Data Base Computers

Olin H. Bray
Harvey A. Freeman

Lexington Book

Data Base Computers

Olin H. Bray
Harvey A. Freeman
Sperry Univac

LexingtonBooks
D.C. Heath and Company
Lexington, Massachusetts
Toronto

Library of Congress Cataloging in Publication Data

Bray, Olin H.
Data base computers.

Bibliography: p.
Includes index.
1. Data base management. 2. Electronic digital computers. I. Freeman, Harvey A., joint author. II. Title.
QA76.9.D3B7 001.6'4 78-24765
ISBN 0-669-02834-7

Third printing, July 1981

Published simultaneously in Canada.

Printed in the United States of America.

International Standard Book Number: 0-669-02834-7

Library of Congress Catalog Card Number: 78-24765

Data Base Computers

Lexington Books Series in Computer Science

Kenneth J. Thurber, General Editor

Computing in the Humanities
Peter C. Patton

Data Base Computers
Olin H. Bray and *Harvey A. Freeman*

Data Structures and Computer Architecture
Kenneth J. Thurber and *Peter C. Patton*

Distributed Database Management Systems
Olin H. Bray

Distributed-Processor Communications Architecture
Kenneth J. Thurber and *G.M. Masson*

Requirements-Oriented Computer System Design
Kenneth J. Thurber

Foreword

The presence of the book by Olin H. Bray and Harvey A. Freeman marks an important turning point in the annals of computing. The point signifies the departure of traditional emphasis on nonnumeric processing of data by general-purpose mainframes and the arrival of innovative solutions to nonnumeric processing of data by special-purpose backends. One of the most important backends is the data base computer. This machine is intended to cost-effectively replace the conventional data base management software, on-line I/O routines, and on-line secondary storage (currently supported by the general-purpose mainframe) while offering better performance.

The timely arrival of data base computers is prompted by a number of factors: the user's application requirement for very large data bases and for a wide range of data base activities, the enhancement of existing and the configuration of new technologies, the performance bottlenecks of the present-day software-laden data base management system, the lack of reliable and secure software solutions to data base management, and better research and understanding of hardware approaches to data base computer architecture. Bray and Freeman have skillfully articulated these factors.

More important, these two talented authors attempt to review the data base computer architecture work to date, classify the various architectures, and compare the different approaches to data base computer architecture. Such a review, classification, and comparison, although difficult to develop, provides the reader a much-needed perspective and understanding of the field of data base computer architecture.

Finally, the authors are congratulated for their concluding remarks about the trend of data base management and the future of data base computers. As practicing computer architects, they should be taken seriously on their views of the upcoming data base management software and hardware evolution.

Since this is the first book written on data base computers, it should be required reading for every serious computer science and engineering practitioner.

David K. Hsiao
The Ohio State University

Preface

Data base computers, first proposed by researchers in various universities in the early 1970s, are now being developed by commercial computer manufacturers. Within the next few years, these special-purpose devices will be available to meet the information storage and processing needs of the 1980s. Which designs will actually be implemented and which ones will achieve a measure of success is not clear at present. What we can say at this time is that we expect a wide variety of data base computers to be offered, with attractive features and capabilities.

This book was written to familiarize the reader with data base management requirements and data base computer concepts. Until now, no systematic means have been available for comparing and evaluating current or future data base computer designs. This book will be useful to designers in their efforts to achieve the benefits attributed to these special-purpose machines. It also provides future purchasers and users of data base computers with the ability to specify the features and capabilities required for their applications and the knowledge to use them effectively. Finally, this book will extend the vision of computer science and engineering students and practitioners in yet another area of the exciting technological realm of the computer.

We thank George Champine and Ken Thurber of Sperry Univac and David Hsiao of The Ohio State University for their help and encouragement in writing this book. The assistance of Jayanta Banerjee, who contributed the CODASYL DMS comparison of chapter 4 and reviewed our manuscript, is greatly appreciated. We also thank Jerry Bill who reviewed our manuscript and Jan Bray who assisted with the Index. Special thanks are due to Linda Sorenson and Penny Svenkeson who contributed their data entry and text editing skills as our deadline rapidly approached.

Contents

List of Figures

List of Tables

1 Introduction

Data base management systems (DBMS) development is facing a challenge to its continued growth and effectiveness. As the variety of functions of DBMSs increases, so does the accompanying overhead. Critical response time and the throughput requirements are not met because of the implementation of DBMSs on unsuitable architectures. A way must be found to reduce this overhead, increase the performance of data management systems, and allow more of the processing power of computer systems to be used in the applications.

The challenge stems from the incompatability between traditional computer hardware architecture and the functional requirements of data base management. General-purpose computers locate data (whether stored in main memory or on secondary storage devices) by position or address, whereas most data management applications employ data by content or value. Significant processing and storage overhead is necessary, therefore, to convert from one reference scheme to another. This overhead can become so extreme that in some data bases the indexes and tables to locate the data require more storage than the data itself.

Since the early 1970s, at least eight universities (Ohio State, Toronto, Syracuse, Florida, Texas, Utah, MIT, and Kansas State) have initiated research projects to address this challenge. The results of this research are now being used by various computer manufacturers to develop commercial products in this area. In October 1977, International Computers Limited (ICL) "announced" CAFS (Content Addressable File Store),[1] which is similar to some of the work done on CASSM (Context Addressed Segment Sequential Memory) at the University of Florida.[2] The University of Toronto's RAP (Relational Associative Processor) project has designed and built a prototype relational architecture which Intel considered implementing.[3] Syracuse University and the Rome Air Development Center are jointly investigating the applicability of STARAN, Goodyear Aerospace's associative processor,[4] for data base management applications.[5] Sperry Univac is working closely with David Hsiao at The Ohio State University in exploring the commercial viability of his DBC (Data Base Computer).[6]

The use of back-end data base computers is similar to the use of front-end message processors. As users required increasingly complex and sophisticated communications and message processing capabilities, these functions were distributed to a specially designed, separate processor. Data base systems are now approaching a comparable stage of development. The use of DBMSs has become widespread enough to consider the possible effi-

ciencies and performance improvements of distributing the data base management function to separate specialized devices or computers.

As with message processing, some of the early work with data base computers, such as Canaday's at Bell Labs, proposed the use of general-purpose processors as the backend.[7] However, new technologies such as microprocessors, VLSI, charge-coupled devices, and magnetic bubble memories, incorporated in special-purpose data base computers (DBC) in conjunction with such design approaches as content addressing and parallel processing, appear better suited to the information storage and processing needs of the 1980s.

Although it is assumed that the reader is familiar with data base concepts, this first chapter provides a brief review. The data management trend of providing greater funtionality in the system software to relieve the application programmer of the routine data management burden is discussed. Also identified are the basic data base management objectives. Then the components and operation of a DBMS are described. This concludes with a treatment of the data base computer concept. The concept of the DBC is described, several classifications are noted, and the advantages and disadvantages of the DBC approach are discussed.

Chapter 2 identifies the requirements of DBMSs. These requirements exist regardless of the way the system is implemented. Three types of requirements—basic, extended, and future—are identified. The basic ones, such as retrieval, update, control, and data administration, are already available in most current DBMSs and therefore must be available in any commercial DBC system. Extended requirements are those which are just being introduced in the more advanced DBMSs. Future requirements cover those which are rarely available today, but which are anticipated to become widespread in the future. These include data compression, encryption, and value-based security criteria.

Chapter 3 describes the example data base which will be used as a means for comparing the various DBC architectures. The example is a production data base for a typical manufacturing application. The set of queries to be processed against the data base is identified, and three data base structures are shown. The first structure is the conventional CODASYL or network structure. Because many DBC architectures are oriented toward the relational model, however, the second and third structures are relational. The second structure is a relational data base as it would be designed given the set of queries to be used. The third alternative is the relational data base that would be obtained using a conversion algorithm to convert the data base from the network to the relational model.

The fourth chapter provides a set of criteria against which to evaluate various DBC architectures. These criteria include performance, cost, range, and evolvability.

Chapter 5 first provides a way of classifying various DBC architectures based on where the data base is searched and on the number of processors involved. Then the benefits and problems of each of the resulting major architectural types are discussed. All the current DBC systems are located within this framework. Finally, a representative example of each type is examined in detail. This includes a description of both the hardware and the way the system would process the example data base described in chapter 3.

Chapter 6 draws some conclusions and makes some projections about future developments in the DBC area. Both data management and system architecture, that is, hardware and software, trends are identified.

Following chapter 6 is a bibliography for those individuals who would like further information about the concepts and systems treated in this book. This bibliography is divided into the major categories of data base management systems, data base computers, and miscellaneous items. These major categories are further divided to assist the reader in locating references to a particular topic.

1-1. Data Base Systems

In data management the basic trend is to relieve the programmer of most of the routine aspects of data management. In the early 1950s, the programmer had to do all of his own I/O, and at a very low level. The actual application and how the data were to be used was only a part of his concern. Much of the I/O programming involved testing the status of the I/O device, readying the device, issuing the appropriate operations (which were different for different types of devices), waiting for the device to complete the operations, rechecking the device status, and perhaps redoing the entire sequence if there was an error. Only at this point could the programmer be sure the data had been properly read or written.

Much of this activity, however, was of no interest to the applications programmer, who simply wanted to read or write a block of data and to be notified if there was a problem. Also much of this programming was determined by the I/O device being used, rather than by the nature of the data. Because these functions were the same for all applications using a particular type of device, computer manufacturers began supplying them in the form of a common set of software. These early IOCSs (input-output control systems) relieved the programmer of the routine burden of directly managing the computer hardware. Application programmers could concentrate on the application, while a few knowledgeable systems programmers could develop I/O routines to make the most efficient use of these specific devices. While these improvements were important for first- and second-generation systems, they were critical for third-generation systems where

multiple users are processed concurrently during each other's I/O wait times. I/O, which had been both an application and a system bottleneck, was now primarily an application bottleneck because multiprogramming and time-sharing systems allowed the system to make more effective use of the I/O wait times. All these improvements were in the use of hardware and were independent of the data content and organization.

A similar trend occurred for data content. The applications programmer knew the form of the data as they came from the devices and the form in which they were needed by the application. Therefore, he or she converted the data before they were used. If the file containing the data base was labeled, the label had to be processed differently from the data in the file. If the tape was blocked, it had to be deblocked and individual records passed to the application. Thus label processing and blocking and buffering were absorbed in the vendor-provided software and became a system rather than an application programmer concern.

Sorting and merging of files also occurred frequently enough that most computer manufacturers and many independent software houses provided standard SORT/MERGE packages which the programmer could use by specifying only a few parameters.

Because many files were being written and read by programs written in high-level languages such as FORTRAN and COBOL, data fields required certain standard conversions. In FORTRAN, these conversions are specified in the format statements. In COBOL, the data division provides this data conversion and structure information. While the applications programmer had to know the format of the file and had to specify which conversions were required, he or she was not concerned with how the conversions were done. These routine operations also were performed by the system software.

Thus the basic trend was that as common data manipulation requirements were identified, system software routines and utilities were developed. Programmers then performed these functions simply by specifying a few required parameters rather than by writing the entire program.

Even the most sophisticated file system, however, has very little information about the content of the file it is processing. The development of file systems proceeded using a bottom-up approach: "Here is a function being done by many applications. Can that function be more effectively provided by system software?" Instead of reexamining the overall requirements of application development, the designers of these data management systems accepted the file approach as given. Certain basic characteristics of file systems, however, limited their ability to continue to relieve the applications programmer of additional data management functions.

There are three basic characteristics of file processing. First, a file is designed for a specific application. It is used by only a few programs, fre-

quently one to create or update the file and one to read and process it. This application-specific design leads to the second characteristic of file processing. The definition of the file's logical (and much of its physical) structure and content is implicit in the program using it. Therefore, the program must know the file's data structure and format. Finally, because the file system knows little or nothing about the file's content, it can provide only basic data management support for the applications programmer. Because of these characteristics, file systems allow only a few simple organizational structures, that is, sequential, indexed sequential, and random.

These file-processing characteristics create three major problems: integrity, consistency, and maintainability. The integrity problem arises because the file system does not know enough about the file to automatically check the validity of the data being entered.

The consistency and maintainability problems arise because there are cases where the same data may be needed by several applications. For example, the cost of an item in inventory is needed for inventory management, accounts/payable, and for reporting profit margins by item. When this overlap occurs, the systems analyst has two options. One is that two or more files can be used, although the supposedly common data, for example, the cost of the item, may not agree. This is particularly true if the files are updated at different times. This creates the consistency problem. The same data item has a different value depending on the file from which it was read. When this inconsistency becomes apparent to the user, the information system loses its credibility.

This inconsistency can be avoided by combining the files into a single file which contains the data needed by all the applications. This creates a maintenance problem. When one of the programs is modified and needs additional data, the file must be changed. The other programs which use the file, however, make certain assumptions about its structure. Therefore, when the file is changed to meet the new needs of the first program, the other programs also will have to be changed so that they can continue to use the file. Thus there is a tradeoff between consistency and maintainability of the data. Data base management (software) systems were developed to eliminate or ease these problems.

1-2. Data Base Management Systems

Objectives

The basic objective of any data base management system (DBMS) is to improve an organization's control and utilization of its data resources. This is accomplished through improvements in the availability, integrity, and

security of the data base. These are basic data management objectives and are applicable regardless of whether or not there is a data base computer (hardware) system.

The first data base objective, availability, has three components, the first of which is ease of use. The easier it is to learn the system and access the data, the more the system will be used, particularly by the nonprogramming user.

The second component involves allowing multiple users to access the data simultaneously. Certain controls must be provided, however, to prevent destructive interference caused by multiple users trying to concurrently update the same data. The performance aspects of multiple-user availability are response time and throughput. If the response time and throughput are degraded too much, the result is that the data are not available.

The third availability component is long-term availability or evolvability. This means that the data base and the DBMS must be flexible so that they can evolve to meet changing user needs. This evolvability requires a generalized rather than a specialized system and a programming language interface so that system modifications can be made as new requirements are identified.

The second requirement for a DBMS is improved data integrity. Data integrity in turn has two components: data quality and data existence. Data quality first requires providing a complete definition of the data base so that the DBMS can perform much of the validity checking automatically. Anything that has been specified previously in the data definition or schema can be checked and maintained by the system. Second, data quality requires access control to prevent multiple users from concurrently updating the same part of the data base and destroying its accuracy.

Integrity also involves the existence of the data base. This protection requires appropriate backup and recovery methods. The recovery methods must allow for both a complete or partial loss of the data base and an erroneous entry. The latter situation is made more difficult because an incorrect entry may not be detected for some time, during which its effects are propogated to other parts of the data base. Recovery must then correct both the original and all subsequent errors.

The third objective of DBMSs involves the security or privacy aspects of the data base. Because a firm's data resources are an invaluable asset, unauthorized personnel should not be allowed access to it. Thus a second type of access control that the DBMS must provide becomes apparent.

If the data base contains information of a personal nature, there is also a question of privacy. The person(s) involved as the object of these data should have some control over its distribution and use. Access to the data must be limited to the authorized purposes. Although this area has fre-