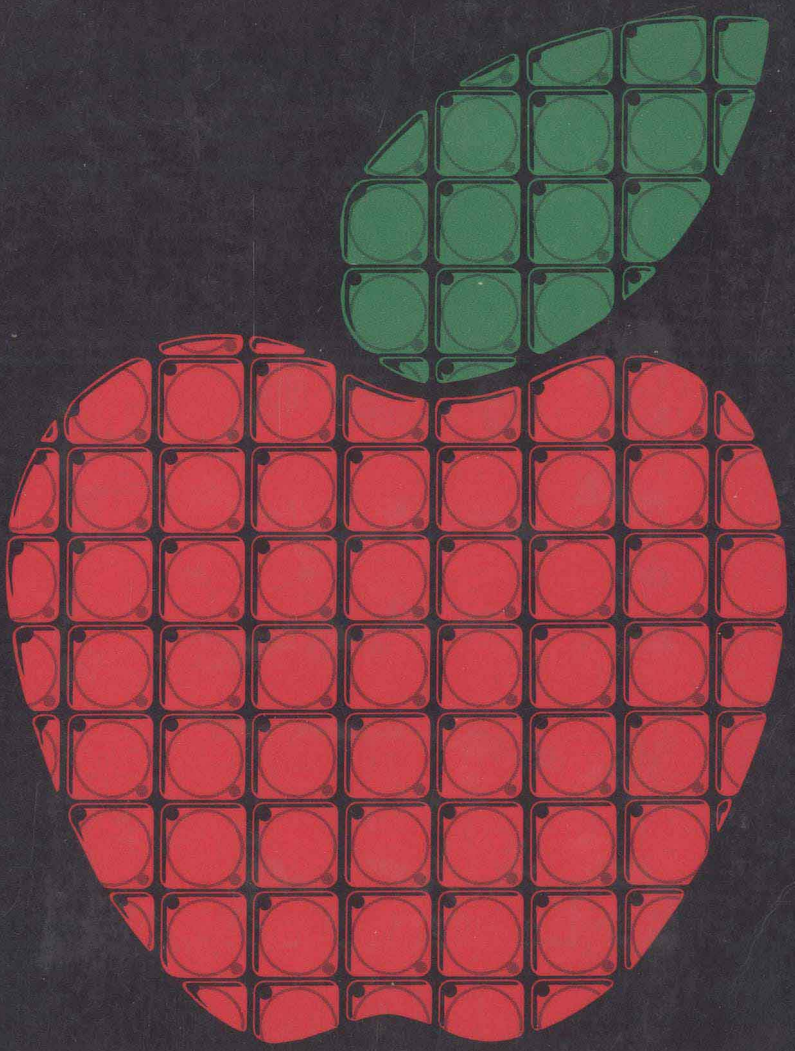
MICROBOOK: DATABASE MANAGEMENT FOR THE APPLE II®

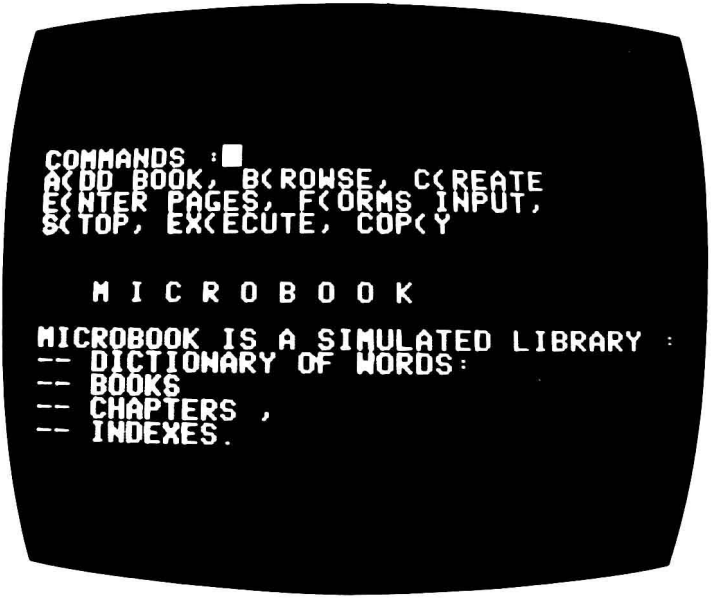


by Ted Lewis

MICROBOOK:

Database Management for the Apple® II Computer

T. G. Lewis



dilithium Press
Beaverton, Oregon

10 9 8 7 6 5 4 3 2

Library of Congress Cataloging in Publication Data

Lewis, T.G. (Theodore Gyle), 1941-
MICROBOOK: database management for the Apple II computer.

Includes index.
1. Apple II (Computer)—Programming. 2. MICROBOOK (Computer system) 3. Data base management. 4. PASCAL (Computer program language) I. Title.
QA76.8.A662L48 001.64'2 82-7457
ISBN 0-88056-072-X (pbk.) AACR2

Cover: Anton C. Kimball

Printed in the United States of America

dilithium Press
P.O. Box 606
Beaverton, Oregon 97075

MICROBOOK:

Database Management for the Apple® II Computer

Preface

MICROBOOK started as a research project in database design and implementation at Oregon State University. The first version of this experimental system grew from lectures given by the author into a crude database system running on the Apple II computer. During the summer of 1981, the author redesigned and re-implemented the experimental version to make it easier to use and faster to run.

My purpose in publishing this book and the associated programs is to make the *MICROBOOK* system widely available at a low cost to microcomputer users. In this spirit, the program diskettes are also available at minimal cost from the publisher. These programs are complete and ready to run. They were tested over a one-year period before they were made available to the public. However, should you find new bugs, please report them so they can be eliminated.

I would like to thank my typist, Donna Lee Norvell-Race for preparing this manuscript, and my publisher, Merl Miller for his support in making *MICROBOOK* generally available.

T. G. Lewis
May 1982

AN IMPORTANT NOTE

The publisher and author have made every effort to assure that the computer programs and programming information in this publication are accurate and complete. However, this publication is prepared for general readership, and neither the publisher nor the authors have any knowledge about or ability to control any third party's use of the programs and programming information. There is no warranty or representation by either the publisher or the authors that the programs or programming information in this book will enable the reader or user to achieve any particular result.

The programs in this book are available on diskette from dilithium Software, a division of dilithium Press. To order the software please use the order form in the back of this book.

Contents

1

What Is a Database System?

1.1 THE PARTS OF A DBMS

A DBMS (database management system) is a collection of programs which store, retrieve, and process data stored in files. More importantly, a DBMS encompasses a *unified, integrated model* of file storage and its corresponding retrieval in order to reduce the labor associated with processing large amounts of data.

This definition of a DBMS may seem a bit abstract, so suppose we look at each part of a DBMS before explaining the details. A DBMS typically includes programs for the following:

- An integrated, unified collection of files. The integrated and unified collection of files is often called the *physical model* of the database.
- An integrated, unified *view* of the physical database. The user's *view* is often called a schema or, as we will call it, a *logical model* of the database. The logical model is how users see the database.
- Query language for processing inquiries from users. These inquiries typically consist of information retrieval or searches for information via a key or collection of keys.
- Input/Output control. Typical DBMS's include input and output control for obvious purposes of data entry and report writing.

1.2 PHYSICAL MODELS

The underlying physical model of a DBMS depends highly on the logical level model. However, nearly all DBMS's must

organize file structures in a manner which leads to (1) high speed retrieval, (2) keyed access on one or more keys, and (3) proper file maintenance.

A common technique for high speed retrieval is called *hashing* or key-to-address transformation. Hashing is used to calculate the location of a piece of information in the database given its key. For example, if a mailing list consisting of name, address, and zipcode were accessed by the name (as a key), we would need to hash the name (alphabetic string) into the corresponding disk file location (number) before we could retrieve the address and zipcode information.

Unfortunately, hashing does not typically provide for rapid access to files which contain multiple-keyed information, nor does hashing facilitate good file maintenance practices. The high speed retrieval of hashing can be used in only the most special cases.

The most flexible and powerful physical level mode of a database is the B-tree structure. A *B-tree index* file is a tree-structured file containing the values of all keys to the information stored in a data file. Each access to the information stored in the database must first access a B-tree index file to find the location of the information. In fact, an access to the B-tree index file is required for every key used to make a retrieval.

Figure 1.1 shows the B-tree index files as triangular objects while the file containing a record of information is shown as a rectangle. Notice that one B-tree index file is used for each key.

A B-tree index file structure also has the advantage that all entries (keys) are stored sorted in alphabetical and/or numerical order. This means we can easily retrieve the records in Figure 1.1 by name, zipcode, or age *in order from smallest to largest.* Hence, JONES, LEWIS, and SMITH are accessed in alphabetical order resulting in records #2, #0, and #1 being retrieved.

MICROBOOK employs a B-tree physical model so that every key in the database is stored in a B-tree index file as well as some master file. In fact, a MICROBOOK database may consist of dozens of B-tree index files, one for each access key needed by the user.

We will not delve into the inner workings of B-tree file structures. For a more detailed explanation the reader is re-

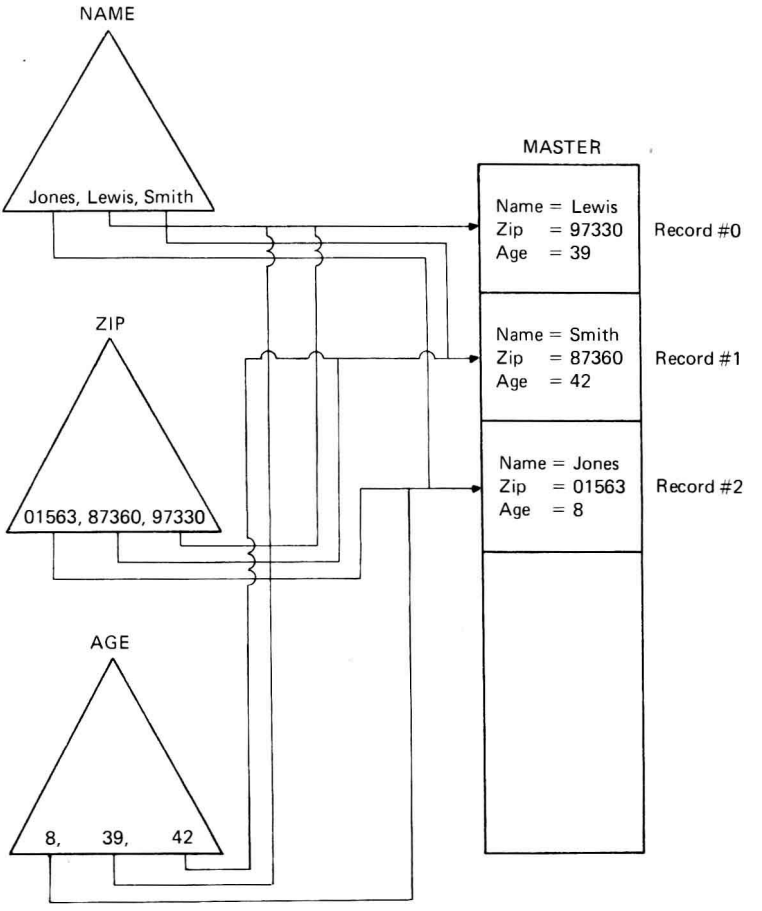


Figure 1.1 Indexed access to multiple-key file

ferred to the author's book, *Pascal Programming For the Apple,* Reston Books, 1981. However, programs for building and accessing keys stored in a B-tree index file are given in this book. See the program listings.

1.3 LOGICAL MODELS

The logical model or logical schema of a database system is the user-perceived structure of the DBMS. It is independent of the physical database and in a way it covers up the physical structure so the user need not think in terms of hashing or B-trees.

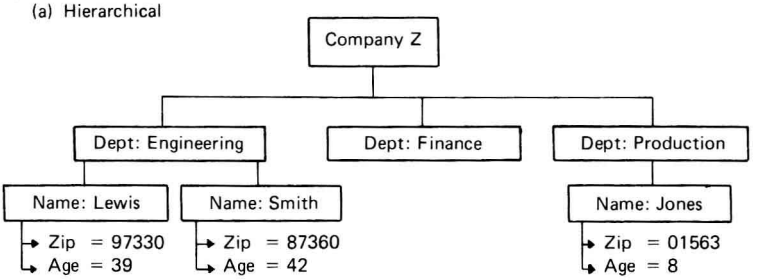
There are three general approaches to designing a logical database:

HIERARCHICAL: Information is viewed as a hierarchical structure with a "top" and "bottom." An example of this view is shown in Figure 1.2(a).

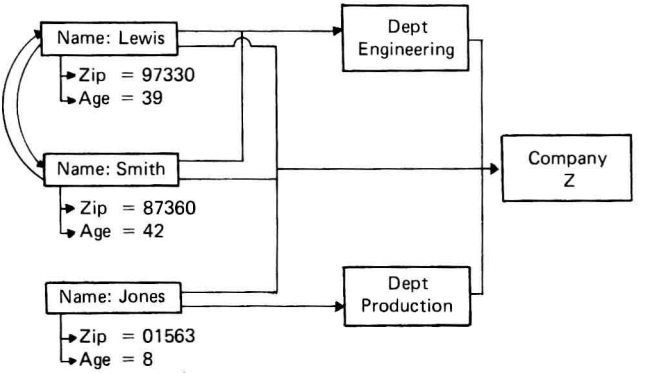
NETWORK: Information is viewed as a network of interacting structures with no apparent structure. This is illustrated by the network of Figure 1.2(b).

RELATIONAL: Information is viewed as a collection of tables called *relations.* This is shown by example in Figure 1.2(c).

(a) Hierarchical



(b) Network



(c) Relational

Company Z

Name:	Dept	Age	Zip
Lewis	Engr.	39	97330
Smith	Engr.	42	87360
Jones	Prod.	8	01563

Figure 1.2 Three logical models

In the hierarchical view every employee is part of a department. Therefore, to access information about an employee, we must search down through the levels of the hierarchy. Furthermore, to retrieve other employees in the same department, we can quickly locate them by searching down to the DEPT level. However, to retrieve all employees of the same age, for example, we may need to search the entire hierarchical structure.

The network view uses links (pointers, as shown by arrows) to associate properties of employees with departments, each other, and so forth. The network view is quite flexible due to the arbitrary linking, but it is not easily implemented at the physical level. Therefore, the network view is rarely employed in practical DBMS's.

The relational view offers the most elegant and simple solution to a DBMS logical organization. Each relation is very similar to a table as shown in Figure 1.2(c). The labeled columns are called *domains* and the rows are called *tuples.* We could retrieve all employees in the same department by searching the domain equal to "ENGR", in Figure 1.2(c). Furthermore, we could retrieve all information associated with an employee by reading a tuple from the relation, e.g., LEWIS, ENGR, 39, 97330. Hence, a tuple closely corresponds to the information stored in an ordinary file.

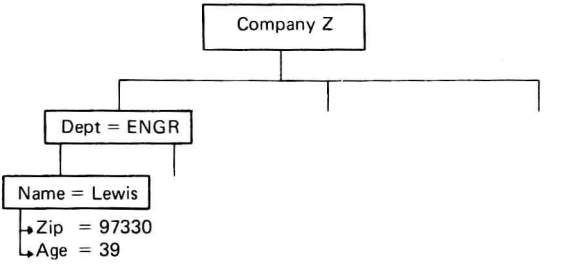
Relational DBMS's are popular because they have such a simple logical structure. However, there are some kinds of processing that a relational DBMS cannot easily perform. For this reason we have adopted a combination of both relational and hierarchical structures for use in MICROBOOK.

1.4 QUERY LANGUAGES

The form of a DBMS query language will conform to the form of its logical view. Thus, a hierarchical query will use tree-searching to locate and output information stored in the hierarchy. For example, the System 2000 query language uses commands to establish a trace through the hierarchy as shown in Figure 1.3(a) and as governed by the query.

The QUALIFY command derives a trace or path from top-to-bottom as shown in Figure 1.3(a). The PRINT statement causes the path to be used to find NAME, AGE, and ZIP, and then print them.

(a) System 2000 Query Qualification (trace).



(b) Relational Query

LEWIS	ENGR	39	97330

(c) Microbook Query (Turn to correct page in the book)

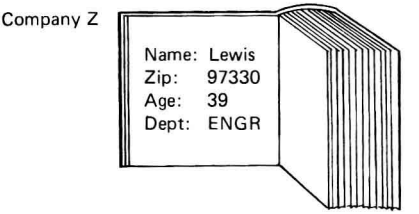


Figure 1.3 Query processing

This approach is used in MICROBOOK to browse the database. However, the simplicity and convenience of a relational query language is needed in more sophisticated processing. Therefore, MICROBOOK employs a very high-level query language much like the query language of INGRES. Here is an example of an INGRES query:

```
RANGE OF T IS COMPANY
RETRIEVE (T. ALL) WHERE NAME = "LEWIS"
```

The INGRES query above uses keyword ALL to abbreviate the names of all domains in the tuple corresponding to "LEWIS". This has a corresponding keyword in the Query Language Processor of MICROBOOK. In MICROBOOK we simply enter the statement

```
FOR *NAME="LEWIS" to "LEWIS" DO;
?;
END ;
```

This statement causes all records with NAME equal to "LEWIS" to be displayed on the screen of the computer. For more details on the Query Language Processor turn to Chapter 9, which describes this system. Also, the Pascal program for implementing QUERY can be found at the end of this book.

In the following chapters we will discuss and illustrate by example the MICROBOOK database system implemented on the Apple II microcomputer. All programs (source and object) can be obtained by ordering them from the publisher. See the ordering card attached to this book.