

THE SYSTEMS PROGRAMMING SERIES

AN INTRODUCTION TO
**DATABASE
SYSTEMS**

C. J. DATE

S I X T H E D I T I O N

S	<table><tr><th>S#</th><th>SNAME</th><th>STATUS</th><th>CITY</th></tr><tr><td>S1</td><td>Smith</td><td>20</td><td>London</td></tr><tr><td>S2</td><td>Jones</td><td>10</td><td>Paris</td></tr><tr><td>S3</td><td>Blake</td><td>30</td><td>Paris</td></tr><tr><td>S4</td><td>Clark</td><td>20</td><td>London</td></tr><tr><td>S5</td><td>Adams</td><td>30</td><td>Athens</td></tr></table>	S#	SNAME	STATUS	CITY	S1	Smith	20	London	S2	Jones	10	Paris	S3	Blake	30	Paris	S4	Clark	20	London	S5	Adams	30	Athens	SP	<table><tr><th>S#</th><th>P#</th><th>QTY</th></tr><tr><td>S1</td><td>P1</td><td>300</td></tr><tr><td>S1</td><td>P2</td><td>200</td></tr><tr><td>S1</td><td>P3</td><td>400</td></tr><tr><td>S1</td><td>P4</td><td>200</td></tr><tr><td>S1</td><td>P5</td><td>100</td></tr><tr><td>S1</td><td>P6</td><td>100</td></tr><tr><td>S2</td><td>P1</td><td>300</td></tr><tr><td>S2</td><td>P2</td><td>400</td></tr><tr><td>S3</td><td>P2</td><td>200</td></tr><tr><td>S4</td><td>P2</td><td>200</td></tr><tr><td>S4</td><td>P4</td><td>300</td></tr><tr><td>S4</td><td>P5</td><td>400</td></tr></table>	S#	P#	QTY	S1	P1	300	S1	P2	200	S1	P3	400	S1	P4	200	S1	P5	100	S1	P6	100	S2	P1	300	S2	P2	400	S3	P2	200	S4	P2	200	S4	P4	300	S4	P5	400
S#	SNAME	STATUS	CITY																																																															
S1	Smith	20	London																																																															
S2	Jones	10	Paris																																																															
S3	Blake	30	Paris																																																															
S4	Clark	20	London																																																															
S5	Adams	30	Athens																																																															
S#	P#	QTY																																																																
S1	P1	300																																																																
S1	P2	200																																																																
S1	P3	400																																																																
S1	P4	200																																																																
S1	P5	100																																																																
S1	P6	100																																																																
S2	P1	300																																																																
S2	P2	400																																																																
S3	P2	200																																																																
S4	P2	200																																																																
S4	P4	300																																																																
S4	P5	400																																																																
P	<table><tr><th>P#</th><th>PNAME</th><th>COLOR</th><th>WEIGHT</th><th>CITY</th></tr><tr><td>P1</td><td>Nut</td><td>Red</td><td>12</td><td>London</td></tr><tr><td>P2</td><td>Bolt</td><td>Green</td><td>17</td><td>Paris</td></tr><tr><td>P3</td><td>Screw</td><td>Blue</td><td>17</td><td>Rome</td></tr><tr><td>P4</td><td>Screw</td><td>Red</td><td>14</td><td>London</td></tr><tr><td>P5</td><td>Cam</td><td>Blue</td><td>12</td><td>Paris</td></tr><tr><td>P6</td><td>Cog</td><td>Red</td><td>19</td><td>London</td></tr></table>	P#	PNAME	COLOR	WEIGHT	CITY	P1	Nut	Red	12	London	P2	Bolt	Green	17	Paris	P3	Screw	Blue	17	Rome	P4	Screw	Red	14	London	P5	Cam	Blue	12	Paris	P6	Cog	Red	19	London																														
P#	PNAME	COLOR	WEIGHT	CITY																																																														
P1	Nut	Red	12	London																																																														
P2	Bolt	Green	17	Paris																																																														
P3	Screw	Blue	17	Rome																																																														
P4	Screw	Red	14	London																																																														
P5	Cam	Blue	12	Paris																																																														
P6	Cog	Red	19	London																																																														

The suppliers-and-parts database (sample values)

S

S#	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

SPJ

S#	P#	J#	QTY
S1	P1	J1	200
S1	P1	J4	700
S2	P3	J1	400
S2	P3	J2	200
S2	P3	J3	200
S2	P3	J4	500
S2	P3	J5	600
S2	P3	J6	400
S2	P3	J7	800
S2	P5	J2	100
S3	P3	J1	200
S3	P4	J2	500
S4	P6	J3	300
S4	P6	J7	300
S5	P2	J2	200
S5	P2	J4	100
S5	P5	J5	500
S5	P5	J7	100
S5	P6	J2	200
S5	P1	J4	100
S5	P3	J4	200
S5	P4	J4	800
S5	P5	J4	400
S5	P6	J4	500

P

P#	PNAME	COLOR	WEIGHT	CITY
P1	Nut	Red	12	London
P2	Bolt	Green	17	Paris
P3	Screw	Blue	17	Rome
P4	Screw	Red	14	London
P5	Cam	Blue	12	Paris
P6	Cog	Red	19	London

J

J#	JNAME	CITY
J1	Sorter	Paris
J2	Display	Rome
J3	OCR	Athens
J4	Console	Athens
J5	RAID	London
J6	EDS	Oslo
J7	Tape	London

The suppliers-parts-projects database (sample values)

C. J. Date

AN INTRODUCTION TO

Database Systems

SIXTH
EDITION



Addison-Wesley Publishing Company

Reading, Massachusetts • Menlo Park, California • New York
Don Mills, Ontario • Wokingham, England • Amsterdam • Bonn
Sydney • Singapore • Tokyo • Madrid • San Juan • Milan • Paris

This book is in the
Addison-Wesley Systems Programming Series

Consulting Editors: IBM Editorial Board

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and Addison-Wesley was aware of a trademark claim, the designations have been printed in initial caps or all caps.

The programs and applications presented in this book have been included for their instructional value. They have been tested with care, but are not guaranteed for any particular purpose. The publisher does not offer any warranties or representations, nor does it accept any liabilities with respect to the programs or applications.

Library of Congress Cataloging-in-Publication Data

Date, C. J.

An introduction to database systems / C. J. Date. — 6th ed.

p. cm. — (The Systems programming series)

Includes bibliographical references and index.

ISBN 0-201-54329-X

I. Database management. I. Title. II. Series: Addison-Wesley systems programming series.

QA76.9.D3D3659 1995

005.74—dc20

94-3187

CIP

Copyright © 1995 by Addison-Wesley Publishing Company, Inc.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher. Printed in the United States of America.

ISBN 0-201-54329-X

1 2 3 4 5 6 7 8 9 10-DOC-98 97 96 95

*This book is dedicated to my wife Lindy
and to the memory of my mother Rene*

The Systems Programming Series

Communications Architecture for Distributed Systems

R. J. Cypser

An Introduction to Database Systems, 6th Edition

C. J. Date

An Introduction to Database Systems, Volume II

C. J. Date

Computer Graphics: Principles and Practice, 2nd Edition

James Foley

Andries van Dam

Steven Feiner

John Hughes

Structured Programming: Theory and Practice

Richard C. Linger

Harland D. Mills

Bernard I. Witt

IBM Editorial Board

Chairman: Gene F. Hoffnagle

Foreword

The field of systems programming primarily grew out of the efforts of many programmers and managers whose creative energy went into producing practical, utilitarian systems programs needed by the rapidly growing computer industry. Programming was practiced as an art where each programmer invented his own solutions to problems, with little guidance beyond that provided by his immediate associates. In 1968, the late Ascher Opler, then at IBM, recognized that it was necessary to bring programming knowledge together in a form that would be accessible to all systems programmers. Surveying the state of the art, he decided that enough useful material existed to justify a significant codification effort. On his recommendation, IBM decided to sponsor The Systems Programming Series as a long term project to collect, organize, and publish those principles and techniques that would have lasting value throughout the industry. Since 1968 nineteen titles have been published in the Series including *An Introduction to Database Systems, Sixth Edition*, the most thoroughly revised of all previous editions. Like its predecessors, the book provides the basis for a solid education in the fundamentals of database technology and reflects the latest developments in the field.

The Series consists of an open-ended collection of text-reference books. The contents of each book represent the individual author's view of the subject area and do not necessarily reflect the views of the IBM Corporation. Each is organized for course use but is detailed enough for reference.

Representative topic areas already published, or that are contemplated to be covered by the Series, include: database systems, communication systems, graphics systems, expert systems, and programming process management. Other topic areas will be included as the systems programming discipline evolves and develops.

The Editorial Board

Petitioner Uber Ex-1051, 0010

About the Author

C. J. Date is an independent author, lecturer, and consultant, specializing in relational database systems. He is based in Healdsburg, California.

In 1967, following several years spent working as a mathematical programmer and programming instructor for Leo Computers Ltd. (London, England), Mr. Date moved to the IBM (UK) Development Laboratories, where he worked on the integration of database functions into PL/I. In 1974 he moved to the IBM Systems Development Center in California, where he was responsible for the design of a database language known as the Unified Database Language, UDL. He was subsequently involved in technical planning and externals design for the IBM relational database products SQL/DS (announced in 1981 for VSE and in 1983 for VM) and DB2 (announced in 1983 for MVS). He left IBM in May 1983.

Mr. Date has been active in the database field for some 25 years. He was one of the first people anywhere to recognize the significance of Codd's pioneering work on the relational model. He has lectured widely on technical subjects—principally on database topics, and especially on relational database—throughout North America and also in Europe, Australia, Latin America, and the Far East. In addition to the present book, he is author or coauthor of several other database books, including *An Introduction to Database Systems: Volume II* (1983), which covers advanced aspects of the subject; *Database: A Primer* (1983), which treats database systems from the nonspecialist's point of view; *Relational Database: Selected Writings* (1986), *Relational Database Writings 1985–1989* (1990), and *Relational Database Writings 1989–1991* (1992), which treat various aspects of relational technology in depth; and a series of books on specific systems and languages—*A Guide to DB2* (4th edition, 1993), *A Guide to SYBASE and SQL Server* (1992), *A Guide to SQL/DS* (1988), *A Guide to INGRES* (1987), and *A Guide to the SQL Standard* (3rd edition, 1993). His books have been translated into many languages, including Chinese, Dutch, French, Italian, Japanese, Polish, Portuguese, Russian, Spanish, and Braille.

Mr. Date has also published well over 100 technical papers and articles and has made a variety of contributions to database theory. He is a regular columnist for *Database Programming & Design* magazine. His professional seminars on database technology (offered both in North America and overseas) are widely considered to be second to none for the quality of their material content and the clarity of the exposition.

Vertrag über die Abgrenzung

Der Vertrag über die Abgrenzung zwischen der Schweiz und Frankreich wurde am 1. Januar 1816 in Paris unterzeichnet. Er regelt die Grenzen zwischen den beiden Ländern und ist ein wichtiger Bestandteil des internationalen Rechts. Der Vertrag ist in drei Teilen gegliedert: der erste Teil enthält die allgemeine Bestimmungen, der zweite Teil die speziellen Bestimmungen und der dritte Teil die Schlussbestimmungen. Der Vertrag ist in drei Sprachen abgefasst: Französisch, Deutsch und Englisch. Die deutsche Fassung ist die offizielle Fassung. Der Vertrag ist in drei Exemplaren abgefasst, die jeweils einem der Unterzeichner zugehen. Die Unterzeichner sind der Schweizer Regierung und der französischen Regierung. Der Vertrag ist in drei Sprachen abgefasst: Französisch, Deutsch und Englisch. Die deutsche Fassung ist die offizielle Fassung. Der Vertrag ist in drei Exemplaren abgefasst, die jeweils einem der Unterzeichner zugehen. Die Unterzeichner sind der Schweizer Regierung und der französischen Regierung.

Preface to the Sixth Edition

The field of database technology is suffering from an information explosion—ironically enough, since information processing is what database technology is supposed to be all about. Here, for example, is a partial list of professional publications in the field that appear in the United States on a regular basis:

- *ACM Transactions on Database Systems (TODS)*, published quarterly—around 700 pages per year
- *ACM SIGMOD Record*, published quarterly—around 250 pages per year
- *Proceedings of the Annual ACM SIGMOD International Conference on Management of Data*—around 450 pages per year
- *Proceedings of the Annual ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*—around 350 pages per year
- *Proceedings of the Annual International Conference on Very Large Data Bases (VLDB)*—around 650 pages per year
- *The VLDB Journal*, published quarterly—around 450 pages per year

Add to the foregoing the various more specialized conferences on distributed database systems, or CAD/CAM databases, or expert database systems, or client/server systems, or object-oriented systems (etc.)—say eight or ten conferences a year, with proceedings typically running at around 300–350 pages . . . add too the huge number of technical reports from universities and industrial research laboratories . . . add the occasional papers that appear in the publications of related disciplines, such as office automation, artificial intelligence, and programming languages . . . add the trade journals such as *Data Base Newsletter*, *Database Review*, *InfoDB*, *Database Programming & Design*, etc., etc., which together represent many thousands of pages per year . . . add the trade shows such as *Database World* and *DB/Expo*, each with its own voluminous set of proceedings . . . add the vendor reference manuals and other documents describing various commercial products, each with a new release every 18 months or so . . . add the numerous textbooks now available that have the word “database” somewhere in their title . . . and it becomes apparent that there are (conservatively) somewhere in

excess of 100,000 pages of new material published *every year*. It is thus clearly impossible to keep abreast of everything that is happening in the database field.

From all of the above, it follows that the present book—despite its inordinate length, for which I apologize—can only be, as its title indicates, an introduction to the subject, not an exhaustive treatment. Because of the explosive growth of the field, moreover, even what it is that constitutes “a good introduction” to the subject changes fairly rapidly; as a result, this new (sixth) edition of the book is once again so different from its predecessor that it is to all intents and purposes a brand new book. The overall objective remains the same, however—namely, to provide the basis for a solid education in the fundamentals of database technology, and in particular to pave the way for an understanding of the directions in which the field is currently developing and is likely to develop in the future. The book is intended primarily as a textbook, not a work of reference, though to some extent I am hopeful that it can serve the latter purpose also. The emphasis is on insight, not algorithms.

Prerequisites: Readers are assumed to be professionally interested in some aspect of data processing; they may, for example, be systems analysts or designers, application programmers, systems programmers, students following a college or similar course in computer science, or teachers of such a course. They are expected:

- To have a reasonable appreciation of the capabilities of a modern computer system, especially the file-management features of such a system (though the book does include an appendix that provides a tutorial overview of this latter topic);
- To have some familiarity with at least one high-level programming language.

Since these prerequisites are not particularly demanding, however, I am hopeful that the book will prove suitable as an introductory text for anyone concerned with using or implementing a database system, or for anyone who simply wishes to broaden a general knowledge of the computer science field.

The book is divided into six major parts, as follows:

- I. Basic Concepts
- II. The Relational Model
- III. Database Design
- IV. Data Protection
- V. Further Topics
- VI. Object-Oriented Systems

Each part in turn is divided into several chapters:

- Part I (three chapters) provides a broad introduction to the concepts of database systems in general and relational database systems in particular.
- Part II (five chapters), which is the longest part of the book—indeed, it is almost a

book within a book—consists of a detailed and very careful examination of the **relational model**, which is not only the theoretical foundation underlying today's relational products, but is in fact the theoretical foundation for the entire database field.* A lengthy chapter on the major concepts of the standard relational language SQL is also included.

- Part III (four chapters) discusses the general question of **database design**; three chapters are devoted to design theory and one to the **entity/relationship model**.
- Part IV (four chapters) is concerned with various aspects of the **data protection** problem. Specifically, it discusses the concepts of recovery, concurrency, security, and integrity in a database environment.
- Part V (five chapters) shows how the ideas of the relational model are relevant to a variety of further aspects of database technology—optimization, distributed database processing, view support, and so on.
- Finally, Part VI (four chapters) describes the important new **object-oriented** approach to database systems. Chapter 25 in particular (the last in the book) considers the possibility of a rapprochement between object-oriented and relational technology.

In addition, there are four appendixes—one on file management and physical storage structures (as already mentioned), one on a commercial relational system (IBM's DB2 product), one on the comparatively new field of logic-based systems, and one that lists some important abbreviations and acronyms.

The book overall is meant to be read in sequence more or less as written. Each chapter opens with an introduction and closes with a summary; in addition, most chapters include a set of exercises, usually with answers (often the answers give additional information about the subject of the exercise). Most chapters also include an extensive list of references, many of them annotated. This structure allows the subject matter to be treated in a multilevel fashion, with the most important concepts and results being presented "in line" in the main body of the text and various subsidiary issues and more complex aspects being deferred to the Exercises, Answers, or References sections, as appropriate.

Note: References are identified in the text by two-part numbers in square brackets. For example, the reference "[4.1]" refers to the first item in the list of references at the end of Chapter 4, namely, a paper by E. F. Codd published in *Communications of the ACM* (CACM), Vol. 13, No. 6 (June 1970).

For readers who might be familiar with the previous edition of the book, the major differences from that edition are summarized below.

- *Part I:* Chapters 1–2 cover roughly the same ground as Chapters 1–2 in the previous edition, but they have been rewritten, and the treatment of certain topics (e.g.,

* By happy coincidence, this book's planned publication date coincides almost exactly with the relational model's silver anniversary (Codd's first publication on the subject [4.2] was dated August 19th, 1969).

client/server processing) has been improved and amplified. Chapter 3 is new; it provides a gentle introduction to some of the material to be covered in more detail in later parts of the book (especially Part II).

- *Part II:* Chapter 4–7 (on the relational model) represent a completely rewritten, considerably expanded, and very much improved version of Chapters 11–15 from the previous edition. Chapter 8 (on the SQL language) is a condensed and considerably revised version of material from Chapters 5–6 and 9 (and elsewhere) from that edition.

Note: Some words of explanation are in order here. Previous editions of this book discussed commercial relational systems, and in particular the commercial SQL language, before getting into details of the theoretical relational model; this was done in the belief that it is usually easier on the student to show the concrete before the abstract. Unfortunately, however, the gulf between SQL and the relational model has grown so wide that I now feel it would be actively misleading to treat SQL first. In fact, SQL in its present form is so far from being a true embodiment of relational principles—it suffers from so many sins of both omission and commission—that I would have preferred to relegate it to an appendix in this edition; but the language is so important from a commercial point of view (and every database professional needs to have some familiarity with it) that it would simply not be appropriate to treat it in so dismissive a manner. I have therefore settled on a compromise—a long chapter on SQL basics in this part of the book, and individual sections in subsequent chapters describing those aspects of SQL that are specific to the subject of the chapter in question (where applicable).

Incidentally, all SQL discussions in this edition are at the level of the current SQL standard known informally as “SQL/92.”

- *Part III:* Chapter 9 (on functional dependencies) is almost entirely new. Chapters 10 and 11 represent a major revision and expansion of the old Chapter 21. Chapter 12 is a rewrite of the old Chapter 22 (the material on the entity/relationship model has been amplified and the material on RM/T has been dropped).

Again, some explanation is in order. Many reviewers complained that the previous edition treated database design issues far too late. It is my own feeling that students are simply not ready to design databases properly or to appreciate design issues until they have some understanding of what databases are and how they are used; in other words, I believe it is important to spend some time on the relational model and related matters before exposing the student to database design questions. However, I do agree that it is possible to treat design matters earlier than the previous edition did, and I have accordingly done so in this edition.

- *Part IV:* The four chapters of this part represent a major revision and expansion of Chapters 16 and 17 from the previous edition.
- *Part V:* Chapters 18 (on optimization) and 21 (on distributed database) are expanded and significantly revised versions of the old Chapters 18 and 23, respectively. Chapter 17 (on views) is almost completely new (it replaces the old Chapter

8); in particular, it includes some important new results regarding view updatability. Chapters 19 (on domains) and 20 (on missing information) are both new.

- *Part VI*: This part is almost completely new, though certain portions of the material did appear in Chapter 25 in the previous edition.

Finally, Appendixes A, B, and C replace the old Chapters 3, 4, and 24, respectively, and Appendix D is an updated version of the old Appendix D.

In addition to the changes sketched above, the following topics have been (mostly) dropped from this edition:

- INGRES (old Chapter 10)
- Frontend subsystems (old Chapter 19)
- Administration facilities (old Chapter 20)
- RM/T (part of the old Chapter 22)
- Further research topics (old Chapter 26)
- Prerelational systems (old Appendixes A, B, and C)

Several readers of the previous edition felt that these topics were somewhat peripheral to the book's main theme; in an attempt to keep the book down to a comparatively manageable size, therefore (and also to keep it more focused), it seemed reasonable to drop them from this new edition.

One further introductory remark: The previous edition included the qualifier *Volume I* in its title. That was because at that time there was a sequel, *An Introduction to Database Systems: Volume II* (first edition, Addison-Wesley, 1983). Volume II is still in print and is still reasonably current; over the years, however, much of the material of Volume II has found its way into Volume I as well, with the result that the "Volume I vs. Volume II" split no longer makes much sense. I have therefore decided to drop the "Volume I" qualifier from the title of this edition.

I would like to close these prefatory notes with the following edited extract from another preface—Bertrand Russell's own preface to *The Bertrand Russell Dictionary of Mind, Matter and Morals* (ed., Lester E. Denonn), Citadel Press, 1993, reprinted here by permission:

I have been accused of a habit of changing my opinions. . . . I am not myself in any degree ashamed of [that habit]. What physicist who was already active in 1900 would dream of boasting that his opinions had not changed during the last half century? . . . The kind of philosophy that I value and have endeavoured to pursue is scientific, in the sense that there is some definite knowledge to be obtained and that new discoveries can make the admission of former error inevitable to any candid mind. For what I have said, whether early or late, I do not claim the kind of truth which theologians claim for their creeds. I claim only, at best, that the opinion expressed was a sensible one to hold at the time. . . . I should be much surprised if subsequent research did not show that it needed to be modified. [Such opinions

were not} intended as pontifical pronouncements, but only as the best I could do at the time towards the promotion of clear and accurate thinking. Clarity, above all, has been my aim.

Readers of earlier editions of this book, if they study the present edition, will find that I too have changed my opinions on many matters (and will no doubt continue to do so). I hope they will accept the remarks quoted above as adequate justification for this state of affairs. I share Bertrand Russell's perception of what the field of scientific inquiry is all about, but he expresses that perception far more eloquently than I could hope to do.

For information about the on-line Instructor's Manual which complements this book, mail may be sent to date@aw.com. To obtain an information file, type `information` on the subject line and `send information` in the body of the text.

Acknowledgments

Once again it is a pleasure to acknowledge my debt to the many people involved, directly or indirectly, in the production of this book. First, the text has certainly benefited from the comments of students on the seminars I have been teaching over the past several years. It has also benefited enormously from the comments of, and discussions with, numerous friends and reviewers, including Nagraj Alur, Marilyn Bohl, Charley Bontempo, Daniel Cooke, Hugh Darwen (alias Andrew Warden), David Embley, Henry Etlinger, Bill Grosky, Tim Hartley, Sebastian Holst, Tom Johnston, Won Kim, Roger King, Adrian Lerner, Tim Martyn, Nelson Mattos, David McGoveran, Jim Panttaja, Fabian Pascal, Arnie Rosenthal, Sharon Salveter, Martin Solomon, Alan Tharp, Colin White, Paul Winsberg, and Salih Yurttas. Each of these people reviewed at least some portion of the manuscript of this edition or made technical material available or otherwise helped me find answers to my many technical questions, and I am very grateful to all of them. I would particularly like to thank my friends Hugh Darwen and David McGoveran for numerous illuminating conversations and much stimulating correspondence. I would also like to thank my wife Lindy for contributing the cover art once again. Finally, I am grateful (as always) to everyone at Addison-Wesley—especially Lynne Doran Cote and Katherine Harutunian—for all of their encouragement and support throughout this project, and to my editor Elydia Davis for her usual sterling job.

Healdsburg, California

C. J. Date

Contents

Part I

BASIC CONCEPTS

Chapter 1

An Overview of Database Management 2

- 1.1 An Introductory Example 2
- 1.2 What Is a Database System? 4
- 1.3 What Is a Database? 9
- 1.4 Why Database? 13
- 1.5 Data Independence 16
- 1.6 Relational Systems and Others 21
- 1.7 Summary 24
- Exercises 25
- Answers to Selected Exercises 26

Chapter 2

An Architecture for a Database System 28

- 2.1 Purpose 28
- 2.2 The Three Levels of the Architecture 28
- 2.3 The External Level 31
- 2.4 The Conceptual Level 34
- 2.5 The Internal Level 35
- 2.6 Mappings 36
- 2.7 The Database Administrator 37
- 2.8 The Database Management System 39
- 2.9 The Data Communications Manager 41
- 2.10 Client/Server Architecture 42
- 2.11 Utilities 44
- 2.12 Distributed Processing 45
- 2.13 Summary 48
- Exercises 49
- References and Bibliography 50

Chapter 3

An Introduction to Relational Databases 52

- 3.1 Introduction 52
- 3.2 Relational Systems 52

3.3	A Note on Terminology	56
3.4	The Relational Model	57
3.5	Optimization	58
3.6	The Catalog	60
3.7	Base Tables and Views	62
3.8	The SQL Language	65
3.9	The Suppliers-and-Parts Database	68
3.10	Summary	70
	Exercises	73
	References and Bibliography	73
	Answers to Selected Exercises	74

Part II

THE RELATIONAL MODEL

Chapter 4 Relational Data Objects: Domains and Relations 79

4.1	An Introductory Example	79
4.2	Domains	81
4.3	Relations	86
4.4	Kinds of Relations	95
4.5	Relations and Predicates	97
4.6	Relational Databases	98
4.7	Summary	99
	Exercises	100
	References and Bibliography	100
	Answers to Selected Exercises	106

Chapter 5 Relational Data Integrity: Candidate Keys and Related Matters 110

5.1	Introduction	110
5.2	Candidate Keys	112
5.3	Primary Keys and Alternate Keys	115
5.4	Foreign Keys	116
5.5	Foreign Key Rules	120
5.6	Nulls (A Digression)	123
5.7	Candidate Keys and Nulls	124
5.8	Foreign Keys and Nulls	126
5.9	Summary	128
	Exercises	130
	References and Bibliography	131
	Answers to Selected Exercises	134

Chapter 6 Relational Operators I: Relational Algebra 139

6.1	Introduction	139
6.2	Closure	141
6.3	A Syntax for the Relational Algebra	143
6.4	Traditional Set Operations	145
6.5	Special Relational Operations	149

6.6	Examples	156
6.7	What Is the Algebra For?	158
6.8	EXTEND and SUMMARIZE	160
6.9	Update Operations	166
6.10	Relational Comparisons	169
6.11	Summary	170
	Exercises	171
	References*and Bibliography	173
	Answers to Selected Exercises	178
Chapter 7	Relational Operators II: Relational Calculus	185
7.1	Introduction	185
7.2	Tuple-Oriented Relational Calculus	187
7.3	Examples	194
7.4	Relational Calculus vs. Relational Algebra	196
7.5	Computational Capabilities	201
7.6	Domain-Oriented Relational Calculus	203
7.7	Summary	206
	Exercises	207
	References and Bibliography	208
	Answers to Selected Exercises	211
Chapter 8	The SQL Language	219
8.1	Introduction	219
8.2	Data Definition	220
8.3	Data Manipulation: Retrieval Operations	226
8.4	Data Manipulation: Update Operations	234
8.5	Table Expressions	235
8.6	Conditional Expressions	241
8.7	Scalar Expressions	243
8.8	Embedded SQL	244
8.9	Summary	252
	Exercises	253
	References and Bibliography	255
	Answers to Selected Exercises	257
Part III	DATABASE DESIGN	
Chapter 9	Functional Dependencies	271
9.1	Introduction	271
9.2	Basic Definitions	272
9.3	Trivial and Nontrivial Dependencies	274
9.4	Closure of a Set of Dependencies	275
9.5	Closure of a Set of Attributes	276
9.6	Irreducible Sets of Dependencies	278
9.7	Summary	280

	Exercises	281
	References and Bibliography	282
	Answers to Selected Exercises	284
Chapter 10	Further Normalization I: 1NF, 2NF, 3NF, BCNF	288
10.1	Introduction	288
10.2	Nonloss Decomposition and Functional Dependencies	292
10.3	First, Second, and Third Normal Forms	295
10.4	Dependency Preservation	303
10.5	Boyce/Codd Normal Form	306
10.6	Summary	312
	Exercises	313
	References and Bibliography	315
	Answers to Selected Exercises	316
Chapter 11	Further Normalization II: Higher Normal Forms	325
11.1	Introduction	325
11.2	Multivalued Dependencies and Fourth Normal Form	325
11.3	Join Dependencies and Fifth Normal Form	330
11.4	The Normalization Procedure Summarized	334
11.5	Other Normal Forms	337
11.6	Summary	338
	Exercises	339
	References and Bibliography	339
	Answers to Selected Exercises	344
Chapter 12	The Entity/Relationship Model	347
12.1	Introduction	347
12.2	The Overall Approach	349
12.3	An Overview of the E/R Model	351
12.4	E/R Diagrams	355
12.5	Database Design with the E/R Model	357
12.6	A Brief Analysis	361
12.7	Summary	364
	Exercises	365
	References and Bibliography	366
Part IV	DATA PROTECTION	
Chapter 13	Recovery	374
13.1	Introduction	374
13.2	Transactions	375
13.3	Transaction Recovery	377
13.4	System Recovery	380
13.5	Media Recovery	382
13.6	Two-Phase Commit	382
13.7	SQL Support	384

13.8	Summary 385
	Exercises 386
	References and Bibliography 386
	Answers to Selected Exercises 389
Chapter 14	Concurrency 391
14.1	Introduction 391
14.2	Three Concurrency Problems 392
14.3	Locking 394
14.4	The Three Concurrency Problems Revisited 396
14.5	Deadlock 398
14.6	Serializability 400
14.7	Levels of Isolation 402
14.8	Intent Locking 404
14.9	SQL Support 406
14.10	Summary 408
	Exercises 409
	References and Bibliography 410
	Answers to Selected Exercises 414
Chapter 15	Security 416
15.1	Introduction 416
15.2	General Considerations 416
15.3	Discretionary Access Control 418
15.4	Request Modification 423
15.5	Mandatory Access Control 425
15.6	Data Encryption 426
15.7	SQL Support 430
15.8	Summary 433
	Exercises 434
	References and Bibliography 435
	Answers to Selected Exercises 437
Chapter 16	Integrity 440
16.1	Introduction 440
16.2	General Considerations 441
16.3	Integrity Rules 442
16.4	Domain Rules 444
16.5	Attribute Rules 445
16.6	Relation Rules 446
16.7	Database Rules 449
16.8	State vs. Transition Rules 451
16.9	Candidate and Foreign Keys 452
16.10	SQL Support 454
16.11	Summary 456
	Exercises 457

References and Bibliography	457
Answers to Selected Exercises	459

Part V**FURTHER TOPICS****Chapter 17****Views 466**

17.1	Introduction	466
17.2	What Are Views For?	468
17.3	Data Definition	470
17.4	Data Manipulation: Retrieval Operations	472
17.5	Data Manipulation: Update Operations	472
17.6	Updating Union, Intersection, and Difference Views	475
17.7	Updating Other Views	479
17.8	Snapshots (A Digression)	489
17.9	SQL Support	490
17.10	Summary	492
	Exercises	493
	References and Bibliography	494
	Answers to Selected Exercises	496

Chapter 18**Optimization 501**

18.1	Introduction	501
18.2	An Introductory Example	503
18.3	The Optimization Process: An Overview	504
18.4	Expression Transformation	508
18.5	Database Statistics	513
18.6	A Divide and Conquer Strategy	514
18.7	Implementing the Relational Operators	518
18.8	Summary	523
	Exercises	524
	References and Bibliography	526
	Answers to Selected Exercises	541

Chapter 19**Domains, Relations, and Data Types 453**

19.1	Introduction	543
19.2	Domains	544
19.3	Relations	551
19.4	Type Inheritance	554
19.5	Relation-Valued Attributes (A Digression)	560
19.6	SQL Support	562
19.7	Summary	562
	Exercises	564
	References and Bibliography	565
	Answers to Selected Exercises	568

Chapter 20	Missing Information 570
20.1	Introduction 570
20.2	An Overview of the 3VL Approach 571
20.3	Some Consequences of the Foregoing Scheme 577
20.4	Outer Join (A Digression) 579
20.5	SQL Support 582
20.6	Summary 586
	Exercises 587
	References and Bibliography 589
	Answers to Selected Exercises 591
Chapter 21	Distributed Database and Client/Server Systems 593
21.1	Introduction 593
21.2	Some Preliminaries 593
21.3	The Twelve Objectives 598
21.4	Problems of Distributed Systems 605
21.5	Gateways 614
21.6	Client/Server Systems 616
21.7	SQL Support 620
21.8	Summary 621
	Exercises 622
	References and Bibliography 622
Part VI	OBJECT-ORIENTED SYSTEMS
Chapter 22	Basic OO Concepts 630
22.1	Introduction 630
22.2	A Motivating Example 632
22.3	Objects, Methods, and Messages 634
22.4	Objects and Object Classes Revisited 638
22.5	Object Identity Revisited 642
22.6	Classes, Instances, and Collections 642
22.7	Class Hierarchies 645
22.8	Miscellaneous Remarks 647
22.9	Summary 647
	Exercises 648
	References and Bibliography 649
	Answers to Selected Exercises 650
Chapter 23	A Cradle-to-Grave Example 652
23.1	Introduction 652
23.2	Data Definition 653
23.3	Populating the Database 656
23.4	Retrieval Operations 662
23.5	Update Operations 664

23.6	Summary 665
	Exercises 667
	References and Bibliography 667
	Answers to Selected Exercises 668
Chapter 24	Additional OO Concepts 669
24.1	Introduction 669
24.2	The Issues 669
24.3	Some Questions 675
24.4	Summary 681
	Exercises 683
	References and Bibliography 684
	Answers to Selected Exercises 684
Chapter 25	Toward an OO/Relational Rapprochement 686
25.1	Introduction 686
25.2	A Review of "The OO Model" 687
25.3	Domains vs. Objects 690
25.4	Relations vs. Objects 693
25.5	Benefits of True Rapprochement 702
25.6	Summary 704
	References and Bibliography 704

APPENDIXES

Appendix A	Storage Structures and Access Methods 712
A.1	Introduction 712
A.2	Database Access: An Overview 713
A.3	Page Sets and Files 717
A.4	Indexing 724
A.5	Hashing 733
A.6	Pointer Chains 738
A.7	Compression Techniques 740
A.8	Summary 746
	Exercises 746
	References and Bibliography 748
	Answers to Selected Exercises 757
Appendix B	DB2: A Relational Implementation 761
B.1	Introduction 761
B.2	Storage Structure 762
B.3	System Components 764
B.4	Detailed Control Flow 766
B.5	Compilation and Recompile 767
B.6	Utilities and Related Matters 769
B.7	Summary 771
	Exercises 772

	References and Bibliography	772
	Answers to Selected Exercises	773
Appendix C	Logic-Based Systems	775
C.1	Introduction	775
C.2	Overview	776
C.3	Propositional Calculus	778
C.4	Predicate Calculus	782
C.5	A Proof-Theoretic View of Databases	789
C.6	Deductive Database Systems	793
C.7	Recursive Query Processing	798
C.8	Summary	804
	Exercises	807
	References and Bibliography	808
	Answers to Selected Exercises	814
Appendix D	Abbreviations and Acronyms	817
Index		821

PART **I**

BASIC CONCEPTS

Part I consists of three introductory chapters.

- Chapter 1 sets the scene by explaining what a database is and why database systems are generally desirable. It also very briefly discusses the difference between *relational* database systems and others.
- Next, Chapter 2 presents a general architecture for database systems, the so-called *ANSI/SPARC architecture*. That architecture serves as a framework on which all later chapters in the book will build.
- Chapter 3 then presents a broad overview of relational systems, in order to serve as a gentle introduction to the much more comprehensive discussions of the same subject to follow in later parts of the book. It also provides a brief introduction to the standard relational language SQL, and introduces and explains the running example (the suppliers-and-parts database).

1 An Overview of Database Management

1.1 An Introductory Example

A **database system** is essentially nothing more than a *computerized record-keeping system*. The **database** itself can be regarded as a kind of electronic filing cabinet; in other words, it is a repository for a collection of computerized data files. The user of the system will be given facilities to perform a variety of operations on such files, including the following among others:

- Adding new, empty files to the database
- Inserting new data into existing files
- Retrieving data from existing files
- Updating data in existing files
- Deleting data from existing files
- Removing existing files, empty or otherwise, from the database

By way of illustration, Fig. 1.1 shows a very small database containing just a single file, called CELLAR, which in turn contains data concerning the contents of a wine cellar. Fig. 1.2 shows an example of a **retrieval** operation against that database, together with the data (more accurately, the *result*—but it is usual in database contexts to refer to results as data also) returned from that retrieval. *Note:* Throughout this book we show all database operations and suchlike material in upper case for clarity. In practice, it is often more convenient to enter such material in lower case. Most systems will accept both.

Fig. 1.3 gives examples, all more or less self-explanatory, of **insert**, **update**, and **delete** operations on the wine cellar database. Examples of adding and removing entire files will be given later, in Chapters 3 and 4.

To conclude this introductory section, a few final remarks:

- First, for obvious reasons, computerized files such as CELLAR in the example are frequently referred to as **tables** rather than files (in fact, they are **relational tables**—see Section 1.6).

BIN	WINE	PRODUCER	YEAR	BOTTLES	READY
2	Chardonnay	Buena Vista	92	1	94
3	Chardonnay	Geyser Peak	92	5	94
6	Chardonnay	Stonestreet	91	4	93
12	Jo. Riesling	Jekel	93	1	94
21	Fumé Blanc	Ch. St. Jean	92	4	94
22	Fumé Blanc	Robt. Mondavi	91	2	93
30	Gewurztraminer	Ch. St. Jean	93	3	94
43	Cab. Sauvignon	Windsor	86	12	95
45	Cab. Sauvignon	Geyser Peak	89	12	97
48	Cab. Sauvignon	Robt. Mondavi	88	12	99
50	Pinot Noir	Gary Farrell	91	3	94
51	Pinot Noir	Stemmler	88	3	95
52	Pinot Noir	Dehlinger	90	2	93
58	Merlot	Clos du Bois	89	9	95
64	Zinfandel	Lytton Spring	89	9	98
72	Zinfandel	Rafanelli	90	2	98

FIG. 1.1 The wine cellar database (CELLAR file)

- Second, the rows of such a table can be thought of as representing the **records** of the file (sometimes referred to explicitly as *logical* records, to distinguish them from other kinds of records to be discussed later). Likewise, the columns can be regarded as representing the **fields** of those logical records. In this book, we will tend to use the “record” and “field” terminology when we are talking about database systems in general, the “row” and “column” terminology when we are talking about relational systems specifically. (Actually, when we get to our more formal relational discussions in later parts of the book, we will switch to more formal terms anyway.)

Retrieval:		
SELECT WINE, BIN, PRODUCER		
FROM CELLAR		
WHERE READY = 95 ;		
Result (as shown on, e.g., a display screen):		
WINE	BIN	PRODUCER
Cab. Sauvignon	43	Windsor
Pinot Noir	51	Stemmler
Merlot	58	Clos du Bois

FIG. 1.2 Sample retrieval against the wine cellar database

<i>Inserting new data:</i> INSERT INTO CELLAR (BIN, WINE, PRODUCER, YEAR, BOTTLES, READY) VALUES (53, 'Pinot Noir', 'Saintybury', 92, 1, 96) ;
<i>Updating existing data:</i> UPDATE CELLAR SET BOTTLES = 4 WHERE BIN = 3 ;
<i>Deleting existing data:</i> DELETE FROM CELLAR WHERE BIN = 2 ;

FIG. 1.3 INSERT, UPDATE, and DELETE examples

- Third, the SELECT, INSERT, UPDATE, and DELETE operations shown in Figs. 1.2 and 1.3 above are actually all examples of statements from a database language called **SQL**. SQL is the language currently supported by most commercial database products; in fact, it is the official standard language for dealing with relational systems (see further discussion in Section 1.6 below). The name “SQL” was originally an abbreviation for “Structured Query Language,” and was pronounced “sequel.” Now that the language has become a standard, however, the name is just a name—it is not officially an abbreviation for anything at all—and the pendulum has swung in favor of the pronunciation “ess-cue-ell.” We will assume this latter pronunciation in this book.

1.2 What Is a Database System?

To repeat from Section 1.1, a database system is basically a computerized record-keeping system; that is, it is a computerized system whose overall purpose is to maintain information and to make that information available on demand. The information concerned can be anything that is deemed to be of significance to the individual or organization the system is intended to serve—anything, in other words, that is needed to assist in the general process of running the business of that individual or organization.

Note: The terms “data” and “information” are treated as synonymous in this book. Some writers prefer to distinguish between the two, using “data” to refer to the values actually stored in the database and “information” to refer to the *meaning* of those values as understood by some user. The distinction is clearly important—so important that it seems preferable to make it explicit, where relevant, instead of relying on a somewhat arbitrary differentiation between two essentially similar terms.

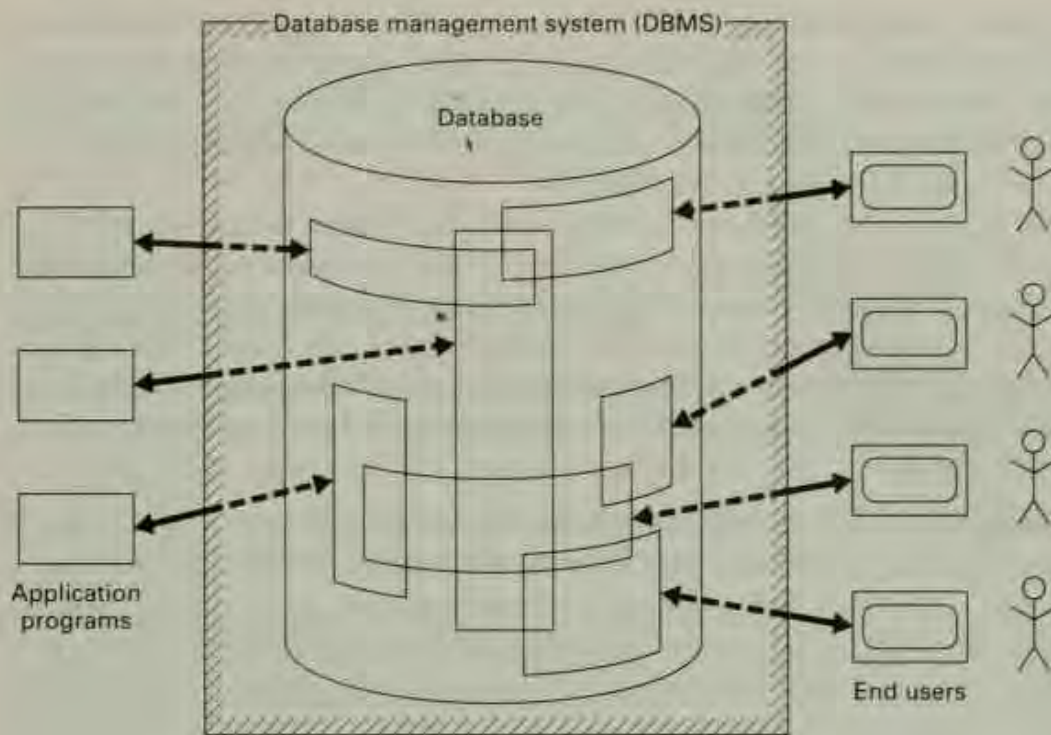


FIG. 1.4 Simplified picture of a database system

Fig. 1.4 shows a greatly simplified view of a database system. The figure is intended to illustrate the point that a database system involves four major components, namely, **data**, **hardware**, **software**, and **users**. We consider these four components briefly below. Later, of course, we will discuss each in much more detail (except for the hardware component, most details of which are beyond the scope of this book).

Data

Database systems are available on machines that range all the way from quite small micros (even portable PCs) to the largest mainframes. Needless to say, the facilities provided by any given system are to some extent determined by the size and power of the underlying machine. In particular, systems on large machines ("large systems") tend to be *multi-user*, whereas those on smaller machines ("small systems") tend to be *single-user*. A **single-user system** is a system in which at most one user can access the database at any given time; a **multi-user system** is a system in which many users can access the database concurrently. As Fig. 1.4 suggests, we will normally assume the latter case in this book, for generality, but in fact the distinction is largely irrelevant so far as most users are concerned: A major objective of most multi-user systems is precisely to allow each individual user to behave as if he or she were working with a *single-user* system. The special problems of multi-user systems are primarily problems that are internal to the system, not ones that are visible to the user (see Part IV of this book, especially Chapter 14).

Incidentally, it is usually convenient to assume for the sake of simplicity that the

totality of data stored in the system is all held in a single database, and we will normally make this assumption, since it does not materially affect any of our other discussions. In practice, however, there might be good reasons, even in a small system, why the data should be split across several distinct databases. We will touch on some of those reasons elsewhere in this book (e.g., in Chapter 2).

In general, then, the data in the database—at least in a large system—will be both *integrated* and *shared*. As we will see in Section 1.4, these two aspects, data integration and data sharing, represent a major advantage of database systems in the “large” environment; and data integration, at least, can be significant in the “small” environment also. Of course, there are many additional advantages also (to be discussed later), even in the small environment. But first let us explain what we mean by the terms “integrated” and “shared.”

- By **integrated**, we mean that the database can be thought of as a unification of several otherwise distinct data files, with any redundancy among those files wholly or partly eliminated. For example, a given database might contain both an EMPLOYEE file, giving employee names, addresses, departments, salaries, etc., and an ENROLLMENT file, representing the enrollment of employees in training courses (refer to Fig. 1.5). Now suppose that, in order to carry out the process of training course administration, it is necessary to know the department for each enrolled student. Then there is clearly no need to include that information, redundantly, in the ENROLLMENT file, because it can always be discovered by referring to the EMPLOYEE file instead.
- By **shared**, we mean that individual pieces of data in the database can be shared among several different users, in the sense that each of those users can have access to the same piece of data (and different users can use it for different purposes). As indicated earlier, different users can even be accessing the same piece of data *at the same time* (“concurrent access”). Such sharing (concurrent or otherwise) is partly a consequence of the fact that the database is integrated. In the EMPLOYEE-ENROLLMENT example cited above, the department information in the EMPLOYEE file would typically be shared by users in the Personnel Department and users in the Education Department—and, as suggested above, those two classes of users would typically use that information for different purposes.

Another consequence of the fact that the database is integrated is that any given user will typically be concerned only with some small portion of the total database

EMPLOYEE	NAME	ADDRESS	DEPARTMENT	SALARY	...
ENROLLMENT	NAME	COURSE	...		

FIG. 1.5 The EMPLOYEE and ENROLLMENT files

(moreover, different users' portions will overlap in many different ways). In other words, a given database will be perceived by different users in a variety of different ways. In fact, even when two users share the same portion of the database, their views of that portion might differ considerably at a detailed level. This latter point is discussed more fully in Section 1.5 and in the next chapter.

Hardware

The hardware portions of the system consist of:

- The secondary storage volumes—typically moving-head magnetic disks—that are used to hold the stored data, together with the associated I/O devices (disk drives, etc.), device controllers, I/O channels, and so forth; and
- The processor(s) and associated main memory that are used to support the execution of the database system software (see the next subsection below).

This book does not concern itself very greatly with the hardware portions of the system, for the following reasons among others: First, these aspects form a major topic in their own right; second, the problems encountered in this area are not peculiar to database systems; and third, those problems have been very thoroughly investigated and documented in numerous other places.

Software

Between the physical database itself (i.e., the data as actually stored) and the users of the system is a layer of software, the **database manager** (DB manager) or, more usually, **database management system** (DBMS). All requests from users for access to the database are handled by the DBMS: the facilities sketched in Section 1.1 for adding and removing files (or tables), retrieving data from and updating data in such files or tables, and so forth, are all facilities provided by the DBMS. One general function provided by the DBMS is thus *the shielding of database users from hardware-level details* (much as programming-language systems shield application programmers from hardware-level details). In other words, the DBMS provides users with a view of the database that is elevated somewhat above the hardware level, and supports user operations (such as the SQL operations discussed briefly in Section 1.1) that are expressed in terms of that higher-level view. We shall discuss this function, and other functions of the DBMS, in considerably more detail throughout the body of this book.

Note: The DBMS is easily the most important software component in the overall system, but it is not the only one. Others include utilities, application development tools, design aids, report writers, and so on. See Chapter 2 for further discussion.

Users

We consider three broad classes of users:

- First, there are the **application programmers**, who are responsible for writing ap-

plication programs that use the database, typically in a language such as COBOL or PL/I or some more modern language such as C or Pascal. Those programs operate on the data in all the usual ways—retrieving existing information, inserting new information, deleting or changing existing information. All of these functions are of course performed by issuing the appropriate request to the DBMS. The programs themselves may be conventional batch applications, or they may be **online** applications, whose function is to support an end user (see the next paragraph) who is accessing the database from an online workstation or terminal. Most modern applications are of the online variety.

- The second class of user, then, is **end users**, who interact with the system from online workstations or terminals. A given end user can access the database via one of the online applications mentioned in the previous paragraph, or he or she can use an interface provided as an integral part of the database system software. Such interfaces are also supported by means of online applications, of course, but those applications are **builtin**, not user-written. Most systems provide at least one such builtin application, namely an interactive **query language processor**, by which the user is able to issue high-level commands or statements (such as SELECT, INSERT, etc.) to the DBMS. The language SQL mentioned in Section 1.1 can be regarded as a typical example of a database query language.

Note: The term “query language,” common though it is, is really a misnomer, inasmuch as the English verb “query” suggests *retrieval* (only), whereas query languages typically provide UPDATE, INSERT, and DELETE operations (and probably other operations) as well.

Most systems also provide additional builtin interfaces in which users do not issue explicit commands such as SELECT at all, but instead operate by (e.g.) choosing items from a menu or filling in boxes on a form. Such **menu- or forms-driven** interfaces tend to be easier to use for people who do not have a formal training in IT (IT = Information Technology; the abbreviation IS = Information Systems is often used with much the same meaning). By contrast, **command-driven interfaces**—i.e., query languages—do tend to require a certain amount of professional IT expertise, though perhaps not a very great deal (obviously not as much as is needed to write an application program in a language like COBOL). Then again, a command-driven interface is likely to be more flexible than a menu- or forms-driven one, in that query languages typically provide certain functions that are not supported by those other interfaces.

- The third class of user (not shown in Fig. 1.4) is the **database administrator** or DBA. Discussion of the DBA function—and the associated (very important) **data administrator** function—is deferred to Sections 1.4 and 2.7.

This completes our preliminary description of the major aspects of a database system. We now go on to discuss the ideas in somewhat more detail.

1.3 What Is a Database?

Persistent Data

It is customary to refer to the data in a database as “persistent” (even though it might not actually persist for very long!). By “persistent,” we mean to suggest that database data differs in kind from other, more ephemeral, data, such as input data, output data, control statements, work queues, software control blocks, intermediate results, and more generally any data that is transient in nature. Let us elaborate briefly on the terms “input data” and “output data”:

- “Input data” refers to information entering the system for the very first time (typically from a terminal or workstation). Such information might cause a change to be made to the persistent data (it might *become* part of the persistent data), but it is not initially part of the database as such.
- Similarly, “output data” refers to messages and results emanating from the system (typically printed or displayed on a screen). Again, such information might be *derived from* the persistent data, but it is not itself considered to be part of the database.

Of course, the distinction between persistent and transient data is not a hard and fast one—it depends to some extent on context (i.e., how the data is being used). However, assuming that the distinction does at least make some intuitive sense, we can now give a slightly more precise definition of the term “database”:

- A **database** consists of some collection of persistent data that is used by the application systems of some given enterprise.

The term “enterprise” here is simply a convenient generic term for any reasonably self-contained commercial, scientific, technical, or other organization. An enterprise might be a single individual (with a small private database), or a complete corporation or similar large body (with a very large shared database), or anything in between. Here are some examples:

1. A manufacturing company
2. A bank
3. A hospital
4. A university
5. A government department

Any enterprise must necessarily maintain a lot of data about its operation. This is the “persistent data” referred to above. The enterprises just mentioned would typically include the following among their persistent data:

1. Product data
2. Account data
3. Patient data

4. Student data
5. Planning data

Note: The first few editions of this book used the term “operational data” in place of “persistent data.” That earlier term reflected the original emphasis in database systems on **operational** or **production** applications—i.e., routine, highly repetitive applications that were executed over and over again to support the day-to-day operation of the enterprise (for example, an application to support the deposit or withdrawal of cash in a banking system). Now that databases are increasingly being used for other kinds of application as well—i.e., **decision support** applications—the term “operational data” is no longer entirely appropriate. Indeed, enterprises nowadays often maintain two distinct databases, one containing operational data and one containing decision support data. The decision support database frequently consists of *summary information* (e.g., totals, averages), where that summary information in turn is extracted from the operational database on a periodic basis—say once a day or once a week.

Entities and Relationships

Let us consider the example of a manufacturing company in a little more detail. Such an enterprise will typically wish to record information about the *projects* it has on hand; the *parts* used in those projects; the *suppliers* who supply those parts; the *warehouses* in which the parts are stored; the *employees* who work on the projects; and so on. Projects, parts, suppliers, etc., thus constitute the basic **entities** about which the company needs to record information (the term “entity” is widely used in database circles to mean any distinguishable object that is to be represented in the database). Refer to Fig. 1.6.

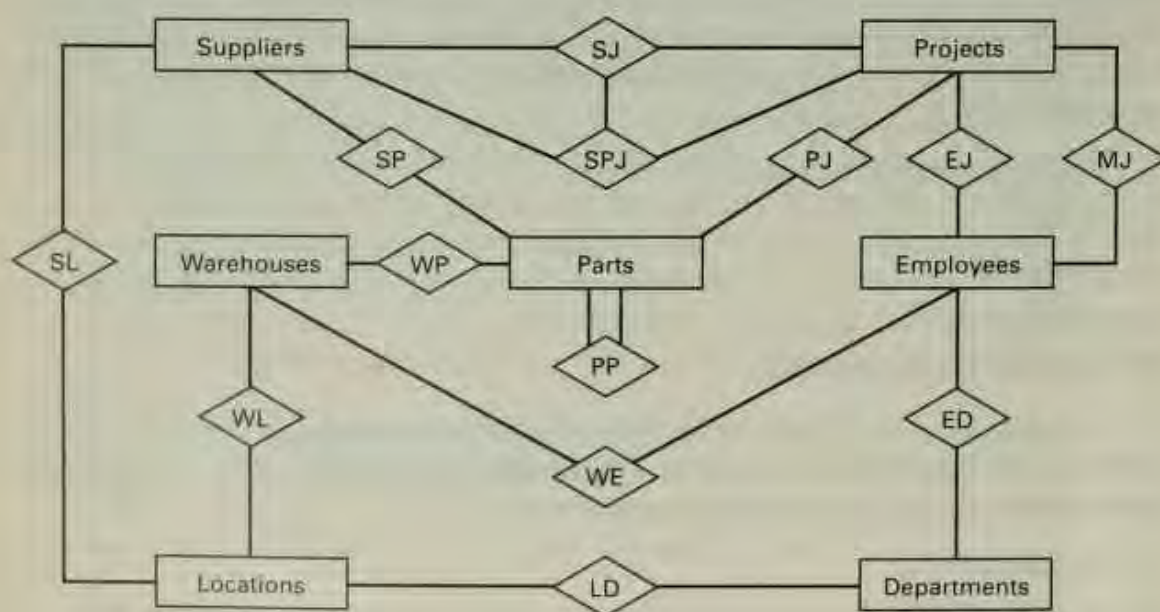


FIG. 1.6 A simple entity/relationship (E/R) diagram

It is important to understand that, in addition to the basic entities themselves, there will also be **relationships** linking those basic entities together. Such relationships are represented by diamonds and connecting lines in Fig. 1.6. For example, there is a relationship ("SP") between suppliers and parts: Each supplier supplies certain parts, and conversely each part is supplied by certain suppliers (more accurately, each supplier supplies certain *kinds* of parts, each *kind* of part is supplied by certain suppliers). Similarly, parts are used in projects, and conversely projects use parts (relationship PJ); parts are stored in warehouses, and warehouses store parts (relationship WP); and so on. Note that these relationships are all *bidirectional*—that is, they can be traversed in either direction. For example, relationship SP between suppliers and parts can be used to answer either of the following questions:

- Given a supplier, find the parts supplied by that supplier
- Given a part, find the suppliers who supply that part

The significant point about this relationship, and all of the others illustrated in the figure, is that *they are just as much a part of the data as are the basic entities*. They must therefore be represented in the database, just like the basic entities. Later in this book we will consider ways in which this can be done.

Incidentally, Fig. 1.6 is a simple example of what is called (for obvious reasons) an **entity/relationship diagram** (E/R diagram for short). In Chapter 12 we will consider such diagrams in some detail.

Fig. 1.6 also illustrates a number of other points:

1. Although most of the relationships in the diagram involve *two* types of entity—i.e., they are *binary* relationships—it is by no means the case that all relationships must necessarily be binary in this sense. In the example there is one relationship ("SPJ") involving three types of entity (suppliers, parts, and projects)—a *ternary* relationship. The intended interpretation is that certain suppliers supply certain parts to certain projects. Note carefully that this ternary relationship ("suppliers supply parts to projects") is *not* equivalent, in general, to the combination of the three binary relationships "suppliers supply parts," "parts are used in projects," and "projects are supplied by suppliers." For example, the statement that

(a) Smith supplies monkey wrenches to the Manhattan project

tells us *more* than the following three statements do:

(b) Smith supplies monkey wrenches,

(c) Monkey wrenches are used in the Manhattan project, and

(d) The Manhattan project is supplied by Smith

—we cannot (validly!) infer (a) knowing only (b), (c), and (d). More precisely, if we know (b), (c), and (d), then we might be able to infer that Smith supplies monkey wrenches to *some* project (say project Jz), that *some* supplier (say supplier Sx) supplies monkey wrenches to the Manhattan project, and that Smith supplies *some* part (say part Py) to the Manhattan project—but we cannot validly infer that Sx is Smith or that Py is monkey wrenches or that Jz is the Manhattan project. False

inferences such as these are examples of what is sometimes called **the connection trap**.

2. The diagram also includes one relationship (PP) involving just *one* type of entity (parts). The relationship here is that certain parts include other parts as immediate components (the so-called **bill-of-materials** relationship)—for example, a screw is a component of a hinge assembly, which is also considered as a part and might in turn be a component of some higher-level part such as a lid. Note that this relationship is still binary; it is just that the two types of entity that are linked together, namely parts and parts, happen to be one and the same.
3. In general, a given set of entity types might be linked together in any number of distinct relationships. In the diagram, there are two relationships between projects and employees: One (EJ) represents the fact that employees are assigned to projects, the other (MJ) represents the fact that employees manage projects.

Note carefully that a relationship can be regarded as an entity in its own right. If we take as our definition of entity “any object about which we wish to record information,” then a relationship certainly fits the definition. For instance, “part P4 is stored in warehouse W8” is an entity about which we might well wish to record information—e.g., the corresponding quantity. Moreover, there are definite advantages (beyond the scope of the present chapter) to be obtained by not making any unnecessary distinctions between entities and relationships. In this book, therefore, we will generally treat relationships as just a special kind of entity.

Properties

As just indicated, we regard an entity as any object about which we wish to record information. In other words, entities (and hence relationships also) have **properties**. For example, suppliers have *locations*; parts have *weights*; projects have *priorities*; assignments have *start dates*; and so on. Such properties must therefore be represented in the database also. For example, the database might include a record type S representing the “suppliers” entity type, and that record type in turn might include a field type CITY representing the “location” property.

Properties in turn might be very simple in nature, or they might have an internal structure of arbitrary complexity. For example, the “supplier location” property is presumably quite simple, consisting as it does of just a city name, and can be represented in the database by a simple character string. By contrast, a warehouse might have a “floor plan” property, and that property might be quite complex, consisting perhaps of an entire architectural drawing plus associated descriptive text. Current database products are mostly not very good at dealing with complex properties such as drawings or text. We will return to this topic later in this book (especially in Chapter 19 and Chapters 22–25); until then, we will generally assume (where it makes any difference) that all properties are “simple” and can be represented by “simple” data types in the database. Examples of such “simple” data types include numbers, strings, dates, times, etc.

1.4 Why Database?

Why use a database system? What are the advantages? To some extent the answer to these questions depends on whether the system in question is single- or multi-user (or rather, to be more accurate, there are numerous *additional* advantages in the multi-user case). Let us consider the single-user case first.

Refer back to the wine cellar example once again (Fig. 1.1), which we can regard as typical of a single-user database. Now, that particular database is so small and so simple that the advantages might not be very immediately obvious. But imagine a similar database for a large restaurant, with a stock of perhaps thousands of bottles and with very frequent changes to that stock; or think of a liquor store, with again a very large stock and with high turnover on that stock. (These would typically still be single-user systems, incidentally, even though the database is larger.) The advantages of a database system over traditional, paper-based methods of record-keeping will perhaps be more readily apparent in these examples. Here are some of them:

- *Compactness*: No need for possibly voluminous paper files.
- *Speed*: The machine can retrieve and change data far faster than a human can. In particular, *ad hoc*, spur-of-the-moment queries (e.g., "Do we have more Zinfandel than Pinot Noir?") can be answered quickly without any need for time-consuming manual or visual searches.
- *Less drudgery*: Much of the sheer tedium of maintaining files by hand is eliminated. Mechanical tasks are always better done by machines.
- *Currency*: Accurate, up-to-date information is available on demand at any time.

The foregoing benefits apply with even more force in a multi-user environment, of course, where the database is likely to be much larger and much more complex than in the single-user case. However, there is one overriding additional advantage in such an environment, namely as follows: *The database system provides the enterprise with centralized control of its data* (which, as the reader should realize from Section 1.3, is one of its most valuable assets). Such a situation contrasts sharply with that found in an enterprise without a database system, where typically each application has its own private files—quite often its own private tapes and disks, too—so that the data is widely dispersed and might thus be difficult to control in any systematic way.

Data Administration and Database Administration

Let us elaborate a little on this concept of centralized control. The concept implies that (in an enterprise with a database system) there will be some identifiable person who has this central responsibility for the data. That person is the **data administrator** (sometimes abbreviated DA) mentioned briefly at the end of Section 1.2. Given that (as indicated above) the data is one of the enterprise's most valuable assets, it is imperative that there should be some person who understands the data, and the needs of the enterprise with respect to the data, *at a senior management level*. The data administrator is that

person. Thus, it is the data administrator's job to decide what data should be stored in the database in the first place, and to establish policies for maintaining and dealing with that data once it has been stored. An example of such a policy would be one that dictates who can perform what operations on what data in what circumstances—in other words, a *data security* policy (see further discussion below).

Note carefully that the data administrator is a manager, not a technician (although he or she certainly does need to have some appreciation of the capabilities of database systems at a technical level). The *technical* person responsible for implementing the data administrator's decisions is the **database administrator** (usually abbreviated DBA). Thus, the DBA, unlike the data administrator, is an *IT professional*. The job of the DBA is to create the actual database and to implement the technical controls needed to enforce the various policy decisions made by the data administrator. The DBA is also responsible for ensuring that the system operates with adequate performance and for providing a variety of other related technical services. The DBA will typically have a staff of systems programmers and other technical assistants (i.e., the DBA function will typically be performed in practice by a team of several people, not just by one person); for simplicity, however, it is convenient to assume that the DBA is indeed a single individual. We will discuss the DBA function in more detail in Chapter 2.

Benefits of the Database Approach

We close this section by identifying some of the specific advantages that accrue from the notion of centralized control of the data.

- Redundancy can be reduced.

In nondatabase systems each application has its own private files. This fact can often lead to considerable redundancy in stored data, with resultant waste in storage space. For example, a personnel application and an education-records application might both own a file that includes department information for employees. As suggested in Section 1.2, those two files can be integrated, and the redundancy eliminated, *if* the data administrator is aware of the data requirements for both applications—i.e., *if* the enterprise has the necessary overall control.

Incidentally, we do not mean to suggest that *all* redundancy can or necessarily should be eliminated. Sometimes there are sound business or technical reasons for maintaining several distinct copies of the same stored data. However, we do mean to suggest that any such redundancy should be *carefully controlled*—that is, the DBMS should be aware of it, if it exists, and should assume responsibility for “propagating updates” (see the next point below).

- Inconsistency can be avoided (to some extent).

This is really a corollary of the previous point. Suppose that a given fact about the real world—say the fact that employee E3 works in department D8—is represented by two distinct entries in the stored database. Suppose also that the DBMS is not aware of this duplication (i.e., the redundancy is not controlled). Then there will necessarily be occasions on which the two entries will not agree—namely, when one of the two has been updated and the other has not. At such times the

database is said to be *inconsistent*. Clearly, a database that is in an inconsistent state is capable of supplying incorrect or contradictory information to its users.

It should also be clear that if the given fact is represented by a single entry (i.e., if the redundancy is removed), then such an inconsistency cannot occur. Alternatively, if the redundancy is not removed but is controlled (by making it known to the DBMS), then the DBMS could guarantee that the database is never inconsistent *as seen by the user*, by ensuring that any change made to either of the two entries is automatically applied to the other one also. This process is known as **propagating updates**—where (as is usually the case) the term “update” is taken to include all of the operations of insertion, deletion, and modification. Note, however, that few commercially available systems today are capable of automatically propagating updates in this manner; that is, most current products do not support controlled redundancy at all, except in certain special situations.

- The data can be shared.

We discussed this point in Section 1.2, but for completeness we mention it again here. Sharing means not only that existing applications can share the data in the database, but also that new applications can be developed to operate against that same stored data. In other words, it might be possible to satisfy the data requirements of new applications without having to create any additional stored data.

- Standards can be enforced.

With central control of the database, the DBA (under the direction of the data administrator) can ensure that all applicable standards are observed in the representation of the data. Applicable standards might include any or all of the following: corporate, installation, departmental, industry, national, and international standards. Standardizing data representation is particularly desirable as an aid to *data interchange*, or migration of data between systems (this consideration is becoming particularly important with the advent of distributed processing technology—see Section 2.12). Likewise, data naming and documentation standards are also very desirable as an aid to data sharing and understandability.

- Security restrictions can be applied.

Having complete jurisdiction over the database, the DBA (a) can ensure that the only means of access to the database is through the proper channels, and hence (b) can define security rules to be checked whenever access is attempted to sensitive data (again, under appropriate direction from the data administrator). Different rules can be established for each type of access (retrieve, insert, delete, etc.) to each piece of information in the database. Note, however, that without such rules the security of the data might actually be *more* at risk than in a traditional (dispersed) filing system; that is, the centralized nature of a database system in a sense *requires* that a good security system be in place also.

- Integrity can be maintained.

The problem of integrity is the problem of ensuring that the data in the database is accurate. Inconsistency between two entries that purport to represent the same “fact” is an example of lack of integrity (see the discussion of this point

above); of course, that particular problem can arise only if redundancy exists in the stored data. Even if there is no redundancy, however, the database might still contain incorrect information. For example, an employee might be shown as having worked 400 hours in the week instead of 40, or as belonging to a department D9 when no such department exists. Centralized control of the database can help in avoiding such problems—insofar as they can be avoided—by permitting the data administrator to define (and the DBA to implement) integrity rules to be checked whenever any data update operation is attempted. (Again we are using the term “update” generically to cover all of the operations of insertion, deletion, and modification.)

It is worth pointing out that data integrity is even more important in a multiuser database system than it is in a “private files” environment, precisely because the database is shared. For without appropriate controls it would be possible for one user to update the database incorrectly, thereby generating bad data and so “infecting” other innocent users of that data. It should also be mentioned that most database products tend to be somewhat weak in their support for integrity controls, although there have been some recent improvements in this area.

- Conflicting requirements can be balanced.

Knowing the overall requirements of the enterprise—as opposed to the requirements of individual users—the DBA (under the data administrator’s direction, as always) can so structure the system as to provide an overall service that is “best for the enterprise.” For example, a representation can be chosen for the data in storage that gives fast access for the most important applications (possibly at the cost of poorer performance for certain other applications).

Most of the advantages listed above are probably fairly obvious. However, one further point, which might not be so obvious (although it is in fact implied by several of the others) needs to be added to the list—namely, *the provision of data independence*. (Strictly speaking, this is an *objective* for database systems, rather than an advantage necessarily.) The concept of data independence is so important that we devote a separate section to it.

1.5 Data Independence

Data independence can most easily be explained by first explaining its opposite. Applications implemented on older systems tend to be data-dependent. What this means is that the way in which the data is organized in secondary storage, and the technique for accessing it, are both dictated by the requirements of the application under consideration, and moreover that *knowledge of that data organization and that access technique is built into the application logic and code*.

- *Example:* Suppose we have an application that processes the EMPLOYEE file, and suppose it is decided, for performance reasons, that the file is to be stored indexed on the “employee name” field. In an older system, the application in question will typically be aware of the fact that the index exists, and aware also of the file se-

quence as defined by that index, and the internal structure of the application will be built around that knowledge. In particular, the precise form of the various data access and exception-checking procedures within the application will depend very heavily on details of the interface presented to the application by the data management software.

We say that an application such as the one in this example is **data-dependent**, because it is impossible to change the storage structure (how the data is physically stored) or access technique (how it is accessed) without affecting the application, probably drastically. For instance, it would not be possible to replace the indexed file in the example by a hash-addressed file without making major modifications to the application. What is more, the portions of the application requiring alteration in such a case are precisely those portions that communicate with the data management software; the difficulties involved are quite irrelevant to the problem the application was originally written to solve—i.e., they are difficulties *introduced* by the nature of the data management interface.

In a database system, however, it would be extremely undesirable to allow applications to be data-dependent, for at least the following two reasons:

1. Different applications will need different views of the same data. For example, suppose that before the enterprise introduces its integrated database, there are two applications, *A* and *B*, each owning a private file that includes the field "customer balance." Suppose, however, that application *A* records this field in decimal, whereas application *B* records it in binary. It will still be possible to integrate the two files, and to eliminate the redundancy, provided the DBMS is ready and able to perform all necessary conversions between the stored representation chosen (which might be decimal or binary or something else again) and the form in which each application wishes to see it. For example, if it is decided to store the field in decimal, then every access by *B* will require a conversion to or from binary.

This is a fairly trivial example of the kind of difference that might exist in a database system between the data as seen by a given application and the data as physically stored. Many other possible differences will be considered later.

2. The DBA must have the freedom to change the storage structure or access technique in response to changing requirements, without having to modify existing applications. For example, new kinds of data might be added to the database; new standards might be adopted; application priorities (and therefore relative performance requirements) might change; new types of storage device might become available; and so on. If applications are data-dependent, such changes will typically require corresponding changes to be made to programs, thus tying up programmer effort that would otherwise be available for the creation of new applications. It is still not uncommon, even today, to find that 25 percent or even more of the programming effort available in the installation is devoted to this kind of maintenance activity—clearly a waste of a scarce and valuable resource.

It follows that the provision of data independence is a major objective of database systems. Data independence can be defined as **the immunity of applications to change in storage structure and access technique**—which implies, of course, that the

applications concerned do not depend on any one particular storage structure or access technique. In Chapter 2, we describe an architecture for database systems that provides a basis for achieving the data independence objective. Before then, however, let us consider in more detail some examples of the types of change that the DBA might wish to make, and that we might therefore wish applications to be immune to.

We start by defining three terms: *stored field*, *stored record*, and *stored file* (refer to Fig. 1.7).

- A **stored field** is the smallest unit of stored data. The database will, in general, contain many **occurrences** (or **instances**) of each of several **types** of stored field. For example, a database containing information about parts would probably include a stored field type called "part number," and there would be one occurrence of this stored field for each kind of part (screw, hinge, lid, etc.).

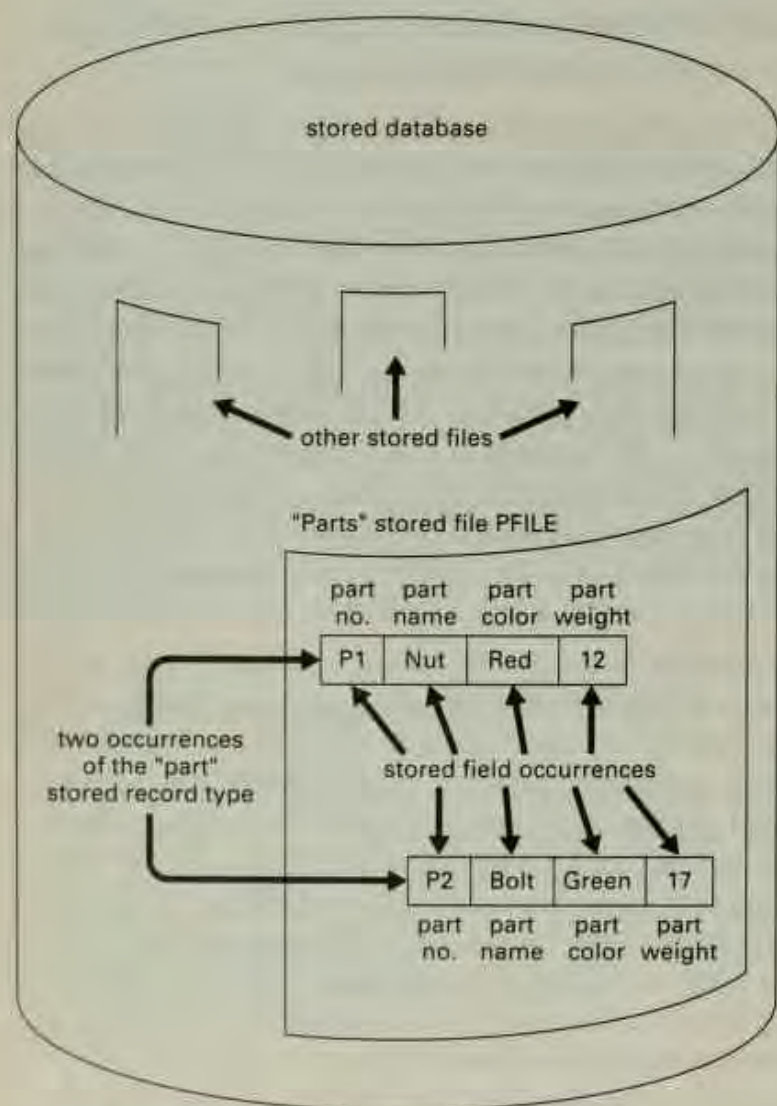


FIG. 1.7 Stored fields, records, and files

- A **stored record** is a collection of related stored fields. Again we distinguish between type and occurrence. A stored record **occurrence** (or **instance**) consists of a group of related stored field occurrences. For example, a stored record occurrence in the “parts” database might consist of an occurrence of each of the following stored fields: part number, part name, part color, and part weight. We say that the database contains many occurrences of the “part” stored record **type** (again, one occurrence for each distinct kind of part).

As an aside, we note that it is common to drop the qualifiers “type” and “occurrence” and to rely on context to indicate which is meant. Although there is a slight risk of confusion, the practice is convenient, and we will adopt it ourselves from time to time in this book.

- Finally, a **stored file** is the collection of all occurrences of one type of stored record. *Note:* We deliberately ignore the possibility of a stored file containing more than one type of stored record. This is another simplifying assumption that does not materially affect any of our subsequent discussions.

Now, in nondatabase systems it is usually the case that an application’s *logical* record is identical to some corresponding *stored* record. As we have already seen, however, this is not necessarily the case in a database system, because the DBA might need to be able to make changes to the storage structure—that is, to the stored fields, records, and files—while the corresponding logical structure does *not* change. For example, the “part weight” field mentioned above might be stored in binary to economize on storage space, whereas a given COBOL application might see it as a PICTURE item (i.e., as a character string). And later the DBA might decide for some reason to change the stored representation of that field from binary to decimal, and yet still allow the application to see it in character form.

As stated earlier, a difference such as this one, involving data type conversion on a particular field on each access, is comparatively minor; in principle, however, the difference between what the application sees and what is actually stored might be quite considerable. To amplify this remark, we present below a list of aspects of the database storage structure that might be subject to variation. The reader should consider in each case what the DBMS would have to do to protect an application from such variation (and indeed whether such protection can always be achieved).

- Representation of numeric data

A numeric field might be stored in internal arithmetic form (e.g., in packed decimal) or as a character string. Either way, the DBA must choose an appropriate base (e.g., binary or decimal), scale (fixed or floating point), mode (real or complex), and precision (number of digits). Any of these aspects might be changed to improve performance or to conform to a new standard or for many other reasons.

- Representation of character data

A character string field might be stored using any of several distinct coded character sets or “forms-of-use” (e.g., ASCII, EBCDIC).

- Units for numeric data

The units in a numeric field might change—from inches to centimeters, for example, during a process of metrication.

- Data encoding

In some situations it might be desirable to represent data in storage by coded values. For example, the “part color” field, which an application sees as a character string (“Red” or “Blue” or “Green” ...), might be stored as a single decimal digit, interpreted according to the coding scheme 1 = “Red,” 2 = “Blue,” and so on.

- Data materialization

In practice the logical field seen by an application will usually correspond to some specific stored field (although, as we have already seen, there might be differences in data type, units, and so on). In such a case, the process of materialization—that is, constructing an occurrence of the logical field from the corresponding stored field occurrence and presenting it to the application—can be said to be *direct*. Sometimes, however, a logical field will have no single stored counterpart; instead, its values will be materialized by means of some computation performed on a set of several stored field occurrences. For example, values of the logical field “total quantity” might be materialized by summing a number of individual stored quantity values. “Total quantity” here is an example of a **virtual** field, and the materialization process is said to be *indirect*. Note, however, that the user might see a difference between real and virtual fields, inasmuch as it might not be (directly) possible to insert or modify an occurrence of a virtual field.

- Structure of stored records

Two existing stored records might be combined into one. For example, the stored records

Part Number	Color	and	Part Number	Weight
-------------	-------	-----	-------------	--------

could be combined to form

Part Number	Color	Weight
-------------	-------	--------

Such a change might occur as predatabase applications are brought into the database system. It implies that an application’s logical record might consist of a subset of the corresponding stored record—that is, certain fields in that stored record would be invisible to the application in question.

Alternatively, a single stored record type might be split into two. Reversing the previous example, the stored record type

Part Number	Color	Weight
-------------	-------	--------

could be broken down into

Part Number	Color	and	Part Number	Weight
-------------	-------	-----	-------------	--------

Such a split would allow less frequently used portions of the original record to be stored on a slower device, for example. The implication is that an application's logical record might contain fields from several distinct stored records—that is, it would be a superset of any given one of those stored records.

■ Structure of stored files

A given stored file can be physically implemented in storage in a wide variety of ways. For example, it might be entirely contained within a single storage volume (e.g., a single disk), or it might be spread across several volumes on several different device types; it might or might not be physically sequenced according to the values of some stored field; it might or might not be sequenced in one or more additional ways by some other means, e.g., by one or more indexes or one or more embedded pointer chains (or both); it might or might not be accessible via hash-addressing; the stored records might or might not be physically blocked (several per physical record); and so on. But none of these considerations should affect applications in any way (other than in performance, of course).

This concludes our list of aspects of the storage structure that are subject to possible change. The list implies (among other things) that the database should be able to **grow** without affecting existing applications; indeed, enabling the database to grow without logically impairing existing applications is probably the single most important reason for requiring data independence in the first place. For example, it should be possible to extend an existing stored record by the addition of new stored fields, representing, typically, further information concerning some existing type of entity (e.g., a “unit cost” field might be added to the “part” stored record). Such new fields should simply be invisible to existing applications. Likewise, it should be possible to add entirely new types of stored record (and hence new stored files), again without requiring any change to existing applications; such records would typically represent new types of entity (e.g., a “supplier” record type could be added to the “parts” database). Again, such additions should be invisible to existing applications.

We close this section by noting that data independence is not an absolute—different systems provide it in different degrees. To put this another way, few systems, if any, provide no data independence at all; it is just that some systems are less data-independent than others. Modern systems tend to be more data-independent than older systems, but they are still not perfect, as we will see in some of the chapters to come.

1.6 Relational Systems and Others

Almost all of the database products developed since the late 1970s have been based on what is called **the relational approach**; what is more, the vast majority of database research over the last 25 years has also been based—albeit a little indirectly, in some

cases—on that approach. In fact, it is undeniable that the relational approach represents the dominant trend in the marketplace today, and that the “relational model” (see Part II of this book) is the single most important development in the entire history of the database field. For these reasons, plus the additional reason that the relational model is solidly based on certain aspects of mathematics and therefore provides an ideal vehicle for teaching the concepts and principles of database systems, the emphasis in this book is very heavily on relational systems and the relational approach.

What then does it mean to say that a system is relational? It is unfortunately not possible to answer this question fully at this early point in our discussions; however, it is possible, and desirable, to give a rough-and-ready answer, which we can make more precise later. Briefly, a relational system is a system in which:

1. The data is perceived by the user as tables (and nothing but tables); and
2. The operators at the user's disposal (e.g., for data retrieval) are operators that generate new tables from old. For example, there will be one operator to extract a subset of the rows of a given table, and another to extract a subset of the columns—and of course a row subset and a column subset of a table can both in turn be regarded as tables themselves.

The reason such systems are called “relational” is that the term “relation” is essentially just a mathematical term for a table. For most practical purposes, indeed, the terms “relation” and “table” can be taken as synonymous. See Part II of this book for further discussion.

As indicated, we will make the foregoing definition considerably more precise later, but it will serve for the time being. Fig. 1.8 provides an illustration. The data—see part (a) of the figure—consists of a single table, named CELLAR (in fact, it is a scaled-down version of the CELLAR table from Fig. 1.1, reduced in size to make it a little more manageable). Two sample retrievals—one involving a row-subsetting operation and the other a column-subsetting operation—are shown in part (b) of the figure. *Note:* Once again, the two retrievals are in fact examples of the `SELECT` statement of the language SQL first mentioned in Section 1.1.

We can now distinguish between relational and nonrelational systems, as follows. As already stated, the user of a relational system sees the data as tables, and nothing but tables. The user of a nonrelational system, by contrast, sees other data structures, either instead of or in addition to the tables of a relational system. Those other structures, in turn, require other operators to manipulate them. For example, in a **hierarchic** system, the data is represented to the user in the form of a set of tree structures (hierarchies), and the operators provided for manipulating such structures include operators for traversing hierarchic paths up and down the trees.

To pursue the point a little further: Database systems can in fact be conveniently categorized according to the data structures and operators they present to the user. First of all, older (prerelational) systems fall into three broad categories, namely **inverted**

a) Given table:		CELLAR	WINE	YEAR	BOTTLES
			Chardonnay	91	4
			Fumé Blanc	91	2
			Pinet Noir	88	3
			Zinfandel	89	9

b) Operators (examples):					
1. Row subset:	Result:	SELECT WINE, YEAR, BOTTLES FROM CELLAR WHERE YEAR > 90 ;	WINE	YEAR	BOTTLES
			Chardonnay	91	4
			Fumé Blanc	91	2
2. Column subset:	Result:	SELECT WINE, BOTTLES FROM CELLAR ;	WINE	BOTTLES	
			Chardonnay		4
			Fumé Blanc		2
			Pinot Noir		3
			Zinfandel		9

FIG. 1.8 Data structure and operators in a relational system (examples)

list, hierarchic, and network systems. Examples of commercially available products in these three categories include:

- Inverted list:* CA-DATACOM/DB, from Computer Associates International Inc. (previously known as DATACOM/DB, from Applied Data Research)
- Hierarchic:* IMS, from IBM Corporation
- Network:* CA-IDMS/DB, from Computer Associates International Inc. (previously known as IDMS, from Cullinet Software Inc.)

The first **relational** products began to appear in the late 1970s and early 1980s. At the time of writing (1993), there are well over 100—perhaps as many as 200—such products commercially available, and those products run on just about every kind of hardware and software platform imaginable. Examples of such products include DB2 from IBM Corporation; Rdb/VMS from Digital Equipment Corporation; ORACLE from Oracle Corporation; INGRES from the Ingres Division of The ASK Group Inc.; SYBASE from Sybase Inc.; and many, many more.

More recently, research has proceeded on a variety of what might be called “postrelational” systems, some of them based on upward-compatible extensions to the original relational approach, others consisting of attempts at doing something entirely different. We content ourselves for now with merely mentioning some of these more

recent approaches by name, without making any attempt at this point to explain what the names mean or what the researchers are trying to achieve:

- Deductive DBMSs
- Expert DBMSs
- Extendable DBMSs
- Object-oriented DBMSs
- Semantic DBMSs
- Universal relation DBMSs

In the case of **object-oriented** systems, in fact, some products have begun to appear, including GemStone from Servio Corporation, ObjectStore from Object Design Corporation, and OpenODB from Hewlett-Packard Corporation. We will examine some of these newer directions—object-oriented systems in particular—in later parts of the book.

1.7 Summary

We close this introductory chapter by summarizing the main points discussed. First, a **database system** can be thought of as a computerized record-keeping system. Such a system involves the **data** itself (stored in the **database**), **hardware**, **software** (in particular the **database management system** or DBMS), and—most important!—**users**. Users in turn can be divided into **application programmers**, **end users**, and the **database administrator** or DBA. The DBA is responsible for administering the database and database system in accordance with policies established by the **data administrator**.

Databases are **integrated** and (usually) **shared**; they are used to store **persistent** data. Such data can be usefully (albeit informally) considered as representing **entities**, together with **relationships** among those entities—although in fact a relationship is really just a special kind of entity. We very briefly examined the idea of **entity/relationship diagrams**.

Database systems provide a number of benefits, of which one of the most important is **data independence** (the immunity of applications to changes in the way the data is stored and accessed).

Finally, database systems can be based on a number of different approaches, including in particular the **relational** approach. From both an economic and a theoretical perspective, the relational approach is easily the most important (and this state of affairs is not likely to change in the foreseeable future). In a relational system, the data is seen by the user as **tables**, and the operators available to the user for dealing with the data are operators that manipulate tables. We have seen a few simple examples of **SQL**, the standard language for dealing with relational systems. This book will be heavily based on the relational approach, although *not*—for reasons explained in the Preface—on *SQL per se*.

Exercises

1.1 Define the following terms:

binary relationship	menu-driven interface
command-driven interface	multi-user system
concurrent access	online application
data administration	persistent data
database	property
database system	query language
data independence	redundancy
DBA	relationship
DBMS	security
entity	sharing
entity/relationship diagram	stored field
forms-driven interface	stored file
integration	stored record
integrity	

1.2 What are the advantages of using a database system?

1.3 What are the disadvantages of using a database system?

1.4 What do you understand by the term "relational system"? Distinguish between relational and nonrelational systems.

1.5 Show the effects of the following SQL retrieval operations on the wine cellar database of Fig. 1.1.

- (a)

```
SELECT WINE, PRODUCER
FROM   CELLAR
WHERE  BIN = 72 ;
```
- (b)

```
SELECT WINE, PRODUCER
FROM   CELLAR
WHERE  YEAR > 91 ;
```
- (c)

```
SELECT BIN, WINE, YEAR
FROM   CELLAR
WHERE  READY < 94 ;
```
- (d)

```
SELECT WINE, BIN, YEAR
FROM   CELLAR
WHERE  PRODUCER = 'Robt. Mondavi'
AND    BOTTLES > 6 ;
```

1.6 Show the effects of the following SQL update operations on the wine cellar database of Fig. 1.1.

- (a)

```
INSERT
INTO   CELLAR ( BIN, WINE, PRODUCER, YEAR, BOTTLES, READY )
VALUES ( 80, 'Syrah', 'Meridian', 89, 12, 94 ) ;
```
- (b)

```
DELETE
FROM   CELLAR
WHERE  READY > 95 ;
```
- (c)

```
UPDATE CELLAR
SET    BOTTLES = 5
WHERE  BIN = 50 ;
```

```
(d) UPDATE CELLAR
    SET    BOTTLES = BOTTLES + 2
    WHERE  BIN = 50 ;
```

1.7 Write SQL statements to perform the following operations on the wine cellar database:

- Retrieve bin number, name of wine, and number of bottles for all Geyser Peak wines.
- Retrieve bin number and name of wine for all wines for which there are more than five bottles in stock.
- Retrieve bin number for all red wines.
- Add three bottles to bin number 30.
- Remove all Chardonnay from stock.
- Add an entry for a new case (12 bottles) of Gary Farrell Merlot: bin number 55, year 91, ready in 96.

1.8 Suppose you have a classical music collection consisting of CDs and/or LPs and/or audio tapes, and you want to build a database that will let you find which recordings you have for a specific composer (e.g., Sibelius) or conductor (e.g., Simon Rattle) or soloist (e.g., Arthur Grumiaux) or work (e.g., Beethoven's Fifth) or orchestra (e.g., the NYPO) or kind of work (e.g., violin concerto) or chamber group (e.g., the Kronos Quartet). Draw an entity/relationship diagram like that of Fig. 1.6 for this database.

Answers to Selected Exercises

1.1 We make one remark here: The trade press, sales brochures, etc., very frequently use the term *database* when they really mean *DBMS* (e.g., "vendor X's database outperformed vendor Y's database by a factor of two to one"). This usage is sloppy, and deprecated, but very, very common. *Caveat lector*.

1.3 Some disadvantages are as follows:

- Security might be compromised (without good controls)
- Integrity might be compromised (without good controls)
- Additional hardware might be required
- Performance overhead might be significant
- Successful operation is crucial (the enterprise might be highly vulnerable to failure)
- The system is likely to be complex (though such complexity should be concealed from the user)

1.5 (a)

WINE	PRODUCER
Zinfandel	Rafanelli

(b)

WINE	PRODUCER
Chardonnay	Buena Vista
Chardonnay	Geyser Peak
Jo. Riesling	Jekel
Pumé Blanc	Ch. St. Jean
Gewürstraminer	Ch. St. Jean

(c)

BIN	WINE	YEAR
6	Chardonnay	91
22	Fumé Blanc	91
52	Pinot Noir	90

(d)

WINE	BIN	YEAR
Cab. Sauvignon	48	88

- 1.6 (a) Row for bin 80 added to the CELLAR table.
 (b) Rows for bins 45, 48, 64, and 72 deleted from the CELLAR table.
 (c) Row for bin 50 has number of bottles set to 5.
 (d) Same as (c).

1.7 (a) SELECT BIN, WINE, BOTTLES
 FROM CELLAR
 WHERE PRODUCER = 'Geyser Peak' ;

(b) SELECT BIN, WINE
 FROM CELLAR
 WHERE BOTTLES > 5 ;

(c) SELECT BIN
 FROM CELLAR
 WHERE WINE = 'Cab. Sauvignon'
 OR WINE = 'Pinot Noir'
 OR WINE = 'Zinfandel'
 OR WINE = 'Syrah'
 OR ;

There is no shortcut answer to this question, because "color of wine" is not explicitly recorded in the database.

(d) UPDATE CELLAR
 SET BOTTLES = BOTTLES + 3
 WHERE BIN = 30 ;

(e) DELETE
 FROM CELLAR
 WHERE WINE = 'Chardonnay' ;

(f) INSERT
 INTO CELLAR (BIN, WINE, PRODUCER, YEAR, BOTTLES, READY)
 VALUES (55, 'Merlot', 'Gary Farrell', 91, 12, 96) ;