbyists, and the smallest of businesses. Indeed, computer electronics have advanced so quickly that personal computer owners today have more computing power and speed available in their own homes than most governments had 20 years ago.

However, the microcomputer revolution has probably done as much to confuse people about data and what data is as it has to help them. Nowhere is this more evident than in the area of data base management. Most people in business and management have a good intuitive sense of what data is used in their office, how it's arranged, and what decisions are made from it.

However, our comfort with data goes out the window when we start dealing with computer-based data base systems.

There are many reasons for this, but two are particularly striking. First, data base discussions in computer books are usually in abstract terms. And second, computerized data bases don't use paper records in data processing.

Where before an accounting clerk dealt with customer accounts or debits and credits in a ledger, now he or she "manages a data pool." Even worse, the file cabinets are gone. To get the answers to the daily questions that pop up in every office, the clerk has to use screen and keyboard to weave through an organization of data that can't be thumbed through with eye and finger. It's no wonder that everyone blames computers for foul-ups nowadays—"Sorry, but our computer's down," or "Your payment must have been lost in the computer."

The advent of dBase II and other microcomputer data base management systems will probably change that situation for the better. As more and more average people design and create their own data base systems, our societal consciousness of what data is, and what its value and dangers are, will grow. And as we all get more sophisticated, we will probably also become far less accepting of excuses that blame inefficiencies on computers.

dBASE II AS A PROGRAMMING LANGUAGE

dBase II is advertised by Ashton-Tate as being a relational data base language, and the statement is true as far as it goes. However, calling dBase a programming language and nothing else is a restricted view that is unfair both to dBase and to the new dBase user.

dBase II can be used on several different levels to do qualitatively different things. For example, when a user has a data problem or a body of data to manage, dBase can be used as a "data processing" program to sort and manipulate the data, and make specific or ad hoc reports from it.

When used this way, dBase is an application program. It has the same relationship to the user's data as a good word processing program has to the user's text—the user supplies the brains, going "one-on-one" with his data, and dBase is just a tool to do the work. dBase users distinguish this from programming by calling it *command level dBase*.

WHAT THIS BOOK IS DESIGNED TO DO

This book is designed to teach new users how to use dBase II both as a programming language and as a command-level application program. The subject matter has been organized so that the reader at any level of sophistication and dBase experience can enter the book where appropriate.

Chapters 1 and 2 are a quick crash course in data base terminology and

organization. These chapters are short, and in no way constitute a substitute for more comprehensive books on data base theory and design, but they will help newcomers to computer data base applications to get started, and they are sufficient introduction to the subject to make the rest of the book comprehensible.

Chapters 3, 4, and 5 are designed as a dBase workbook and would best be read with the book in your lap and dBase running on a computer in front of you. In these chapters, the 40 or so commands most frequently used in dBase programs are described and demonstrated. These middle chapters are divided into "runs," exercises that should take no more than an hour or two.

Chapters 6, 7, and 8 each concern a specific topic central to dBase programming—the dBase control structures, program I/O (input and output), and the process of relating multiple data files into one data base. The experienced dBase programmer may wish to jump straight to these chapters, and a programmer at any level may return to them from time to time.

Finally, the late chapters, 9, 10, and 11, concern such advanced topics as overall program design, debugging, and advanced techniques.

CHAPTER 1

A Data Base Crash Course

This chapter is designed to be a quick crash course in what a data base is, and how dBase II affects a data base on a microcomputer. You should be warned that the concepts and definitions presented in this chapter have been streamlined and simplified. The goal is to present concepts that make dBase II data bases easier to understand, not to teach a rigorously correct course in data base design or theory.

WHAT IS A DATA BASE?

For our purposes, *data* can be defined as two or more items of factual information that have a definable relationship to each other.

Let's consider an example. Imagine a computer screen (or piece of paper) with the unlabeled number "32767." This excellent number may or may not have useful meaning to us, but in an unlabeled, solitary context, it's impossible to tell. Is it a street address? A debit? A Mexican telephone number? If it is any of those, whom or what does it belong to? By itself, 32767 is not a practical data item.

Now consider this two-item array:

ZIPCODE 32767

The number now has a label, and suddenly, we're in business. The two related items of fact, a number and its label, have some meaning that is implicit in their being related. The postal zip code 32767 means in the real world

Lake Paisley, Florida. In this trivial example, the array of number and label constitutes a minimal data base containing one meaningful fact.

RETRIEVAL SPEED

Usually data that is relatively constant, like zip codes and telephone numbers, is compiled into listings, dictionaries, or other reference books, and we don't normally think of such books as data bases.

Reference books are an excellent way to store this kind of data. Indeed, putting data like this onto a computer is a poor choice over paper in most cases. For most of us, it's impractical to record all the zip codes on a computer for faster access, since we might look up fewer than ten new zip codes a year, and each time we did it, we would have to boot the computer, put the zip-code program in, enter the zip code for search, and so on.

For postal workers, however, the ability to look up a zip code instantly is highly useful, and they have the computers and programs to call up zip codes and their cities right now. This is an example of a case where speed of retrieval justifies using a computerized data base rather than a reference book.

AUTOMATIC DATA OPERATIONS

A data base management system does more than store and recover data quickly, however. A properly designed data base program stores data in a way that is convenient to the user, then operates on it in a predetermined manner, and reports it in a more meaningful form.

This can be illustrated by considering the flow of data in a single-owner small retail business—a sporting goods shop, for example. The owner would spend her day waiting on customers, collecting receipts—both cash and charge—and checking out items as they leave inventory.

All of the receipts and inventory changes would have to be posted at the end of the day in a batch, with reorder levels of inventory being noted as well as the cash-on-hand situation. When she paid her clerks, our business owner would have to calculate their gross sales for commission and withhold the proper federal and state taxes for their deductions.

If she has customers who charge on account, our businessperson will need to post their accounts against her receivables, generate billings, and do at least cursory cash flow projections. In all these operations, our store owner would be as mechanical and methodical as possible, to minimize errors and make things easier.

If the store owner were to put her records into a data base, it would change the nature of her bookkeeping. She would still be responsible for the data entry and the report generation, but the bookwork operations—adding

and subtracting dollar amounts, looking up the proper customer's page in the ledger, and decrementing inventory—could all be done automatically.

DATA POOLING AND DIFFERING DATA VIEWS

Data pooling means storing large bodies of related data in one or more files for later recovery. Two principles of data pooling are (1) data should not be duplicated within a data pool, and (2) data that are related to each other belong in the same pool.

To illustrate the first principle, consider a distribution office that services several hundred clients. When a client load gets that heavy, most businesses assign client numbers and centralize the function of keeping client names and addresses in one master file. The inconvenience of having to look up the name of the client that a record applies to is offset by the decrease in misspelled names and missent letters.

Likewise, a pooled data base will have customer names and addresses entered only once. In a well-designed data base, no data item would be repeated, since that would introduce the possibility of an embarrassing error, such as a personalized letter sent to "Ellen Campbell" on the address label that started out "Dear Allen" inside.

It's harder to justify the principle that all related data belongs in one data base, and this point is often argued at length when a data base is designed. The manufacturing inventory program used as a teaching program later in this book is a case in point. When it was designed, some of the assembly engineers objected to having costing data kept with the inventory record of each part. They reasoned that a part doesn't change in its function, no matter what its price, and the extra data was an annoyance for their people to track.

The inventory managers, on the other hand, objected that the costing data was too simplistic, since it ignored tooling costs on some custom parts and value added (primarily labor) on parts assembled in-house. In any event, the year-to-date unit costs that were actually put into the system were designed for the planners, and retained a rough validity for spotting the causes of cost overruns and the effects of hasty ordering.

On a fundamental level, however, the point is this: Unless data items for costing data had been designed into the system from the outset, no costing questions of any kind could ever have been answered by the data base.

To put it another way, you can't ever get data out of a data base unless someone puts it in first. So if a piece of information relates to the data base and you think you might ask questions about that information some day, it should probably have a place designed for it in the data base.

User Views of a Data Pool

Because related data can be organized in a variety of different ways, a single data base can appear to be many different things, depending on the selection of facts drawn from the pooled data. The set of facts that a given user sees in a data base, and the way that the facts are related, constitute his or her "user view."

We can illustrate the concept of user views by returning to the example of the retail sporting goods store, assuming that the owner has put her inventory, accounts, and transactions on a relational data base. A retail operation's data base program will appear to be several different programs, depending on who is looking at it. Typically, the fundamental record will be a point-of-sale entry made by a clerk at the time of a purchase.

To the clerk, the data base system will appear to be a cash register, prompting him for the customer's name, the purchase by stock number, the date, and method of payment. In return, the computer will tell the clerk the price, calculate the tax, post any discounts, and possibly look up the customer in a credit file to see if there is a history of bad checks.

The point-of-sale data would take on different appearances when abstracted for different purposes. To the accountant, a compendium of the month's sales, taxes collected, clerks' commissions and the like would appear to have come from an accounting program. Likewise, the stock clerk would see the data base as an inventory program, since the purchases would be charged to the inventory automatically.

In fact, one of the strengths of data base software is its ability to replace more specialized computer programs with one generalized system that can be adapted to the changing needs of the user. Methods for writing these kinds of programs using dBase II will be discussed in Chapters 8, 9, and 10. The point is that once a body of data has been gathered in a retrievable form, its usefulness is limited only by the nature of the data and the imagination of the person asking questions of the data base.

THE VALUE OF DATA

The value of data increases as it is processed. In the example of the sporting goods store, the week's raw receipts contained all the information necessary to post the store's financial and inventory records, but the posted records were obviously more valuable, if only by the value of the labor added.

The value of data is often highly subjective, and one person's junk data is another's gold mine. The sporting goods store would be wasting time collecting the astrological sign of each customer, whereas the customer's favorite sport would be something useful to have on file. For an occult book store, the value of the data would be reversed.

In this context, most books on computerized data processing urge their readers to back up their data storage frequently. So does this book. A 5¼-inch floppy disk costs between $2 and $4—less if you buy in bulk. Measure that cost against the projected cost of recreating your records after accidental loss or destruction of a primary copy of your data files. Then consider the hassle and uncertainty involved, and decide for yourself whether or not backing up data makes sense.

However, frequent backup is only one aspect of protecting the value of data. Other prudent practices should be made a fixed part of computer-room routine, and data base programs should have safety features built into them.

These practices and features include the following:

1. Retain all source documents in an organized way for a reasonable period after they've been entered into the computer.
2. Mark source documents when they are entered to prevent duplicate entry.
3. Design programs that automatically generate periodic hard-copy audit trail reports of all changes to the data base.
4. Invest in proper equipment to back up hard disk systems.
5. Institute disciplined procedures for labeling floppy disks with fixed labels that reflect their actual contents.

SORTS AND INDEXES

Random data becomes more valuable when it is ordered. Consider the case of a mail-order company that has a list of names and addresses of potential customers. While the list has value, it would be more valuable if the recent mail-order purchasers in the list were separated from the people who just like to get catalogues. Furthermore, this second list of hot leads becomes even more valuable to the mail-order house when it has been sorted by zip code, since the post office requires bundling by zip code for bulk mail rates.

dBase orders data through two mechanisms—**sorts** and **indexes**. When we speak of sorting in daily life, we usually mean taking a set of objects and rearranging them in a different sequence. To make an alphabetical sort of 25 file cards, for example, we would spread the 25 cards out on a table, pick the A's, then the B's, and so on until they were in the new order. We had 25 cards at the start and 25 cards at the end.

When a computer sorts, however, it does a conceptually different thing. First it opens the input file of 25 records. In another place on the disk (or any mass storage device), it opens a new file for the sorted output. Then it reads through the input file to determine the order of records, and writes the 25 records to the output file in the correct order. The original file isn't altered in

any way, but a brand-new sort file, containing the same records in a different order, is created.

While this is terribly oversimplified (there are whole books on sorting), the definitional distinction of a sort in dBase is that a sort creates a new file the same size as the sorted file, but under a new name and in a new order.

Indexes in dBase are a different matter altogether. Like sorts, indexes give order to the records in a data base, but through a different mechanism.

A good analogy for understanding indexes can be seen in a college library. The open stacks of a large library may contain several miles of books, and even though these books have been sorted twice (by subject and by author within the subject), it would nonetheless represent a formidable errand to locate a given book by starting at the top shelf of the top rack on the top floor, and going through sequentially.

It's much more practical to start a search in the two indexes that the librarians maintain in the lobby, the card catalogues by author, title and by subject. In these indexes, each entry or card keeps a search key for an individual book in the form of an author or subject listing, and associates that key with a "pointer" to the book in the form of a Library of Congress or Dewey decimal number. Armed with the book's number, and after an intermediate stop at a library map that shows where various number categories are shelved, we can walk directly to the book.

dBase indexes work much the same way. Technically speaking, dBase is an **Index Sequential Access Method (ISAM)** data base (see Fig. 1-1), meaning that the records in dBase are stored sequentially as they're entered. Rapid access to those records depends on one or more indexes being constructed and kept up-to-date.

If we allow some liberties with the library analogy, an ISAM system can be described in terms of a librarian without a shred of imagination in sorting books, but infinite patience and speed in cataloguing and retrieving them. When our librarian received the library's first book, he would put it in position 1 on his shelves no matter what it was about. As more books came in, he'd put them sequentially next to each other, filling his library from left to right, top to bottom.

The only saving grace to this chaotic situation would be that our idiot savant librarian would instantly post each new book in as many different card files as we defined for him. We could index by author, title, subject, Library of Congress number, and if we wanted, by publisher. Then if we needed a book, we could simply tell the librarian a fact about it, such as who wrote it, and he'd be back in a flash with the right book.

In dBase, indexes are preferred over sorts in imposing order on a data base. That indexes "put records in order" is a fact. Even though the records are randomly (sequentially) entered into the data base, dBase is fast enough so that indexing records makes them appear to have been sorted.

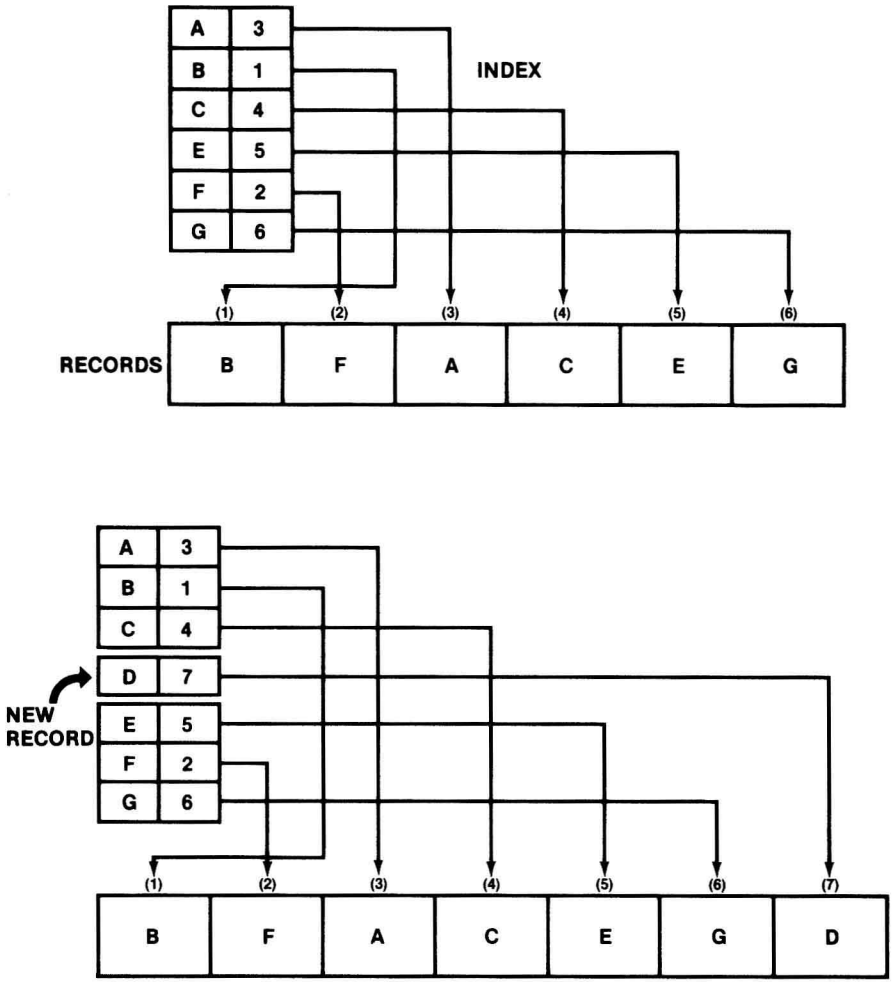


Fig. 1-1. An ISAM (Index Sequential Access Method) Scheme.

Our lightning-fast, infinitely loyal librarian could demonstrate this to us. Imagine standing at the different catalogues, thumbing through the cards one by one, as the librarian rushes the referenced book for each card to us in a microsecond. As we go through the authors (Abernathy, Akerbaum, Astor), we see one library; flipping through the subject catalogue (abalone, aerosols, alcohol), we see another.

RELATIONAL VERSUS HIERARCHICAL DATA BASES

Obviously, data bases may be designed in a variety of different ways. The two most frequently mentioned design concepts in microcomputer books are **relational** data bases and **hierarchical** data bases. In both cases, the adjective used to describe the data base's design refers to the organization of the inter-relationship between the data files in the data base.

Hierarchical Design

The feature that distinguishes a hierarchical data base is the ability of each file listed in a directory to contain a directory of its own. An analogy from the world of paper files will make this clearer.

Let's start by imagining a large manila file folder with the label "Western Nations." When we opened it, we would find a collection of smaller file folders with labels such as "England," "Monaco," "Canada," and "United States." If we were then to open one of these smaller folders, such as the one for the United States, we might find that it contained 50 sub-folders, labeled "Alaska," "Ohio," "Rhode Island," and so on.

Hierarchical data bases are organized along similar lines, with each item in a particular data file having at least the potential of representing a data file of its own (see Fig. 1-2). Such systems can be extremely powerful and flexible, since users can concentrate their attention on whatever task is before them in the data base, without having to consider the level they are on. If we were

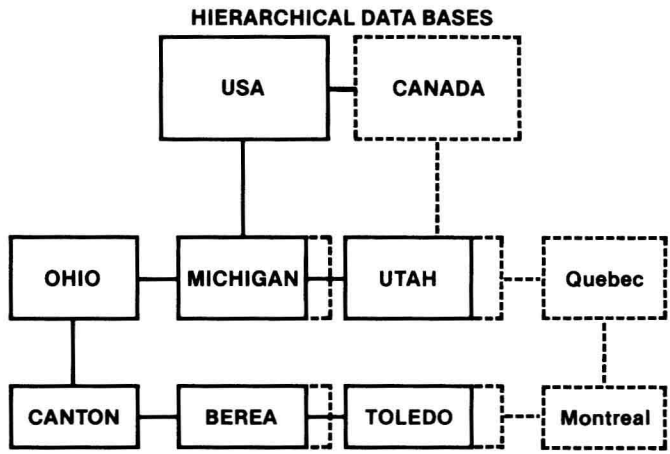


Fig. 1-2. In hierarchical data bases, every subdivision of a record may also have subdivisions.