# Relational Databases and Knowledge Bases

Georges Gardarin

Patrick Valduriez

# Relational Databases and Knowledge Bases

Georges Gardarin
INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE (INRIA)
UNIVERSITY OF PARIS, VI (MASI)

Patrick Valduriez
MICROELECTRONIC AND COMPUTER TECHNOLOGY CORPORATION (MCC)



ADDISON-WESLEY PUBLISHING COMPANY
Reading, Massachusetts • Menlo Park, California • New York
Don Mills, Ontario • Wokingham, England • Amsterdam • Bonn
Sydney • Singapore • Tokyo • Madrid • San Juan

**Library of Congress Cataloging-in-Publication Data**

Gardarin, G. (Georges)
Relational databases and knowledge bases.

Bibliography: p.
Includes index.
1. Data base management. 2. Relational data bases.
I. Valduriez, Patrick. II. Title.
QA76.9.D3G375 1989 005.75'6 88-6295
ISBN 0-201-09955-1



CDEFGHIJ-DO-89

# PREFACE

A database management system (DBMS), or simply database system, is characterized by the data model it supports. The first DBMSs, designed in the 1960s, were based on hierarchical or network models and have been viewed as extensions of file systems in which interfile links are provided through pointers. The data manipulation languages of these systems are low level and require users to optimize the access to the data by carefully navigating in hierarchies or networks.

Since its inception in 1970, the relational data model has continued to gain wide acceptance among database researchers and practitioners. Unlike the hierarchical and network models, the relational model has a solid theoretical basis and provides simple and uniform representation for all data independent of any physical implementation. These advantages enabled the development of high-level data manipulation languages, freeing the user from data access optimization, and provided a solid basis for automating the database design process.

In the early 1980s, the first relational database systems appeared on the market, bringing definite advantages in terms of usability and end user productivity. Since then, their functional capabilities and performance have been significantly improved. Most relational DBMSs today support an integrated set of fourth-generation programming language tools to increase end user productivity. Most also provide extensive support for distributed database management. The development of fourth-generation tools and distributed database management capabilities has been facilitated by the relational model.

Current relational DBMSs are targeted to traditional data processing (busi-

ness) applications. More recently, important application domains, such as expert systems and computer-aided design, have shown the need to manage large amounts of data. Many believe the relational model will be used in future database systems that will support these new applications.

This book gives a comprehensive presentation of the relational model and the state-of-the-art principles and algorithms that guide the design of relational DBMSs. In addition, it shows how these systems can be extended to support new database application domains that typically require deductive capabilities and the support of complex objects. Such extensions are the basis of relational knowledge base systems (KBMSs) as developed in several research centers. Finally, the book introduces the use of technology to manage distributed databases. This book will enable readers to appreciate the practical superiority of the relational model for intensive database management. The application of the principles and algorithms presented in this book is illustrated in our recent book *Analysis and Comparison of Relational Database Systems,* (Addison-Wesley), which examines the best relational database systems and main relational database machines.

The current book is based on material used in undergraduate and graduate courses taught at the University of Paris 6 and other French schools. It follows a natural progression from introductory matter, such as file systems, to advanced topics, such as deductive databases. Knowledge of the basic principles of database systems will facilitate reading of this book but is not necessary. The book can be used by teachers to extend a traditional introduction to database technology and by professionals to extend their understanding of relational databases and knowledge bases. The main concepts are clearly exhibited by informal definitions.

The book contains twelve chapters. **Chapter 1** gives an introduction to first-order logic and operating systems, important topics since relational DBMSs and KBMSs are mainly implementations of logic-based languages on operating systems. Chapters 2 and 3 provide an introduction to database technology. **Chapter 2** deals with file management, more in the perspective of data access methods used by relational DBMSs. **Chapter 3** introduces the objectives and architectures of DBMSs, emphasizing standardization and relational DBMS architectures.

Chapters 4 and 5 present the relational data model and associated database design technology. **Chapter 4** gives a comprehensive presentation of the relational model including the model data structures and associated query languages based on relational algebra and relational calculus. In particular, the high-level, and now standard, Structured Query Language (SQL) is introduced. **Chapter 5** deals with database design in a relational framework. It demonstrates the value of the relational model and the associated normalization theory in helping the database designer's task.

Chapters 6 to 9 treat problems peculiar to the implementation of a relational DBMS. **Chapter 6** looks at semantic data control: views, security, and semantic integrity. Chapters 7 and 8 deal with transaction management in DBMSs in general. **Chapter 7** is devoted to concurrency control. **Chapter 8** is devoted to

reliability; it examines the solutions that maintain data consistency despite system and media failures. **Chapter 9** deals with query processing and query optimization.

Chapters 10 to 12 deal with the extension of relational database systems to support knowledge based applications and distributed environments. **Chapter 10** is devoted to the addition of deductive capabilities in relational database systems. These capabilities are of major importance for knowledge-based applications. **Chapter 11** deals with the incorporation of some key aspects of object orientation in relational database systems. These aspects are the support of potentially large objects of rich type and complex structures. These capabilities are required by applications such as office automation and computer-aided design. **Chapter 12** is an introduction to distributed databases whose management, although difficult, is simplified by the relational model.

We have been helped by many colleagues that we would like to thank: S. Abiteboul, B. Boettcher, H. Boral, C. Delobel, M. Franklin, G. Kiernan, R. Krishnamurthy, G. Lohman, R. Michel, C. Mohan, P. Neches, F. Pasquer, M. Scholl, E. Simon, M. Smith, Y. Viemont, and J. Zeleznikow. We are also grateful to the following reviewers for their comments and suggestions: Daniel Rosenkrantz, State University of New York at Albany and Dennis Shasha, Courant Institute of Mathematics. Furthermore, we would like to thank all our colleagues of the Advanced Computer Architecture program at MCC, Austin, and the SABRE project at INRIA, Paris, and the University of Paris, VI, for their support.

# CONTENTS

# 1 PRELIMINARIES

## 1.1 Introduction

Database management systems (DBMSs) are software products managing large sets of data stored in computers that can be queried and updated by end users. The design and implementation of DBMSs require practical and theoretical knowledge. In this chapter, operating system concepts and logic concepts relevant to the study of DBMSs are presented.

## 1.2 Disk Operating Systems

Operating systems are the basic data management tool offered by general-purpose modern computers. A relational database management system is built on top of a disk operating system (DOS). It stores structured data on a magnetic disk and retrieves structured data from a magnetic disk. The disk and the computer are managed by the operating system. Thus a DBMS is a layer of software components between the user and the DOS. It strongly interacts with the operating system. Clearly it is important to understand the functions and services offered by an operating system before studying database systems.

## 1.2.1 Computer Architecture

For a discussion on disk operating systems, it is useful to review computer architecture (see, for example [Baer80], for a detailed analysis). A simple computer architecture is shown in Fig. 1.1. At the heart of the computer is the *instruction* processor (CPU). It decodes and processes instructions that it obtains from main memory through the bus, the information highway on which all units talk and listen. Specialized sets of instructions can be executed by special-purpose processors in an enhanced computer architecture. Such a processor would be the input/output (I/O) processor, which, operating in parallel with other processors, transfers data from main memory to secondary peripheral devices (output) or from these devices to main memory (input).

A computer architecture like the one in Fig. 1.1 supports a basic software program that is the DOS. The DOS is a set of software layers built around a hardware machine to take charge of the user commands. These commands execute user programs submitted by multiple users in parallel. Thus the DOS supports the simultaneous execution of user programs that reside in main memory. One of the main functions of a DOS is to manage these simultaneous executions.

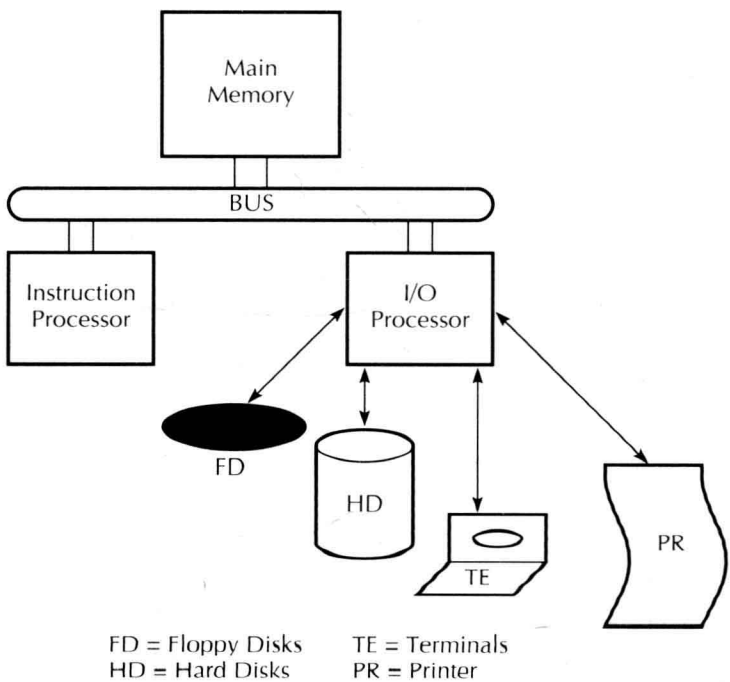


**FIGURE 1.1** A typical computer architecture

To run several user programs simultaneously, a DOS must share the computer resources between them. Resources are the CPU, main memory, external devices, and mass memory.

### 1.2.2 CPU Management

Resource sharing and simultaneous execution of user programs require the management of parallel processes. In general, a process is created by a master process using a special command. The process dedicated to receiving and analyzing user commands is responsible for creating first-level user processes. When a process has been created, it can execute a program by overlaying itself with the program machine code. It can also create new processes. Processes are deleted either by themselves or by other processes. In general, creating and deleting processes is the role of the *process manager.*

Processes must be synchronized as they compete for the machine resources. The main resource is the CPU. It is shared among the processes by time-sharing techniques. The system is organized around a *scheduler,* which allocates the CPU to the processes for a given period of time (perhaps a few milliseconds). Because certain resources cannot be simultaneously shared (such as I/O devices), a process may be waiting either for the completion of an external task (for example, an I/O on a disk) or for the exclusive use of a resource (for example, an I/O channel or a shared portion of main memory). In that case, it is in a dormant state, meaning that it is not candidate for the CPU. Each time the scheduler is invoked, it must select one process among the ready list of processes (those that are ready to use the CPU). Then it gives the CPU to the elected ready process, which becomes active. Up to the completion of either the task it wants to perform or its period of time, the active process code is executed by the CPU. The active process can also try to use a resource that is not free (because it is used in parallel by another process). The scheduler makes it wait and then reallocates the CPU to another ready process, if any, or waits for a ready process. A process waiting for a resource is awakened (put in ready state) by a *signal handler,* which receives and processes external events (interruptions).

In summary, the first layer of a DOS is the CPU manager, which may be divided in three modules (Fig. 1.2).

### 1.2.3 Memory Management

The next layer of a DOS is the *memory manager.* Memory management shares the main memory among the concurrent processes. A process may need memory space to store program code and data. It then demands allocation of memory

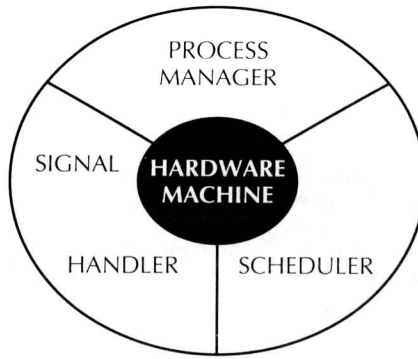


**FIGURE 1.2** CPU manager

pages (that is, blocks of continuous memory space, such as 4K bytes) to that part of the memory manager called the *memory allocator.* If pages are available, they are allocated to the process. When no page is available, a page replacement strategy is applied to bring out a page content and reallocate it to the demanding process. Such strategies may consist of writing on disk the content of the least recently used page (LRU) or the least frequently used one (LFU) or be more sophisticated. When a process no longer needs a memory page, it must release it, using a page release command of the memory allocator.

As pages are allocated and deallocated dynamically to processes, there is no reason for the memory space of a process to be continuous. Thus the memory manager must perform the mapping of continuous virtual addresses to the discontinuous physical space. This can be done by keeping pages allocated to processes continuous (one solution is to move process code and data in main memory when necessary) or by using a virtual memory management unit. Virtual memory management organizes pages in segments. A virtual address is composed of a segment number (S#), a page number in the segment (P#), and an offset in the page (D#). A hardware device called the *memory management unit* (MMU) maps virtual addresses <S#,P#,D#> to physical addresses. The role of the memory manager is to maintain the necessary tables for the MMU. This includes segment tables and page tables for each process. The memory manager must also ensure that processes correctly access to memory, particularly that a process does not read or write the memory allocated to another process (except if both processes agree to share memory). Such violations are generally detected by the machine hardware, which then invokes the memory manager.

In summary, the memory manager may be seen as composed of three parts, the first one allocating/deallocating pages (the page allocator), the second one controlling accesses to pages, and the third one swapping in/out pages from memory to disk (Fig. 1.3).

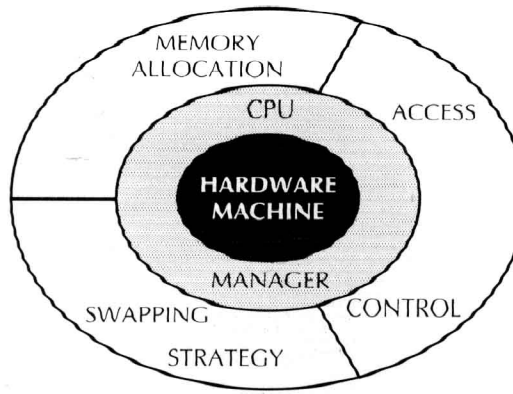


**FIGURE 1.3** Memory management functions in the DOS architecture

### 1.2.4 I/O Management

The DOS must also be able to communicate with the external world. This is the role of the I/O control layer. I/O management performs data transfer between the external devices (such as the magnetic disk and the user terminal) and a buffer pool maintained in main memory. A *buffer* is a data space in main memory composed of continuous pages in which data enter and go out at different speeds, generally in blocks of different sizes. A buffer pool is composed of several buffers, which are often shared between user processes and system I/O processes to avoid data moves.

Specific modules called *device handlers* read and write buffer contents (blocks of data) on devices. In general, there is one handler per class of device. A handler may manage a queue of I/O requests. When a data unit is to be read from a device by a process, the system checks to see if it is in the buffer pool. If it is, the system does not have to access the device, avoiding an I/O. If it is not, then a full block is read from the device in a buffer. When a data unit is to be written, it is put in a buffer, and control is often returned to the calling process before the actual I/O is performed on the device.

The effectiveness of the I/O manager is measured as a ratio between the number of I/O requests it saves (because data units are found or stored in buffers) and the number of requests it receives. To improve effectiveness, I/O managers use sophisticated strategies, such as read ahead and deferred write. Also the strategy for replacing buffers in the pool when none is free is an important parameter.

In summary, the I/O manager has two parts: the buffer pool manager and the I/O device handler (Fig. 1.4).

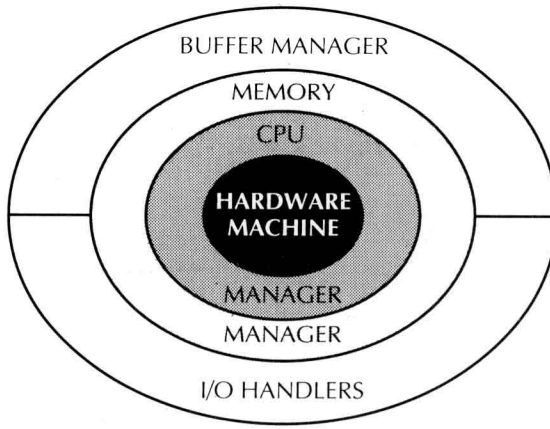


**FIGURE 1.4** I/O management in the DOS architecture

### 1.2.5 File and Basic Communication Management

With the next layer of the system, a user process can communicate with external memories, with other processes, and with terminals. Messages or records may be sent to or received from ideal external memories identified by a name (a file), other processes, or terminals. The exchange of messages requires that a communication path be opened between the sender and the receiver using a specific open command. Then reading and/or writing of messages can be done using the name of the logical path. At communication end, the communication path must be closed.

File management systems will be examined in Chapter 2. Communication management is a complex process; only a basic part is included in the DOS kernel presented here. This part corresponds to the basic functions of communication access methods, such as exchanging blocks of data between processes with possible synchronization of the processes and communicating with terminals. Fig. 1.5 depicts the system architecture as it appears now.

### 1.2.6 Program Management and Utilities

The interface between the end user and the system implies a *command language*. Thus the last layer of the DOS supplies a language to enter or leave the system, to control files, and to manage programs. Program management includes the necessary tools to execute a program. We briefly summarize the execution cycle of a program.

Application programs are translated from high-level source program code to executable machine instructions. Fig. 1.6 illustrates the steps performed to

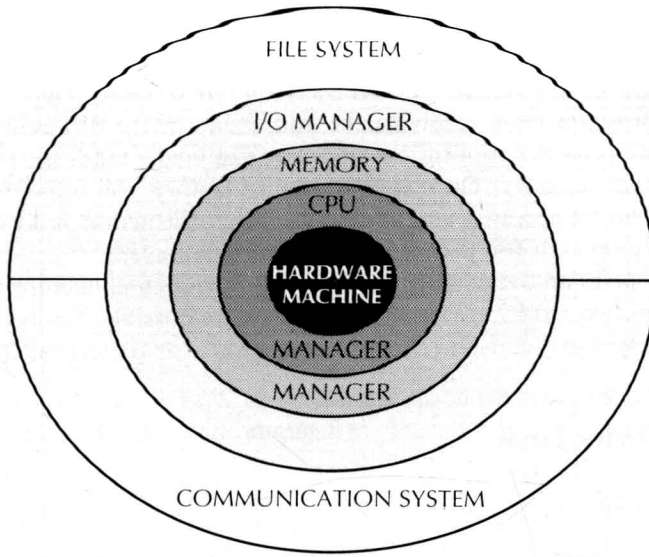


**FIGURE 1.5** File and basic communication in the DOS architecture

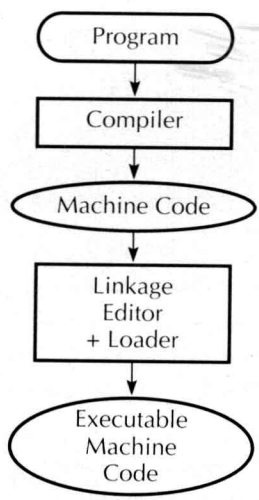


**FIGURE 1.6** Source program translation