PRINCIPLES OF DATABASE AND KNOWLEDGE-BASE SYSTEMS

VOLUME I

Jeffrey D. Ullman
STANFORD UNIVERSITY

PRINCIPLES OF DATABASE AND KNOWLEDGE-BASE SYSTEMS

VOLUME I

Jeffrey D. Ullman
STANFORD UNIVERSITY

COMPUTER SCIENCE PRESS

Computer Science Press
1803 Research Boulevard
Rockville, Maryland 20850

Library of Congress Cataloging-in-Publication Data

Ullman, Jeffrey D., 1942-
Principles of database and knowledgebase systems.

(Principles of computer science series, ISSN 0888-2096 ; 14-)
Bibliography: p.
Includes index.
1. Data base management. 2. Expert systems (Computer science) I. Title.
II. Series. Principles of computer science series; 14, etc.
QA76.9.D3U443 1988 005.74 87-38197
ISBN 0-7167-8158-1

3 4 5 6 7 8 9 0 RRD 9 9 8 7 6 5 4 3 2 1 0

PRINCIPLES OF COMPUTER SCIENCE SERIES

ISSN 0888-2096

Series Editors

Alfred V. Aho, Bell Telephone Laboratories, Murray Hill, New Jersey
Jeffrey D. Ullman, Stanford University, Stanford, California

1. *Algorithms for Graphics and Image Processing**
   Theo Pavlidis
2. *Algorithmic Studies in Mass Storage Systems**
   C. K. Wong
3. *Theory of Relational Databases**
   Jeffrey D. Ullman
4. *Computational Aspects of VLSI**
   Jeffrey D. Ullman
5. *Advanced C: Food for the Educated Palate**
   Narain Gehani
6. *C: An Advanced Introduction**
   Narain Gehani
7. *C for Personal Computers: IBM PC, AT&T PC 6300, and Compatibles**
   Narain Gahani
8. *Principles of Computer Design**
   Leonard R. Marino
9. *The Theory of Database Concurrency Control**
   Christos Papadimitriou
10. *Computer Organization**
    Michael Andrews
11. *Elements of Artificial Intelligence Using LISP*
    Steven Tanimoto
12. *Trends in Theoretical Computer Science*
    Egon Börger, Editor
13. *An Introduction to Solid Modeling*
    Martti Mäntylä
14. *Principles of Database and Knowledge Base Systems, Volume I*
    Jeffrey D. Ullman

*These previously-published books are in the *Principles of Computer Science Series* but they are not numbered within the volume itself. All future volumes in the *Principles of Computer Science Series* will be numbered.

OTHER BOOKS OF INTEREST

Jewels of Formal Language Theory
Arto Salomaa

Principles of Database Systems
Jeffrey D. Ullman

Fuzzy Sets, Natural Language Computations, and Risk Analysis
Kurt J. Schmucker

LISP: An Interactive Approach
Stuart C. Shapiro

PREFACE

This book is the first of a two-volume set that is intended as a replacement for my earlier book *Principles of Database Systems* (Ullman [1982] in the refer ences). Since the latter book was written, it became clear that what I though of as "database systems" formed but one (important) point in a spectrum c systems that share a capability to manage large amounts of data efficientl but differ in the expressiveness of the languages used to access that data. I has become fashionable to refer to the statements of these more expressive lan guages as "knowledge," a term I abhor but find myself incapable of avoiding Thus, in the new book I tried to integrate "classical" database concepts with the technolgy that is just now being developed to support applications where "knowledge" is required along with "data."

The first volume is devoted primarily to classical database systems. How ever, knowledge, as represented by logical rules, is covered extensively in Chap ter 3. From that chapter, only the material on relational calculus, a "classical" database topic, is used extensively in this volume. We shall return to the topic of logic as a user interface language in the second volume, where it is one o the major themes. We also find in the first volume a discussion of "object oriented" database systems, which, along with "knowledge-base systems," is an important modern development.

Chapter 1 introduces the terminology for database, object-base, and knowledge-base systems; it attempts to explain the relationships among these systems, and how they fit into an unfolding development of progresssively more powerful systems. Chapters 2 and 3 introduce us to data models as used in these three classes of systems; "data models" are the mathematical abstrac tions we use to represent the real world by data and knowledge. In Chapter 4 we meet several important query languages that are based on the relational data model, and in Chapter 5 we meet languages that are based on one of several "object-oriented" models.

Chapter 6 covers physical organization of data and the tricks that are used to answer, efficiently, queries posed in the languages of Chapters 4 and 5. Then, in Chapter 7 we discuss some of the theory for relational database systems, especially how one represents data in that model in ways that avoid redundancy and other problems. Chapter 8 covers security and integrity aspects of database systems, and in Chapter 9 we discuss concurrency control, the techniques that make it possible for many processes to operate on one database

simultaneously, without producing paradoxical results. Finally, in Chapter 10 we consider techniques for dealing with distributed database systems.

It is expected that the second volume will cover query optimization techniques, both for "classical" database systems (chiefly relational systems) and for the new class of "knowledge-base" systems that are presently under development, and which will rely heavily on the optimization of queries expressed in logical terms. We shall also find a discussion of some of these experimental systems. Finally, Volume II will cover "universal relation" systems, a body of techniques developed to make sense of queries that are expressed in natural language, or in a language sufficiently informal that the querier does not have to know about the structure of the database.

Mapping the Old Book to the New

Readers familiar with Ullman [1982] will find most of that material in this volume. Only the chapters on optimization and on universal relations are deferred to Volume II, and a few sections of the old book have been excised. The material in the old Chapter 1 has been divided between the new Chapters 1 and 2. Sections 1.1 and 1.2 remain in Chapter 1, while Sections 1.3 and 1.4 form the core of Chapter 2 (data models) in the new book. Chapter 2 of the old (physical organization) now appears in Chapter 6, along with material on physical organization that formerly appeared in Sections 3.2, 4.2, and 5.1. Some of the material in the old Section 2.8 (partial-match queries) has been excised.

The remainders of Chapters 3 and 4 (network and hierarchical languages) appear in the new Chapter 5 (object-oriented langauges), along with new material on OPAL, which is a true, modern object-oriented language for database systems. The old Chapter 5, on the relational model, has been dispersed. Section 5.1, on physical structures, moves to Chapter 6, Section 5.2, on relational algebra, moves to Chapter 2 (data models), while Section 5.3, on relational calculus, moves to Chapter 3 (logic and knowledge). The old Chapter 6 (relational languages) becomes the new Chapter 4. The discussion of the language SQUARE has been omitted, but the language SQL is covered much more extensively, including an example of how SQL can be interfaced with a host language, C in particular.

Only Chapter 7 (relational theory) remains where it was and remains relatively unchanged. A discussion of the Tsou-Fischer algorithm for constructing Boyce-Codd normal form schemes is included, as well as a pragmatic discussion of the virtues and dangers of decomposition or "normalization." Chapters 8 (query optimization) and 9 (universal relation systems) are deferred to the second volume. Chapter 10 (security and integrity) becomes Chapter 8. The discussion on statistical databases is excised, but more examples, drawn from SQL and OPAL, are included. Chapter 11 (concurrency) becomes Chapter 9, and is expanded in several ways. Chapter 12 (distributed systems) is divided

in two. The first half, on query optimization for distributed systems, is moved to Volume II, while the second half forms the core of the new Chapter 10; the latter includes not only distributed locking, but also covers other issues such as distributed agreement ("distributed commit").

Exercises

Each chapter, except the first, includes an extensive set of exercises, both to test the basic concepts of the chapter and in many cases to extend these ideas. The most difficult exercises are marked with a double star, while exercises of intermediate difficulty have a single star.

Acknowledgements

The following people made comments useful in the preparation of this volume: David Beech, Bernhard Convent, Jim Cutler, Wiebren de Jonge, Michael Fine, William Harvey, Anil Hirani, Arthur Keller, Michael Kifer, Hans Kraamer, Vladimir Lifschitz, Alberto Mendelzon, Jaime Montemayor, Inderpal Mumick, Mike Nasdos, Jeff Naughton, Meral Ozsoyoglu, Domenico Sacca, Shuky Sagiv, Yatin Saraiya, Bruce Schuchardt, Mary Shaw, Avi Silberschatz, Leon Sterling, Rodney Topor, Allen Van Gelder, Moshe Vardi, and Elizabeth Wolf.

Alberto Mendelzon, Jeff Naughton, and Shuky Sagiv also served as the publisher's referees. My son Peter Ullman developed some of the TEX macros used in the preparation of this manuscript. The writing of this book was facilitated by computing equipment contributed to Stanford University by ATT Foundation and by IBM Corp.

Corrections to the first printing were provided by: Francisco Carrasco, Chen-Lieh Huang, Elie Kanaan, Dorothee Koch, Byung-Suk Lee, Mike Migliore, Inderpal Mumick, Geoff Phipps, and Yumi Tsugi.

Old Debts

The two editions of Ullman [1982] acknowleged many people who contributed to that book, and many of these suggestions influenced the present book. I thank in this regard: Al Aho, Brenda Baker, Dan Blosser, Martin Brooks, Peter deJong, Ron Fagin, Mary Feay, Shel Finkelstein, Vassos Hadzilacos, Kevin Karplus, Zvi Kedem, Arthur Keller, Hank Korth, Keith Lantz, Dave Maier, Dan Newman, Mohammed Olumi, Shuky Sagiv, Charles Shub, Joe Skudlarek, and Joseph Spinden.

Gerree Pecht, at Princeton, typed the first edition of the old book; vestiges of her original troff can be found in the TEX source of this volume. Luis Trabb-Pardo assisted me in translation of Ullman [1982] from troff to TEX.

J. D. U.
Stanford CA

TABLE OF CONTENTS

CHAPTER 1

Databases, Object Bases, and Knowledge Bases

A database management system (DBMS) is an important type of programming system, used today on the biggest and the smallest computers. As for other major forms of system software, such as compilers and operating systems, a well-understood set of principles for database management systems has developed over the years, and these concepts are useful both for understanding how to use these systems effectively and for designing and implementing DBMS's. In this book we shall study the key ideas that make database management systems possible. The first three sections of this chapter introduce the basic terminology and viewpoints needed for the understanding of database systems.

In Section 1.4, we discuss some of the newer applications for which the classical form of database management system does not appear to be adequate. Then, we discuss two classes of enhanced DBMS's that are of rising importance. In Section 1.5 we mention "object-base" systems and discuss how they solve the problems posed by the new applications. Section 1.6 introduces us to "knowledge systems," which are generally systems implementing logic, in one or another form, as a programming language. A "knowledge-base management system" (KBMS) is then a programming system that has the capabilities of both a DBMS and a knowledge system. In essence, the highly touted "Fifth Generation" project's goal is to implement a KBMS and the hardware on which it can run efficiently. The relationships among these different kinds of systems are summarized in Section 1.7.

The reader may find some of the material in this chapter difficult to follow at first. All important concepts found in Chapter 1 will be covered in greater detail in later chapters, so it is appropriate to skim the material found here at a first reading.

1.1 THE CAPABILITIES OF A DBMS

There are two qualities that distinguish database management systems from other sorts of programming systems.

1. The ability to manage persistent data, and
2. The ability to access large amounts of data efficiently.

Point (1) merely states that there is a *database* which exists permanently; the contents of this database is the data that a DBMS accesses and manages. Point (2) distinguishes a DBMS from a file system, which also manages persistent data, but does not generally help provide fast access to arbitrary portions of the data. A DBMS's capabilities are needed most when the amount of data is very large, because for small amounts of data, simple access techniques, such as linear scans of the data, are usually adequate. We shall discuss this aspect of a DBMS briefly in the present section; in Chapter 6 the issue of access efficiency is studied in detail.

While we regard the above two properties of a DBMS as fundamental, there are a number of other capabilities that are almost universally found in commercial DBMS's. These are:

a) Support for at least one *data model*, or mathematical abstraction through which the user can view the data.
b) Support for certain high-level languages that allow the user to define the structure of data, access data, and manipulate data.
c) *Transaction management*, the capability to provide correct, concurrent access to the database by many users at once.
d) *Access control*, the ability to limit access to data by unauthorized users, and the ability to check the validity of data.
e) *Resiliency*, the ability to recover from system failures without losing data.

Data Models

Each DBMS provides at least one abstract model of data that allows the user to see information not as raw bits, but in more understandable terms. In fact, it is usually possible to see data at several levels of abstraction, as discussed in Section 1.2. At a relatively low level, a DBMS commonly allows us to visualize data as composed of files.

Example 1.1: A corporation would normally keep a file concerning its employees, and the record for an employee might have fields for his first name, last name, employee ID number, salary, home address, and probably dozens of other pieces of information. For our simple example, let us suppose we keep in the record only the employee's name and the manager of the employee. The record structure would look like:

```
record
    name:  char[30];
    manager:  char[30];
end
```

The file itself is a sequence of records, one for each employee of the company. □

In many of the data models we shall discuss, a file of records is abstracted to what is often called a *relation*, which might be described by

EMPLOYEES(NAME, MANAGER)

Here, EMPLOYEES is the name of the relation, corresponding to the file mentioned in Example 1.1. NAME and MANAGER are field names; fields are often called *attributes*, when relations are being talked about.

While we shall, in this informal introductory chapter, sometimes use "file" and "relation" as synonyms, the reader should be alert to the fact that they are different concepts and are used quite differently when we get to the details of database systems. A relation is an abstraction of a file, where the data type of fields is generally of little concern, and where order among records is not specified. Records in a relation are called *tuples*. Thus, a file is a list of records, but a relation is a set of tuples.

Efficient File Access

The ability to store a file is not remarkable; the file system associated with any operating system does that. The capability of a DBMS is seen when we access the data of a file. For example, suppose we wish to find the manager of employee "Clark Kent." If the company has thousands of employees, it is very expensive to search the entire file to find the one with NAME = "Clark Kent". A DBMS helps us to set up "index files," or "indices," that allow us to access the record for "Clark Kent" in essentially one stroke, no matter how large the file is. Likewise, insertion of new records or deletion of old ones can be accomplished in time that is small and essentially constant, independent of the file's length. An example of an appropriate index structure that may be familiar to the reader is a hash table with NAME as the key. This and other index structures are discussed in Chapter 6.

Another thing a DBMS helps us do is *navigate* among files, that is, to combine values in two or more files to obtain the information we want. The next example illustrates navigation.

Example 1.2: Suppose we stored in an employee's record the department for which he works, but not his manager. In another file, called DEPARTMENTS, we have records that associate a department's name with its manager. In the style of relations, we have:

EMPLOYEES(NAME, DEPT)
DEPARTMENTS(DEPT, MANAGER)

Now, if we want to find Clark Kent's manager, we need to navigate from EMPLOYEES to DEPARTMENTS, using the equality of the DEPT field in both files. That is, we first find the record in the EMPLOYEES file that has NAME = "Clark Kent", and from that record we get the DEPT value, which we all know is "News". Then, we look into the DEPARTMENTS file for the record having DEPT = "News", and there we find MANAGER = "Perry White". If we set up the right indices, we can perform each of these accesses in some small, constant amount of time, independent of the lengths of the files. □

Query Languages

To make access to files easier, a DBMS provides a *query language*, or *data manipulation language*, to express operations on files. Query languages differ in the level of detail they require of the user, with systems based on the relational data model generally requiring less detail than languages based on other models.

Example 1.3: The query discussed in Example 1.2, "find the manager of Clark Kent," could be written in the language SQL, which is based on the relational model of data, as shown in Figure 1.1. The language SQL will be taught beginning in Section 4.6. For the moment, let us note that line (1) tells the DBMS to print the manager as an answer, line (2) says to look at the EMPLOYEES and DEPARTMENTS relations, (3) says the employee's name is "Clark Kent," and the last line says that the manager is connected to the employee by being associated (in the DEPARTMENTS relation) with the same department that the employee is associated with (in the EMPLOYEES relation).

```
(1) SELECT MANAGER
(2) FROM EMPLOYEES, DEPARTMENTS
(3) WHERE EMPLOYEES.NAME = 'Clark Kent'
(4)     AND EMPLOYEES.DEPT = DEPARTMENTS.DEPT;
```

Figure 1.1 Example SQL query.

In Figure 1.2 we see the same query written in the simplified version of the network-model query language DML that we discuss in Chapter 5. For a rough description of what these DML statements mean, lines (1) and (2) together tell the DBMS to find the record for Clark Kent in the EMPLOYEES file. Line (3) uses an implied "set" structure EMP-DEPT that connects employees to their departments, to find the department that "owns" the employee ("set" and "owns" are technical terms of DML's data model), i.e., the department

to which the employee belongs. Line (4) exploits the assumption that there is another set structure DEPT-MGR, relating departments to their managers. On line (5) we find and print the first manager listed for Clark Kent's department, and technically, we would have to search for additional managers for the same department, steps which we omit in Figure 1.2. Note that the print operation on line (5) is not part of the query language, but part of the surrounding "host language," which is an ordinary programming language.

The reader should notice that navigation among files is made far more explicit in DML than in SQL, so extra effort is required of the DML programmer. The difference is not just the extra line of code in Figure 1.2 compared with Figure 1.1; rather it is that Figure 1.2 states how we are to get from one record to the next, while Figure 1.1 says only how the answer relates to the data. This "declarativeness" of SQL and other languages based on the relational model is an important reason why systems based on that model are becoming progressively more popular. We shall have more to say about declarativeness in Section 1.4. □

```
(1) EMPLOYEES.NAME := "Clark Kent"
(2) FIND EMPLOYEES RECORD BY CALC-KEY
(3) FIND OWNER OF CURRENT EMP-DEPT SET
(4) FIND FIRST MANAGER RECORD IN CURRENT DEPT-MGR SET
(5) print MANAGER.NAME
```

Figure 1.2 Example query written in DML.

Transaction Management

Another important capability of a DBMS is the ability to manage simultaneously large numbers of *transactions*, which are procedures operating on the database. Some databases are so large that they can only be useful if they are operated upon simultaneously by many computers; often these computers are dispersed around the country or the world. The database systems used by banks, accessed almost instantaneously by hundreds or thousands of automated teller machines, as well as by an equal or greater number of employees in the bank branches, is typical of this sort of database. An airline reservation system is another good example.

Sometimes, two accesses do not interfere with each other. For example, any number of transactions can be reading your bank balance at the same time, without any inconsistency. But if you are in the bank depositing your salary check at the exact instant your spouse is extracting money from an automatic teller, the result of the two transactions occurring simultaneously

and without coordination is unpredictable. Thus, transactions that modify a data item must "lock out" other transactions trying to read or write that item at the same time. A DBMS must therefore provide some form of *concurrency control* to prevent uncoordinated access to the same data item by more than one transaction. Options and techniques for concurrency control are discussed in Chapter 9.

Even more complex problems occur when the database is distributed over many different computer systems, perhaps with duplication of data to allow both faster local access and to protect against the destruction of data if one computer crashes. Some of the techniques useful for distributed operation are covered in Chapter 10.

Security of Data

A DBMS must not only protect against loss of data when crashes occur, as we just mentioned, but it must prevent unauthorized access. For example, only users with a certain clearance should have access to the salary field of an employee file, and the DBMS must be able to associate with the various users their privileges to see files, fields within files, or other subsets of the data in the database. Thus, a DBMS must maintain a table telling for each user known to it, what access privileges the user has for each object. For example, one user may be allowed to read a file, but not to insert or delete data; another may not be allowed to see the file at all, while a third may be allowed to read or modify the file at will.

To provide an adequately rich set of constructs, so that users may see parts of files without seeing the whole thing, a DBMS often provides a *view* facility, that lets us create imaginary objects defined in a precise way from real objects, e.g., files or (equivalently) relations.

Example 1.4: Suppose we have an EMPLOYEES file with the following fields:

EMPLOYEES(NAME, DEPT, SALARY, ADDRESS)

and we wish most people to have access to the fields other than SALARY, but not to the SALARY field. In the language SQL, we could define a view SAFE-EMPS by:

```
CREATE VIEW SAFE-EMPS BY
SELECT NAME, DEPT, ADDRESS
FROM EMPLOYEES;
```

That is, view SAFE-EMPS consists of the NAME, DEPT, and ADDRESS fields of EMPLOYEES, but not the SALARY field. SAFE-EMPS may be thought of as a relation described by

SAFE-EMPS(NAME, DEPT, ADDRESS)