Second Edition

Principles of DATABASE SYSTEMS

JEFFREY D. ULLMAN
Stanford University

Second Edition

Principles of DATABASE SYSTEMS

JEFFREY D. ULLMAN
Stanford University

PITMAN

PITMAN PUBLISHING LIMITED
128 Long Acre, London WC2E 9AN

Associated Companies
Pitman Publishing New Zealand Ltd, Wellington
Pitman Publishing Pty Ltd, Melbourne

First published in Great Britain 1983
First published in USA 1982

This book was first published in 1982 by
Computer Science Press, Inc.
11 Taft Ct.
Rockville, Maryland 20850

1 2 3 4 5 6 87 86 85 84 83 82

Cover concept by Jeffrey D. Ullman

Cover artist—Ruth Kamminger

Library of Congress Cataloging in Publication Data

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

(Computer software engineering series)
Bibliography: p.
Includes index.
1. Data base management. I. Title. II. Title: Database systems. III. Series.
QA76.9.D3U44 1983 001.64 82-2510
US ISBN 0-914894-36-6 AACR2
UK ISBN 0-273-085948

PREFACE TO THE FIRST EDITION

It is evident that a course in database systems now plays a central role in the undergraduate and graduate programs in computer science. However, unlike the more traditional and better established systems areas, like compilers and operating systems, where a good mix of principles and practice was established many years ago, the subject matter in database systems has been largely descriptive.

This book is developed from notes I used in a course at Princeton that attempted to bring database systems into the mainstream of computer science. The course was taught to a mix of seniors and first-year graduate students. In it, I tried to relate database ideas to concepts from other areas, such as programming languages, algorithms, and data structures. A substantial amount of descriptive material was included, since students, being used to conventional programming languages, may find query languages rather unusual. The data structures relevant to databases are also somewhat different from the kinds of structures used in conventional programming, since the large scale of a database makes practical many structures that would be only of theoretical interest otherwise.

However, I added to the mix of topics the relevant theory that is now available. The principal concepts that have been found useful are concerned with relations and with concurrency. I have devoted a large portion of the book to a description of relations, their algebra and calculus, and to the query languages that have been designed using these concepts. Also included is an explanation of how the theory of relational databases can be used to design good systems, and a description of the optimization of queries in relation-based query languages. A chapter is also devoted to the recently developed protocols for guaranteeing consistency in databases that are operated on by many processes concurrently.

Exercises

Each chapter includes exercises to test basic concepts and, in some cases, to extend the ideas of the chapter. The most difficult exercises are marked with a double star, while problems of intermediate difficulty have a single star.

Acknowledgments

I am grateful for the comments and suggestions I received from Al Aho, Brenda Baker, Peter deJong, Ron Fagin, Vassos Hadzilacos, Zvi Kedem, Hank Korth, and Joseph Spinden. The initial draft of this manuscript was ably typed by Gerree Pecht. Her efforts and the support facilities at Princeton University are appreciated.

J. D. U.

PREFACE TO THE SECOND EDITION

The appearance of a large number of interesting and important developments in the database field prodded me to make a substantial revision of the book. The general direction of changes is towards more prominence for the relational model and systems based on that model. I have therefore simplified the material on the hierarchical and network models; however, I brought it up to the front of the book, so there will be some slight tendency for that material to be covered in a course of limited scope, as it should be. The other major changes to the book are the following.

1. The material on optimization has been expanded, by including a discussion of the System R approach to optimization, tableau-based methods, and optimization in the distributed environment.
2. There is a discussion of universal relation systems, which are relational database systems that support a user view that looks like a single relation.
3. Concurrency control by "optimistic," or timestamp-based methods has been introduced.
4. Distributed systems are covered, both for optimization issues and concurrency control.
5. A discussion of data structures for range queries appears.
6. There is an introduction to generalized dependencies and their inference.

Mapping the Old Edition to the New

Those familiar with the first edition will notice that some material has been moved around. A brief guide for finding the transplanted material follows.

Material from the old Chapters 1 and 2 is still there, along with some additional material. The old Chapter 3 has been split up. Introductory material from there now appears in Chapter 1, while some material on the network model (Section 3.2) has been combined with the old Chapter 7 to form the new Chapter 3. Section 3.3 on the hierarchical model has been combined with some of the material from the old Chapter 8 to form the new Chapter 4. Some of the IMS-specific material from Chapter 8 has been removed.

Chapter 4, on the relational model, has been divided into Chapters 5 and 6. Chapter 5 now contains relational algebra and calculus, and all the material on physical organization for relational systems. The query langauges themselves are described in Chapter 6. The old Section 3.4, comparing the

various models, now appears in Chapter 5. The old chapters 5, 6, 9, and 10 have been renumbered 7, 8, 10, and 11, and some new material added.

About the Cover

It's a "data baste," get it? No? Uh, well, the chef has this bulb baster, see, and $\cdots$, aw, forget it.

More Acknowledgements

The following people made comments on the first edition or made suggestions that were helpful in the preparation of the second edition: Dan Blosser, Martin Brooks, Mary Feay, Shel Finkelstein, Kevin Karplus, Arthur Keller, Keith Lantz, Dave Maier, Dan Newman, Mohammed Olumi, Shuky Sagiv, Charles Shub, and Joe Skudlarek. Their thoughts are much appreciated.

The second edition was prepared using Don Knuth's TEX typesetting system. I would also like to thank Luis Trabb-Pardo for invaluable assistance debugging my attempts to use the system.

J. D. U.

TABLE OF CONTENTS

1

INTRODUCTION TO DATABASE SYSTEM CONCEPTS

In this chapter we consider the principal functions of a database management system. We introduce basic concepts such as the different levels of abstraction present in such a system and the kinds of languages used to deal with the system. We then discuss a "real world" model against which to measure the capability of database systems to represent and manipulate realistic data. This model, called the "entity-relationship" model, is discussed in Section 1.3. Then, we briefly introduce three "data models," the relational, network, and hierarchical models. These are abstractions of the real world similar to the entity-relationship model, but they are more closely tuned to the needs of database system designers to deal with efficiency issues than the entity-relationship model is. Thus, it is the latter three models, rather than the entity-relationship model, that most frequently are used in database systems.

1.1 AN OVERVIEW OF A DATABASE SYSTEM

Let us consider an enterprise, such as an airline, that has a large amount of data kept for long periods of time in a computer. This data might include information about passengers, flights, aircraft, and personnel, for example. Typical relationships that might be represented include bookings (which passengers have seats on which flights?) flight crews (who is to be the pilot, copilot, etc., on which flights?), and service records (when and by whom was each aircraft last serviced?).

Data, such as the above, that is stored more-or-less permanently in a computer we term a *database*. The software that allows one or many persons to use and/or modify this data is a *database management system* (DBMS). A major role of the DBMS is to allow the user to deal with the data in abstract terms, rather than as the computer stores the data. In this sense, the DBMS acts as an interpreter for a high-level programming language, ideally allowing the user to specify what must be done, with little or no attention on the user's part to the detailed algorithms or data representation used by the system.

However, in the case of a DBMS, there may be far less relationship between the data as seen by the user and as stored in the computer, than between, say, arrays as defined in a typical programming language and the representation of those arrays in memory.

The database management system is one of the most complex varieties of software in existence. One way to get a feel for the different aspects of a DBMS is to consider the various kinds of users of such a system and the ways they interact with the system and with each other.

The Programmer/User and His Interaction with the System

Thinking again in terms of the airline database, the assistant operations manager has had some training as a programmer, and if he wants to find out what planes are being repaired and where they are located, he might formulate his query in a *query language* or *data manipulation language* (DML), and it might come out something like

```
PRINT Plane, Location
WHERE Status="broken"
```

Figure 1.1 shows what happens to such a query; it is represented by the query Q_1. First it is handled by a query processor, which is like a compiler for the query, although the output of this "compiler" is not machine language but rather a sequence of commands that are passed to other parts of the database management system. The query processor needs to know about the structure of the database, and we have shown it in Fig. 1.1 accessing information called the "database description." This information is needed so terms like "Plane" or "Status" can be interpreted in the context of the particular database system. Optimization of the query may also be attempted at this stage, since the speed with which the query can be answered may depend on the choices made by the query processor as to the sequence of steps to be taken by the system. We discuss optimization of queries in Chapter 8.

Many database systems are "ad hoc," in the sense that the software represented by the rectangles in Fig. 1.1 are written for the purpose of the database at hand only. In that case, the information about the database may be built into the query processor itself. Other database systems are built from commercial products; the various pieces of software shown in Fig. 1.1 are supplied by a vendor who has written them to deal with any, or almost any, database his customers may want. In that case, the tables referred to as a database description are essential, since the general-purpose query processor can have no built-in knowledge of the particular database. The description of the database is written in a specialized language called a *data definition language* or DDL, and is compiled into tables that are used by the rest of the DBMS.

The processed query is passed to a collection of routines that we shall term

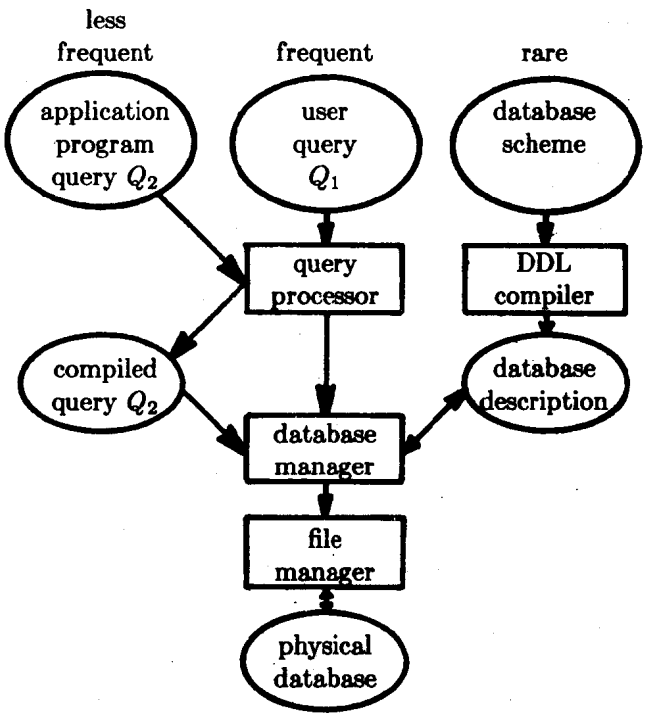

Fig. 1.1. Schematic diagram of a database system.

the *database manager*. One role of the database manager is to translate the query into terms that the file manager can understand, that is, into operations on files, rather than on the more abstract data structures of the database description. The *file manager* may be the general-purpose file system provided by the underlying operating system, or it may be a specialized file system that knows about the particular ways in which the data of the database is stored. The translation of the query into operations on files may be less than trivial, since the database could be represented by complex file structures such as are discussed in Chapter 2; the purpose of these structures is to make access and manipulation of the database as rapid as possible.

The database manager is frequently given several other tasks to perform, such as the following.

1. *Security.* Not every user should have access to all the data. For example, if personnel records are kept, only key personnel with the right and need to know salaries should be able to access this data. The user has presumably identified himself by password to the database manager, and it knows, perhaps from tables included with the database description, that this person is entitled to access data about salaries. We shall discuss the security

aspect of a DBMS in Chapter 10.

2. *Integrity.* Certain kinds of *consistency constraints,* (i.e., required properties of the data) can be checked by the DBMS, if it is told to do so. It is useful to have such checks made whenever a user gives a command in the data manipulation language to insert, delete, or change some data. Easiest to check are properties of values, such as the requirement that the number of passengers booked on a flight does not exceed the capacity of the aircraft. Somewhat harder to check are structural requirements involving equalities and inequalities of values, without reference to the values themselves (e.g., two aircraft may not be assigned to the same flight). Chapter 7 covers some aspects of structural integrity; Chapter 10 discusses integrity in general.
3. *Synchronization.* Often many users are running programs that access the database at the same time. The DBMS should provide protection against inconsistencies that result from two approximately simultaneous operations on a data item. For example, suppose that at about the same time, two reservation clerks issue requests to reserve a seat on flight 999. Each request results in the execution of a program that might examine the number of seats available (say one seat is left), subtracts one, and stores the resulting number of seats in the database. If the DBMS does not sequence these two transactions (the two invocations of the reservation program) properly, two passengers might wind up sitting in the same seat. We shall investigate measures for assuring proper synchronization in Chapter 11. Chapter 12 covers some of the more complex problems that occur when the database is not only accessed concurrently by several processes, but the processes and data may be distributed over several machines in widely separated locations.

The Naive User

While the assistant operations manager may have some programming ability, his boss, the operations manager may choose not to develop such a capability. He would rather type a single command like

RUN REPAIRS

and have the information printed out for him.

Similarly, the reservations clerk will sit at a terminal and type a command such as BOOK. The program invoked would engage in a dialog with the clerk, asking him for information in a fixed order, e.g. "enter name of passenger," and "enter desired flight number." Obtaining the requisite information from the clerk, the program interrogates the database to determine if space is available and, if so, modifies the database to reflect the reservation; if not, the program so informs the clerk.

The Applications Programmer

Programs such as REPAIRS or BOOK that are stored permanently and are available to users are called *applications programs*. Their creation is the responsibility of the *applications programmer*, a professional who writes and maintains programs written in the data manipulation language. The path taken by such a program is shown in Fig. 1.1 as the query Q_R. This program is written once, or perhaps a few times as needs change. It is compiled by the query processor and stored in the file system. The stored, compiled version can be invoked by commands, and it need not go through the compilation and optimization processes again (unless the database description has changed since the last time it was used).

The Database Administrator

We have shown in Fig. 1.1 a path labeled "rare" in which the database description itself is changed. That is, the data definition language program that describes the database is modified and recompiled into a new description that replaces the old. This operation is indeed an infrequent but important one, and a high-level person, generally called a *database administrator*, is granted responsibility for matters that deal with the database as a whole, while individual queries and manipulations of the database are handled by the applications programmers and users. Some of the responsibilities of the database administrator or his staff are the following.

1. The creation of the original description of the database structure and the way that structure is reflected by the files of the physical database.
2. The granting to the various users of authorization to access the database or parts of it.
3. Modification of the database description or its relationship to the physical organization of the database, should experience indicate that another organization would prove more efficient.
4. Making backup copies of the database and repairing damage to the database due to hardware or software failures or misuse.

1.2 BASIC DATABASE SYSTEM TERMINOLOGY

In this section we shall elaborate upon the concepts introduced in the previous section and develop them more precisely. There are several kinds of distinctions we must make when talking about a database system. We shall begin by discussing three levels of abstraction used in describing databases. We shall also emphasize the scheme/instance dichotomy, that is, the distinction between plans for a thing and the thing itself. Finally, we shall discuss the two different kinds of languages used in a database system, those for data definition and those for data manipulation.

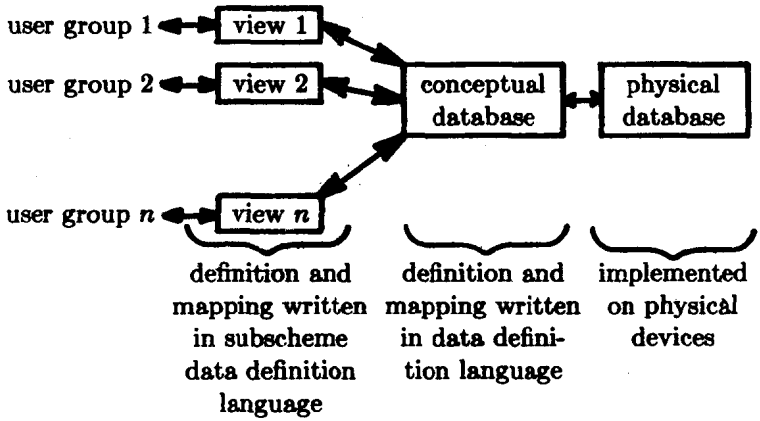

Fig. 1.2. Levels of abstraction in a database system.

Levels of Abstraction in a DBMS

It should be obvious that between the computer, dealing with bits, and the ultimate user dealing with abstractions such as flights or assignment of personnel to aircraft, there will be many levels of abstraction. A fairly standard viewpoint regarding levels of abstraction is shown in Fig. 1.2. There we see a single database, which may be one of many databases using the same DBMS software, at three different levels of abstraction. It should be emphasized that only the physical database actually exists.

The physical database resides permanently on secondary storage devices, such as disks and tapes. We may view the physical database itself at several levels of abstraction, ranging from that of records and files in a programming language such as Pascal, through the level of logical records, as supported by the operating system underlying the DBMS, down to the level of bits and physical addresses on storage devices. In this book we shall concentrate on the level of files and simple data structures. Chapter 2 will discuss the principal data structures used to implement a physical database, while Chapters 3, 4, and 5 point out the special data structures suitable for certain kinds of database systems.

The *conceptual database* is an abstraction of the real world pertinent to an enterprise. It is roughly at the level of passengers, flights, and so on, which we have discussed in connection with the enterprise of an airline. A DBMS provides a data definition language to specify the conceptual scheme and, most likely, some of the details regarding the implementation of the conceptual scheme by the physical scheme. The data definition language is a high-level language, enabling one to describe the conceptual database in terms of a "data model." An example of a suitable data model is the directed graph (the *network model*

in the jargon), where nodes represent sets of similar entities (e.g., all passengers, or all flights) and arcs represent associations (e.g., the assignment of aircraft to flights). Section 1.4 discusses the three major data models, network, relational, and hierarchical, in overview, while Chapters 3–5 discuss them in detail.

A *view* or *subscheme* is an abstract model of a portion of the conceptual database. Many, but not all, database management systems provide a facility for declaring views, called a *subscheme data definition language* and a facility for expressing queries and operations on the views, which would be called a *subscheme data manipulation language*.

As an example of the utility of views, an airline may provide a computerized reservation service, consisting of data and a collection of programs that deal with flights and passengers. These programs, and the people who use them, do not need to know about personnel files or the assignment of pilots to flights. The dispatcher may need to know about flights, aircraft, and aspects of the personnel files (e.g., which pilots are qualified to fly a 747), but does not need to know about personnel salaries or the passengers booked on a flight. Thus, there may be one view of the database for the reservations department and another for the dispatcher's office.

In a sense, a view is just a small conceptual database, and it is at the same level of abstraction as the conceptual database. However, there are senses in which a view can be "more abstract" than a conceptual database, as the data dealt with by a view may be constructable from the conceptual database but not actually present in that database.

For a canonical example, the personnel department may have a view that includes each employee's age. However, it is unlikely that ages would be found in the conceptual database, as ages would have to be changed each day for some of the employees. Rather, it is more likely that the conceptual database would include the employee's date of birth. When a user program, which believed it was dealing with a view that held age information, requested from the database a value for an employee's age, the DBMS would translate this request into "current date minus date of birth," which makes sense to the conceptual database, and the calculation would be performed on the corresponding data taken from the physical database.

Example 1.1: Let us illustrate the difference between physical, conceptual, and view levels of abstraction by an analogy from the programming languages world. In particular, we shall talk about arrays. On the conceptual level, we might use an ordinary declaration such as

integer array $A[1..n, 1..m]$

while on the physical level we might see the array A as stored in a block of consecutive storage locations, with $A[i, j]$ in location $a_0 + 4(m(i-1) + j - 1)$. A view of the array A might be formed by declaring a function $f(i)$ to be the