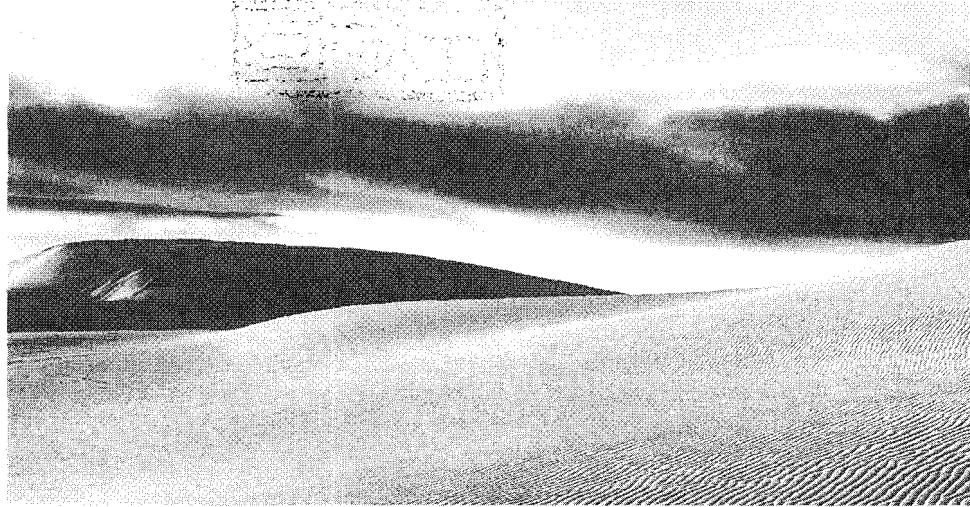


FUNDAMENTALS OF

Fourth Edition

DATABASE SYSTEMS

FUNDAMENTALS OF *Fourth Edition* DATABASE SYSTEMS



Ramez Elmasri

*Department of Computer Science Engineering
University of Texas at Arlington*

Shamkant B. Navathe

*College of Computing
Georgia Institute of Technology*



Boston San Francisco New York
London Toronto Sydney Tokyo Singapore Madrid
Mexico City Munich Paris Cape Town Hong Kong Montreal

Sponsoring Editor:	Maite Suarez-Rivas
Project Editor:	Katherine Harutunian
Senior Production Supervisor:	Juliet Silveri
Production Services:	Argosy Publishing
Cover Designer:	Beth Anderson
Marketing Manager:	Nathan Schultz
Senior Marketing Coordinator:	Lesly Hershman
Print Buyer:	Caroline Fell

Cover image © 2003 Digital Vision

Access the latest information about Addison-Wesley titles from our World Wide Web site:
<http://www.aw.com/cs>

Figure 12.14 is a logical data model diagram definition in Rational Rose[®]. Figure 12.15 is a graphical data model diagram in Rational Rose[®]. Figure 12.17 is the company database class diagram drawn in Rational Rose[®]. IBM[®] has acquired Rational Rose[®].

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

Elmasri, Ramez.

Fundamentals of database systems / Ramez Elmasri, Shamkant B.

Navathe.—4th ed.

p. cm.

Includes bibliographical references and index.

ISBN 0-321-12226-7

1. Database management. I. Navathe, Sham. II. Title.

QA76.9.D3E57 2003

005.74--dc21

2003057734

ISBN 0-321-12226-7

For information on obtaining permission for the use of material from this work, please submit a written request to Pearson Education, Inc., Rights and Contracts Department, 75 Arlington St., Suite 300, Boston, MA 02116 or fax your request to 617-848-7047.

Copyright © 2004 by Pearson Education, 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.

1 2 3 4 5 6 7 8 9 10—HT—06050403

To Amalia with love
R. E.

*To my mother Vijaya and wife Aruna
for their love and support*
S. B. N.

Preface

This book introduces the fundamental concepts necessary for designing, using, and implementing database systems and applications. Our presentations stresses the fundamentals of database modeling and design, the languages and facilities provided by the database management systems, and system implementation techniques. The book is meant to be used as a textbook for a one- or two-semester course in database systems at the junior, senior or graduate level, and as a reference book. We assume that the readers are familiar with elementary programming and data-structuring concepts and that they have had some exposure to the basic computer organization.

We start in Part 1 with an introduction and a presentation of the basic concepts and terminology, and database conceptual modeling principles. We conclude the book in Parts 7 and 8 with an introduction to emerging technologies, such as data mining, XML, security, and Web databases. Along the way—in Parts 2 through 6—we provide an in-depth treatment of the most important aspects of database fundamentals.

The following key features are included in the fourth edition:

- The entire book follows a self-contained, flexible organization that can be tailored to individual needs.
- Coverage of data modeling now includes both the ER model and UML.
- A new advanced SQL chapter with material on SQL programming techniques, such as JDBC and SQL/CLI.

- Two examples running throughout the book—called COMPANY and UNIVERSITY—allow the reader to compare different approaches that use the same application.
- Coverage has been updated on security, mobile databases, GIS, and Genome data management.
- A new chapter on XML and Internet databases.
- A new chapter on data mining.
- A significant revision of the supplements to include a robust set of materials for instructors and students, and an online case study.

Main Differences from the Third Edition

There are several organizational changes in the fourth edition, as well as some important new chapters. The main changes are as follows:

- The chapters on file organizations and indexing (Chapters 5 and 6 in the third edition) have been moved to Part 4, and are now Chapters 13 and 14. Part 4 also includes Chapters 15 and 16 on query processing and optimization, and physical database design and tuning (this corresponds to Chapter 18 and sections 16.3-16.4 of the third edition).
- The relational model coverage has been reorganized and updated in Part 2. Chapter 5 covers relational model concepts and constraints. The material on relational algebra and calculus is now together in Chapter 6. Relational database design using ER-to-relational and EER-to-relational mapping is in Chapter 7. SQL is covered in Chapters 8 and 9, with the new material in SQL programming techniques in sections 9.3 through 9.6.
- Part 3 covers database design theory and methodology. Chapters 10 and 11 on normalization theory correspond to Chapters 14 and 15 of the third edition. Chapter 12 on practical database design has been updated to include more UML coverage.
- The chapters on transactions, concurrency control, and recovery (19, 20, 21 in the third edition) are now Chapters 17, 18, and 19 in Part 5.
- The chapters on object-oriented concepts, ODMG object model, and object-relational systems (11, 12, 13 in the third edition) are now 20, 21, and 22 in Part 6. Chapter 22 has been reorganized and updated.
- Chapters 10 and 17 of the third edition have been dropped. The material on client-server architectures has been merged into Chapters 2 and 25.
- The chapters on security, enhanced models (active, temporal, spatial, multimedia), and distributed databases (Chapters 22, 23, 24 in the third edition) are now 23, 24, and 25 in Part 7. The security chapter has been updated. Chapter 25 of the third edition on deductive databases has been merged into Chapter 24, and is now section 24.4.

- Chapter 26 is a new chapter on XML (eXtended Markup Language), and how it is related to accessing relational databases over the Internet.
- The material on data mining and data warehousing (Chapter 26 of the third edition) has been separated into two chapters. Chapter 27 on data mining has been expanded and updated.

Contents of This Edition

Part 1 describes the basic concepts necessary for a good understanding of database design and implementation, as well as the conceptual modeling techniques used in database systems. Chapters 1 and 2 introduce databases, their typical users, and DBMS concepts, terminology, and architecture. In Chapter 3, the concepts of the Entity-Relationship (ER) model and ER diagrams are presented and used to illustrate conceptual database design. Chapter 4 focuses on data abstraction and semantic data modeling concepts and extends the ER model to incorporate these ideas, leading to the enhanced-ER (EER) data model and EER diagrams. The concepts presented include subclasses, specialization, generalization, and union types (categories). The notation for the class diagrams of UML are also introduced in Chapters 3 and 4.

Part 2 describes the relational data model and relational DBMSs. Chapter 5 describes the basic relational model, its integrity constraints and update operations. Chapter 6 describes the operations of the relational algebra and introduces the relational calculus. Chapter 7 discusses relational database design using ER and EER-to-relational mapping. Chapter 8 gives a detailed overview of the SQL language, covering the SQL standard, which is implemented in most relational systems. Chapter 9 covers SQL programming topics such as SQLJ, JDBC, and SQL/CLI.

Part 3 covers several topics related to database design. Chapters 10 and 11 cover the formalisms, theories, and algorithms developed for the relational database design by normalization. This material includes functional and other types of dependencies and normal forms of relations. Step-by-step intuitive normalization is presented in Chapter 10, and relational design algorithms are given in Chapter 11, which also defines other types of dependencies, such as multivalued and join dependencies. Chapter 12 presents an overview of the different phases of the database design process for medium-sized and large applications, using UML.

Part 4 starts with a description of the physical file structures and access methods used in database systems. Chapter 13 describes primary methods of organizing files of records on disk, including static and dynamic hashing. Chapter 14 describes indexing techniques for files, including B-tree and B+-tree data structures and grid files. Chapter 15 introduces the basics of query processing and optimization, and Chapter 16 discusses physical database design and tuning.

Part 5 discusses transaction processing, concurrency control, and recovery techniques, including discussions of how these concepts are realized in SQL.

Part 6 gives a comprehensive introduction to object databases and object-relational systems. Chapter 20 introduces object-oriented concepts. Chapter 21 gives a detailed overview of the ODMG object model and its associated ODL and OQL languages. Chapter 22 describes how relational databases are being extended to include object-oriented concepts and presents the features of object-relational systems, as well as giving an overview of some of the features of the SQL3 standard, and the nested relational data model.

Parts 7 and 8 cover a number of advanced topics. Chapter 23 gives an overview of database security and authorization, including the SQL commands to GRANT and REVOKE privileges, and expanded coverage on security concepts such as encryption, roles, and flow control. Chapter 24 introduces several enhanced database models for advanced applications. These include active databases and triggers, temporal, spatial, multimedia, and deductive databases. Chapter 25 gives an introduction to distributed databases and the three-tier client-server architecture. Chapter 26 is a new chapter on XML (eXtended Markup Language). It first discusses the differences between structured, semi-structured, and unstructured models, then presents XML concepts, and finally compares the XML model to traditional database models. Chapter 27 on data mining has been expanded and updated. Chapter 28 introduces data warehousing concepts. Finally, Chapter 29 gives introductions to the topics of mobile databases, multimedia databases, GIS (Geographic Information Systems), and Genome data management in bioinformatics.

Appendix A gives a number of alternative diagrammatic notations for displaying a conceptual ER or EER schema. These may be substituted for the notation we use, if the instructor so wishes. Appendix C gives some important physical parameters of disks. Appendixes B, E, and F are on the web site. Appendix B is a new case study that follows the design and implementation of a bookstore's database. Appendixes E and F cover legacy database systems, based on the network and hierarchical database models. These have been used for over thirty years as a basis for many existing commercial database applications and transaction-processing systems and will take decades to replace completely. We consider it important to expose students of database management to these long-standing approaches. Full chapters from the third edition can be found on the web site for this edition.

Guidelines for Using This Book

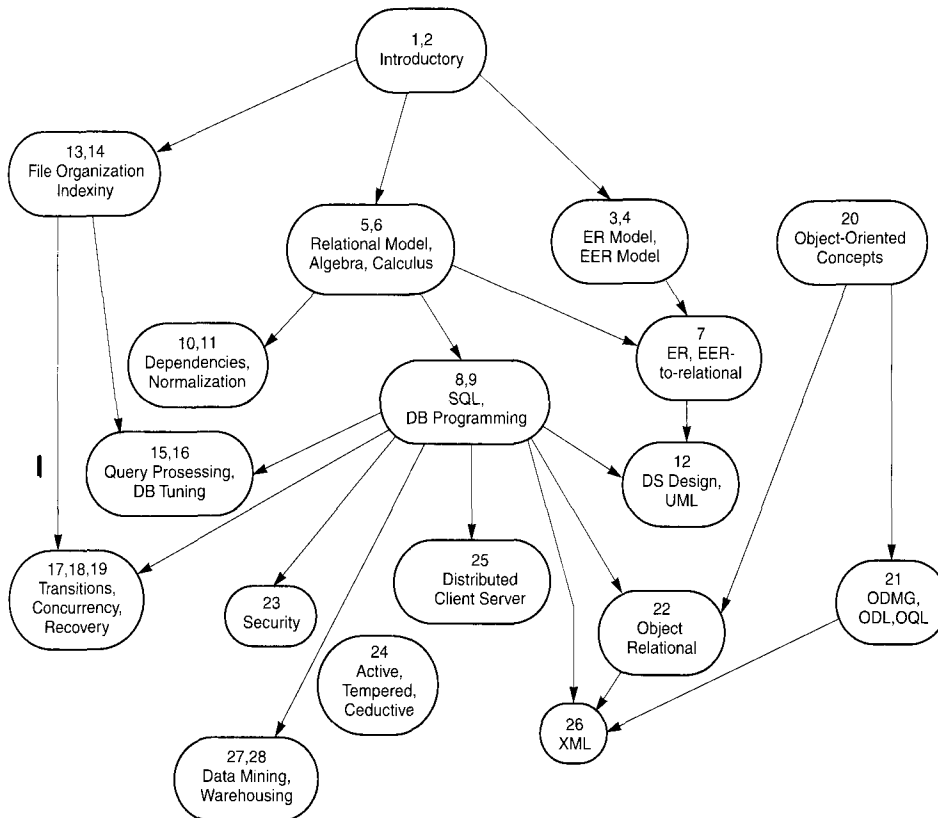
There are many different ways to teach a database course. The chapters in Parts 1 through 5 can be used in an introductory course on database systems in the order that they are given or in the preferred order of each individual instructor. Selected chapters and sections may be left out, and the instructor can add other chapters from the rest of the book, depending on the emphasis of the course. At the end of each chapter's opening section, we list sections that are candidates for being left out whenever a less detailed discussion of the topic in a particular chapter is desired. We suggest covering up to Chapter 14 in an introductory database course and including selected parts of other chapters, depending on the background of the students and the desired coverage. For an emphasis on system implementation techniques, chapters from Parts 4 and 5 can be included.

Chapters 3 and 4, which cover conceptual modeling using the ER and EER models, are important for a good conceptual understanding of databases. However, they may be par-

tially covered, covered later in a course, or even left out if the emphasis is on DBMS implementation. Chapters 13 and 14 on file organizations and indexing may also be covered early on, later, or even left out if the emphasis is on database models and languages. For students who have already taken a course on file organization, parts of these chapters could be assigned as reading material or some exercises may be assigned to review the concepts.

A total life-cycle database design and implementation project covers conceptual design (Chapters 3 and 4), data model mapping (Chapter 7), normalization (Chapter 10), and implementation in SQL (Chapter 9). Additional documentation on the specific RDBMS would be required.

The book has been written so that it is possible to cover topics in a variety of orders. The chart included here shows the major dependencies between chapters. As the diagram illustrates, it is possible to start with several different topics following the first two introductory chapters. Although the chart may seem complex, it is important to note that if the chapters are covered in order, the dependencies are not lost. The chart can be consulted by instructors wishing to use an alternative order of presentation.



For a single-semester course based on this book, some chapters can be assigned as reading material. Parts 4, 7, and 8 can be considered for such an assignment. The book can also

be used for a two-semester sequence. The first course, “Introduction to Database Design/Systems,” at the sophomore, junior, or senior level, could cover most of Chapters 1 to 14. The second course, “Database Design and Implementation Techniques,” at the senior or first-year graduate level, can cover Chapters 15 to 28. Chapters from Parts 7 and 8 can be used selectively in either semester, and material describing the DBMS available to the students at the local institution can be covered in addition to the material in the book.

Supplemental Materials

The supplements to this book have been significantly revised. With Addison-Wesley’s Database Place there is a robust set of interactive reference materials to help students with their study of modeling, normalization, and SQL. Each tutorial asks students to solve problems (such as writing an SQL query, drawing an ER diagram or normalizing a relation), and then provides useful feedback based on the student’s solution. Addison-Wesley’s Database Place helps students master the key concepts of all database courses. For more information visit aw.com/databaseplace.

In addition the following supplements are available to all readers of this book at www.aw.com/cssupport.

- Additional content: This includes a new Case Study on the design and implementation of a bookstore’s database as well as chapters from previous editions that are not included in the fourth edition.
- A set of PowerPoint lecture notes

A solutions manual is also available to qualified instructors. Please contact your local Addison-Wesley sales representative, or send e-mail to aw.cse@aw.com, for information on how to access it.

Acknowledgements

It is a great pleasure for us to acknowledge the assistance and contributions of a large number of individuals to this effort. First, we would like to thank our editors, Maite Suarez-Rivas, Katherine Harutunian, Daniel Rausch, and Juliet Silveri. In particular we would like to acknowledge the efforts and help of Katherine Harutunian, our primary contact for the fourth edition. We would like to acknowledge also those persons who have contributed to the fourth edition. We appreciated the contributions of the following reviewers: Phil Bernhard, *Florida Tech*; Zhengxin Chen, *University of Nebraska at Omaha*; Jan Chomicki, *University of Buffalo*; Hakan Ferhatosmanoglu, *Ohio State University*; Len Fisk, *California State University, Chico*; William Hankley, *Kansas State University*; Ali R. Hurson, *Penn State University*; Vijay Kumar, *University of Missouri-Kansas City*; Peretz Shoval, *Ben-Gurion University, Israel*; Jason T. L. Wang, *New Jersey Institute of Technology*; and Ed Omiecinski of *Georgia Tech*, who contributed to Chapter 27.

Ramez Elmasri would like to thank his students Hyoil Han, Babak Hojabri, Jack Fu, Charley Li, Ande Swathi, and Steven Wu, who contributed to the material in Chapter

26. He would also like to acknowledge the support provided by the University of Texas at Arlington.

Sham Navathe would like to acknowledge Dan Forsythe and the following students at Georgia Tech: Weimin Feng, Angshuman Guin, Abrar Ul-Haque, Bin Liu, Ying Liu, Wanxia Xie and Waigen Yee.

We would like to repeat our thanks to those who have reviewed and contributed to previous editions of *Fundamentals of Database Systems*. For the first edition these individuals include Alan Apt (editor), Don Batory, Scott Downing, Dennis Heimbinger, Julia Hodges, Yannis Ioannidis, Jim Larson, Dennis McLeod, Per-Ake Larson, Rahul Patel, Nicholas Roussopoulos, David Stemple, Michael Stonebraker, Frank Tompa, and Kyu-Young Whang; for the second edition they include Dan Joraanstad (editor), Rafi Ahmed, Antonio Albano, David Beech, Jose Blakeley, Panos Chrysanthis, Suzanne Dietrich, Vic Ghorpadey, Goets Graefe, Eric Hanson, Junguk L. Kim, Roger King, Vram Kouramajian, Vijay Kumar, John Lowther, Sanjay Manchanda, Toshimi Minoura, Inderpal Mumick, Ed Omiecinski, Girish Pathak, Raghu Ramakrishnan, Ed Robertson, Eugene Sheng, David Stotts, Marianne Winslett, and Stan Zdonick. For the third edition they include Suzanne Dietrich, Ed Omiecinski, Rafi Ahmed, Francois Bancilhon, Jose Blakeley, Rick Cattell, Ann Chervenak, David W. Embley, Henry A. Etlinger, Leonidas Fegaras, Dan Forsyth, Farshad Fotouhi, Michael Franklin, Sreejith Gopinath, Goetz Craefe, Richard Hull, Sushil Jajodia, Ramesh K. Karne, Harish Kotbagi, Vijay Kumar, Tarcisio Lima, Ramon A. Mata-Toledo, Jack McCaw, Dennis McLeod, Rokia Missaoui, Magdi Morsi, M. Narayanaswamy, Carlos Ordonez, Joan Peckham, Betty Salzberg, Ming-Chien Shan, Junping Sun, Rajshekhar Sunderraman, Aravindan Veerasamy, and Emilia E. Villareal.

Last but not least, we gratefully acknowledge the support, encouragement, and patience of our families.

R.E.
S.B.N.

Contents

PART 1 INTRODUCTION AND CONCEPTUAL MODELING

CHAPTER 1 Databases and Database Users	3
1.1 Introduction	4
1.2 An Example	6
1.3 Characteristics of the Database Approach	8
1.4 Actors on the Scene	12
1.5 Workers behind the Scene	14
1.6 Advantages of Using the DBMS Approach	15
1.7 A Brief History of Database Applications	20
1.8 When Not to Use a DBMS	23
1.9 Summary	23
Review Questions	23
Exercises	24
Selected Bibliography	24

CHAPTER 2 Database System Concepts and Architecture 25

- 2.1 Data Models, Schemas, and Instances 26
- 2.2 Three-Schema Architecture and Data Independence 29
- 2.3 Database Languages and Interfaces 32
- 2.4 The Database System Environment 35
- 2.5 Centralized and Client/Server Architectures for DBMSs 38
- 2.6 Classification of Database Management Systems 43
- 2.7 Summary 45
 - Review Questions 46
 - Exercises 46
 - Selected Bibliography 47

CHAPTER 3 Data Modeling Using the Entity-Relationship Model 49

- 3.1 Using High-Level Conceptual Data Models for Database Design 50
- 3.2 An Example Database Application 52
- 3.3 Entity Types, Entity Sets, Attributes, and Keys 53
- 3.4 Relationship Types, Relationship Sets, Roles, and Structural Constraints 61
- 3.5 Weak Entity Types 68
- 3.6 Refining the ER Design for the COMPANY Database 69
- 3.7 ER Diagrams, Naming Conventions, and Design Issues 70
- 3.8 Notation for UML Class Diagrams 74
- 3.9 Summary 77
 - Review Questions 78
 - Exercises 78
 - Selected Bibliography 83

CHAPTER 4 Enhanced Entity-Relationship and UML Modeling 85

- 4.1 Subclasses, Superclasses, and Inheritance 86
- 4.2 Specialization and Generalization 88
- 4.3 Constraints and Characteristics of Specialization and Generalization 91
- 4.4 Modeling of UNION Types Using Categories 98
- 4.5 An Example UNIVERSITY EER Schema and Formal Definitions for the EER Model 101

4.6	Representing Specialization/Generalization and Inheritance in UML Class Diagrams	104
4.7	Relationship Types of Degree Higher Than Two	105
4.8	Data Abstraction, Knowledge Representation, and Ontology Concepts	110
4.9	Summary	115
	Review Questions	116
	Exercises	117
	Selected Bibliography	121

PART 2 RELATIONAL MODEL: CONCEPTS, CONSTRAINTS, LANGUAGES, DESIGN, AND PROGRAMMING

CHAPTER 5 The Relational Data Model and Relational Database Constraints 125

5.1	Relational Model Concepts	126
5.2	Relational Model Constraints and Relational Database Schemas	132
5.3	Update Operations and Dealing with Constraint Violations	140
5.4	Summary	143
	Review Questions	144
	Exercises	144
	Selected Bibliography	147

CHAPTER 6 The Relational Algebra and Relational Calculus 149

6.1	Unary Relational Operations: SELECT and PROJECT	151
6.2	Relational Algebra Operations from Set Theory	155
6.3	Binary Relational Operations: JOIN and DIVISION	158
6.4	Additional Relational Operations	165
6.5	Examples of Queries in Relational Algebra	171
6.6	The Tuple Relational Calculus	173
6.7	The Domain Relational Calculus	181
6.8	Summary	184
	Review Questions	185
	Exercises	186
	Selected Bibliography	189

CHAPTER 7 Relational Database Design by ER- and EER-to-Relational Mapping	191
7.1 Relational Database Design Using ER-to-Relational Mapping	192
7.2 Mapping EER Model Constructs to Relations	199
7.3 Summary	203
Review Questions	204
Exercises	204
Selected Bibliography	205
CHAPTER 8 SQL-99: Schema Definition, Basic Constraints, and Queries	207
8.1 SQL Data Definition and Data Types	209
8.2 Specifying Basic Constraints in SQL	213
8.3 Schema Change Statements in SQL	217
8.4 Basic Queries in SQL	218
8.5 More Complex SQL Queries	229
8.6 Insert, Delete, and Update Statements in SQL	245
8.7 Additional Features of SQL	248
8.8 Summary	249
Review Questions	251
Exercises	251
Selected Bibliography	252
CHAPTER 9 More SQL: Assertions, Views, and Programming Techniques	255
9.1 Specifying General Constraints as Assertions	256
9.2 Views (Virtual Tables) in SQL	257
9.3 Database Programming: Issues and Techniques	261
9.4 Embedded SQL, Dynamic SQL, and SQLJ	264
9.5 Database Programming with Function Calls: SQL/CLI and JDBC	275
9.6 Database Stored Procedures and SQL/PSM	284
9.7 Summary	287
Review Questions	287
Exercises	287
Selected Bibliography	289

PART 3 DATABASE DESIGN THEORY AND METHODOLOGY**CHAPTER 10 Functional Dependencies and
Normalization for Relational Databases 293**

10.1	Informal Design Guidelines for Relation Schemas	295
10.2	Functional Dependencies	304
10.3	Normal Forms Based on Primary Keys	312
10.4	General Definitions of Second and Third Normal Forms	320
10.5	Boyce-Codd Normal Form	324
10.6	Summary	326
	Review Questions	327
	Exercises	328
	Selected Bibliography	331

**CHAPTER 11 Relational Database Design
Algorithms and Further Dependencies 333**

11.1	Properties of Relational Decompositions	334
11.2	Algorithms for Relational Database Schema Design	340
11.3	Multivalued Dependencies and Fourth Normal Form	347
11.4	Join Dependencies and Fifth Normal Form	353
11.5	Inclusion Dependencies	354
11.6	Other Dependencies and Normal Forms	355
11.7	Summary	357
	Review Questions	358
	Exercises	358
	Selected Bibliography	360

**CHAPTER 12 Practical Database Design Methodology
and Use of UML Diagrams 361**

12.1	The Role of Information Systems in Organizations	362
12.2	The Database Design and Implementation Process	366
12.3	Use of UML Diagrams as an Aid to Database Design Specification	385
12.4	Rational Rose, A UML Based Design Tool	395
12.5	Automated Database Design Tools	402
12.6	Summary	404
	Review Questions	405
	Selected Bibliography	406

PART 4 DATA STORAGE, INDEXING, QUERY PROCESSING, AND PHYSICAL DESIGN

CHAPTER 13 Disk Storage, Basic File Structures, and Hashing 411

13.1	Introduction	412	
13.2	Secondary Storage Devices	415	
13.3	Buffering of Blocks	421	
13.4	Placing File Records on Disk	422	
13.5	Operations on Files	427	
13.6	Files of Unordered Records (Heap Files)	430	
13.7	Files of Ordered Records (Sorted Files)	431	
13.8	Hashing Techniques	434	
13.9	Other Primary File Organizations	442	
13.10	Parallelizing Disk Access Using RAID Technology	443	
13.11	Storage Area Networks	447	
13.12	Summary	449	
	Review Questions	450	
	Exercises	451	
	Selected Bibliography	454	

CHAPTER 14 Indexing Structures for Files 455

14.1	Types of Single-Level Ordered Indexes	456	
14.2	Multilevel Indexes	464	
14.3	Dynamic Multilevel Indexes Using B-Trees and B ⁺ -Trees	469	
14.4	Indexes on Multiple Keys	483	
14.5	Other Types of Indexes	485	
14.6	Summary	486	
	Review Questions	487	
	Exercises	488	
	Selected Bibliography	490	

CHAPTER 15 Algorithms for Query Processing and Optimization 493

15.1	Translating SQL Queries into Relational Algebra	495	
15.2	Algorithms for External Sorting	496	
15.3	Algorithms for SELECT and JOIN Operations	498	
15.4	Algorithms for PROJECT and SET Operations	508	

15.5	Implementing Aggregate Operations and Outer Joins	509
15.6	Combining Operations Using Pipelining	511
15.7	Using Heuristics in Query Optimization	512
15.8	Using Selectivity and Cost Estimates in Query Optimization	523
15.9	Overview of Query Optimization in ORACLE	532
15.10	Semantic Query Optimization	533
15.11	Summary	534
	Review Questions	534
	Exercises	535
	Selected Bibliography	536

CHAPTER 16 Practical Database Design and Tuning 537

16.1	Physical Database Design in Relational Databases	537
16.2	An Overview of Database Tuning in Relational Systems	541
16.3	Summary	547
	Review Questions	547
	Selected Bibliography	548

PART 5 TRANSACTION PROCESSING CONCEPTS

CHAPTER 17 Introduction to Transaction Processing Concepts and Theory 551

17.1	Introduction to Transaction Processing	552
17.2	Transaction and System Concepts	559
17.3	Desirable Properties of Transactions	562
17.4	Characterizing Schedules Based on Recoverability	563
17.5	Characterizing Schedules Based on Serializability	566
17.6	Transaction Support in SQL	576
17.7	Summary	578
	Review Questions	579
	Exercises	580
	Selected Bibliography	581

CHAPTER 18 Concurrency Control Techniques 583

18.1	Two-Phase Locking Techniques for Concurrency Control	584
18.2	Concurrency Control Based on Timestamp Ordering	594
18.3	Multiversion Concurrency Control Techniques	596
18.4	Validation (Optimistic) Concurrency Control Techniques	599

18.5	Granularity of Data Items and Multiple Granularity Locking	600
18.6	Using Locks for Concurrency Control in Indexes	605
18.7	Other Concurrency Control Issues	606
18.8	Summary	607
	Review Questions	608
	Exercises	609
	Selected Bibliography	609

CHAPTER 19 Database Recovery Techniques 611

19.1	Recovery Concepts	612
19.2	Recovery Techniques Based on Deferred Update	618
19.3	Recovery Techniques Based on Immediate Update	622
19.4	Shadow Paging	624
19.5	The ARIES Recovery Algorithm	625
19.6	Recovery in Multidatabase Systems	629
19.7	Database Backup and Recovery from Catastrophic Failures	630
19.8	Summary	631
	Review Questions	632
	Exercises	633
	Selected Bibliography	635

PART 6 OBJECT AND OBJECT-RELATIONAL DATABASES

CHAPTER 20 Concepts for Object Databases 639

20.1	Overview of Object-Oriented Concepts	641
20.2	Object Identity, Object Structure, and Type Constructors	643
20.3	Encapsulation of Operations, Methods, and Persistence	649
20.4	Type and Class Hierarchies and Inheritance	654
20.5	Complex Objects	657
20.6	Other Objected-Oriented Concepts	659
20.7	Summary	662
	Review Questions	663
	Exercises	664
	Selected Bibliography	664

CHAPTER 21 Object Database Standards, Languages, and Design 665

21.1	Overview of the Object Model of ODMG	666
------	--------------------------------------	-----

21.2	The Object Definition Language ODL	679
21.3	The Object Query Language OQL	684
21.4	Overview of the C++ Language Binding	693
21.5	Object Database Conceptual Design	694
21.6	Summary	697
	Review Questions	698
	Exercises	698
	Selected Bibliography	699

CHAPTER 22 Object-Relational and Extended-Relational Systems 701

22.1	Overview of SQL and Its Object-Relational Features	702
22.2	Evolution and Current Trends of Database Technology	709
22.3	The Informix Universal Server	711
22.4	Object-Relational Features of Oracle 8	721
22.5	Implementation and Related Issues for Extended Type Systems	724
22.6	The Nested Relational Model	725
22.7	Summary	727
	Selected Bibliography	728

PART 7 FURTHER TOPICS

CHAPTER 23 Database Security and Authorization 731

23.1	Introduction to Database Security Issues	732
23.2	Discretionary Access Control Based on Granting and Revoking Privileges	735
23.3	Mandatory Access Control and Role-Based Access Control for Multilevel Security	740
23.4	Introduction to Statistical Database Security	746
23.5	Introduction to Flow Control	747
23.6	Encryption and Public Key Infrastructures	749
23.7	Summary	751
	Review Questions	752
	Exercises	753
	Selected Bibliography	753

CHAPTER 24 Enhanced Data Models for Advanced Applications 755

24.1	Active Database Concepts and Triggers	757
24.2	Temporal Database Concepts	767
24.3	Multimedia Databases	780
24.4	Introduction to Deductive Databases	784
24.5	Summary	797
	Review Questions	797
	Exercises	798
	Selected Bibliography	801

CHAPTER 25 Distributed Databases and Client–Server Architectures 803

25.1	Distributed Database Concepts	804
25.2	Data Fragmentation, Replication, and Allocation Techniques for Distributed Database Design	810
25.3	Types of Distributed Database Systems	815
25.4	Query Processing in Distributed Databases	818
25.5	Overview of Concurrency Control and Recovery in Distributed Databases	824
25.6	An Overview of 3-Tier Client-Server Architecture	827
25.7	Distributed Databases in Oracle	830
25.8	Summary	832
	Review Questions	833
	Exercises	834
	Selected Bibliography	835

PART 8 EMERGING TECHNOLOGIES

CHAPTER 26 XML and Internet Databases 841

26.1	Structured, Semistructured, and Unstructured Data	842
26.2	XML Hierarchical (Tree) Data Model	846
26.3	XML Documents, DTD, and XML Schema	848
26.4	XML Documents and Databases	855
26.5	XML Querying	862
26.6	Summary	865
	Review Questions	865
	Exercises	866
	Selected Bibliography	866

CHAPTER 27 Data Mining Concepts	867
27.1 Overview of Data Mining Technology	868
27.2 Association Rules	871
27.3 Classification	882
27.4 Clustering	885
27.5 Approaches to Other Data Mining Problems	888
27.6 Applications of Data Mining	891
27.7 Commercial Data Mining Tools	891
27.8 Summary	894
Review Questions	894
Exercises	895
Selected Bibliography	896

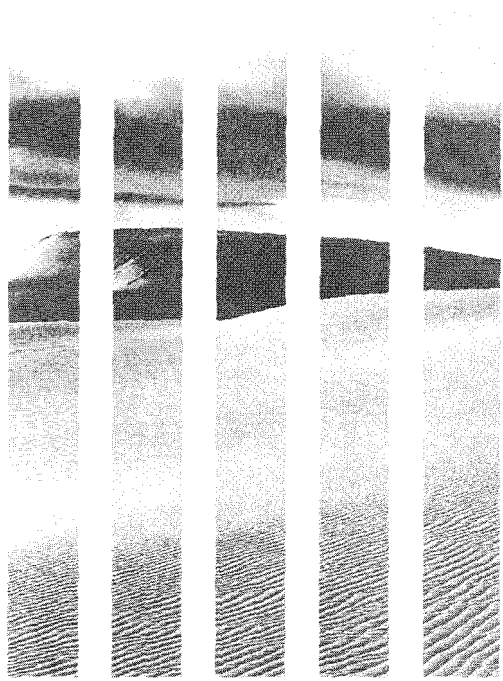
CHAPTER 28 Overview of Data Warehousing and OLAP **899**

28.1 Introduction, Definitions, and Terminology	900
28.2 Characteristics of Data Warehouses	901
28.3 Data Modeling for Data Warehouses	902
28.4 Building a Data Warehouse	907
28.5 Typical Functionality of a Data Warehouse	910
28.6 Data Warehouse Versus Views	911
28.7 Problems and Open Issues in Data Warehouses	912
28.8 Summary	913
Review Questions	914
Selected Bibliography	914

CHAPTER 29 Emerging Database Technologies and Applications **915**

29.1 Mobile Databases	916
29.2 Multimedia Databases	923
29.3 Geographic Information Systems	930
29.4 Genome Data Management	936

APPENDIX A	Alternative Diagrammatic Notations	947
APPENDIX B	Database Design and Application Implementation Case Study— <i>located on the web</i>	
APPENDIX C	Parameters of Disks	951
APPENDIX D	Overview of the QBE Language	955
APPENDIX E	Hierarchical Data Model— <i>located on the web</i>	
APPENDIX F	Network Data Model— <i>located on the web</i>	
	Selected Bibliography	963
	Index	1009



1

INTRODUCTION AND CONCEPTUAL MODELING



1

Databases and Database Users

Databases and database systems have become an essential component of everyday life in modern society. In the course of a day, most of us encounter several activities that involve some interaction with a database. For example, if we go to the bank to deposit or withdraw funds, if we make a hotel or airline reservation, if we access a computerized library catalog to search for a bibliographic item, or if we buy some item—such as a book, toy, or computer—from an Internet vendor through its Web page, chances are that our activities will involve someone or some computer program accessing a database. Even purchasing items from a supermarket nowadays in many cases involves an automatic update of the database that keeps the inventory of supermarket items.

These interactions are examples of what we may call **traditional database applications**, in which most of the information that is stored and accessed is either textual or numeric. In the past few years, advances in technology have been leading to exciting new applications of database systems. **Multimedia databases** can now store pictures, video clips, and sound messages. **Geographic information systems (GIS)** can store and analyze maps, weather data, and satellite images. **Data warehouses** and **online analytical processing (OLAP)** systems are used in many companies to extract and analyze useful information from very large databases for decision making. **Real-time and active database technology** is used in controlling industrial and manufacturing processes. And database search techniques are being applied to the World Wide Web to improve the search for information that is needed by users browsing the Internet.

To understand the fundamentals of database technology, however, we must start from the basics of traditional database applications. So, in Section 1.1 of this chapter we define what a database is, and then we give definitions of other basic terms. In Section 1.2, we provide a simple UNIVERSITY database example to illustrate our discussion. Section 1.3 describes some of the main characteristics of database systems, and Sections 1.4 and 1.5 categorize the types of personnel whose jobs involve using and interacting with database systems. Sections 1.6, 1.7, and 1.8 offer a more thorough discussion of the various capabilities provided by database systems and discuss some typical database applications. Section 1.9 summarizes the chapter.

The reader who desires only a quick introduction to database systems can study Sections 1.1 through 1.5, then skip or browse through Sections 1.6 through 1.8 and go on to Chapter 2.

1.1 INTRODUCTION

Databases and database technology are having a major impact on the growing use of computers. It is fair to say that databases play a critical role in almost all areas where computers are used, including business, electronic commerce, engineering, medicine, law, education, and library science, to name a few. The word *database* is in such common use that we must begin by defining what a database is. Our initial definition is quite general.

A **database** is a collection of related data.¹ By **data**, we mean known facts that can be recorded and that have implicit meaning. For example, consider the names, telephone numbers, and addresses of the people you know. You may have recorded this data in an indexed address book, or you may have stored it on a hard drive, using a personal computer and software such as Microsoft Access, or Excel. This is a collection of related data with an implicit meaning and hence is a database.

The preceding definition of database is quite general; for example, we may consider the collection of words that make up this page of text to be related data and hence to constitute a database. However, the common use of the term *database* is usually more restricted. A database has the following implicit properties:

- A database represents some aspect of the real world, sometimes called the **miniworld** or the **universe of discourse (UoD)**. Changes to the miniworld are reflected in the database.
- A database is a logically coherent collection of data with some inherent meaning. A random assortment of data cannot correctly be referred to as a database.
- A database is designed, built, and populated with data for a specific purpose. It has an intended group of users and some preconceived applications in which these users are interested.

1. We will use the word *data* as both singular and plural, as is common in database literature; context will determine whether it is singular or plural. In standard English, *data* is used only for plural; *datum* is used for singular.

In other words, a database has some source from which data is derived, some degree of interaction with events in the real world, and an audience that is actively interested in the contents of the database.

A database can be of any size and of varying complexity. For example, the list of names and addresses referred to earlier may consist of only a few hundred records, each with a simple structure. On the other hand, the computerized catalog of a large library may contain half a million entries organized under different categories—by primary author's last name, by subject, by book title—with each category organized in alphabetic order. A database of even greater size and complexity is maintained by the Internal Revenue Service to keep track of the tax forms filed by U.S. taxpayers. If we assume that there are 100 million taxpayers and if each taxpayer files an average of five forms with approximately 400 characters of information per form, we would get a database of $100 \times 10^6 \times 400 \times 5$ characters (bytes) of information. If the IRS keeps the past three returns for each taxpayer in addition to the current return, we would get a database of 8×10^{11} bytes (800 gigabytes). This huge amount of information must be organized and managed so that users can search for, retrieve, and update the data as needed.

A database may be generated and maintained manually or it may be computerized. For example, a library card catalog is a database that may be created and maintained manually. A computerized database may be created and maintained either by a group of application programs written specifically for that task or by a database management system. Of course, we are only concerned with computerized databases in this book.

A **database management system (DBMS)** is a collection of programs that enables users to create and maintain a database. The DBMS is hence a *general-purpose software system* that facilitates the processes of *defining*, *constructing*, *manipulating*, and *sharing* databases among various users and applications. **Defining** a database involves specifying the data types, structures, and constraints for the data to be stored in the database. **Constructing** the database is the process of storing the data itself on some storage medium that is controlled by the DBMS. **Manipulating** a database includes such functions as querying the database to retrieve specific data, updating the database to reflect changes in the miniworld, and generating reports from the data. **Sharing** a database allows multiple users and programs to access the database concurrently.

Other important functions provided by the DBMS include *protecting* the database and *maintaining* it over a long period of time. **Protection** includes both *system protection* against hardware or software malfunction (or crashes), and *security protection* against unauthorized or malicious access. A typical large database may have a life cycle of many years, so the DBMS must be able to **maintain** the database system by allowing the system to evolve as requirements change over time.

It is not necessary to use general-purpose DBMS software to implement a computerized database. We could write our own set of programs to create and maintain the database, in effect creating our own *special-purpose* DBMS software. In either case—whether we use a general-purpose DBMS or not—we usually have to deploy a considerable amount of complex software. In fact, most DBMSs are very complex software systems.

To complete our initial definitions, we will call the database and DBMS software together a **database system**. Figure 1.1 illustrates some of the concepts we discussed so far.

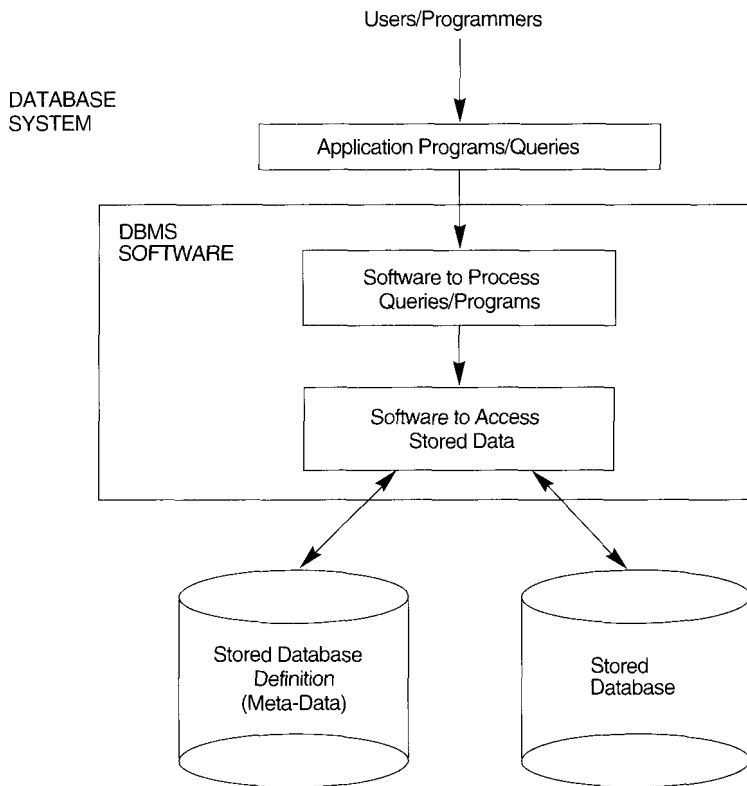


FIGURE 1.1 A simplified database system environment.

1.2 AN EXAMPLE

Let us consider a simple example that most readers may be familiar with: a UNIVERSITY database for maintaining information concerning students, courses, and grades in a university environment. Figure 1.2 shows the database structure and a few sample data for such a database. The database is organized as five files, each of which stores data records of the same type.² The STUDENT file stores data on each student, the COURSE file stores data on each course, the SECTION file stores data on each section of a course, the GRADE_REPORT file stores the grades that students receive in the various sections they have completed, and the PREREQUISITE file stores the prerequisites of each course.

To define this database, we must specify the structure of the records of each file by specifying the different types of **data elements** to be stored in each record. In Figure 1.2, each STUDENT record includes data to represent the student's Name, StudentNumber, Class

2. We use the term *file* informally here. At a conceptual level, a *file* is a collection of records that may or may not be ordered.

STUDENT	Name	StudentNumber	Class	Major
	Smith	17	1	CS
	Brown	8	2	CS

COURSE	CourseName	CourseNumber	CreditHours	Department
	Intro to Computer Science	CS1310	4	CS
	Data Structures	CS3320	4	CS
	Discrete Mathematics	MATH2410	3	MATH
	Database	CS3380	3	CS

SECTION	SectionIdentifier	CourseNumber	Semester	Year	Instructor
	85	MATH2410	Fall	98	King
	92	CS1310	Fall	98	Anderson
	102	CS3320	Spring	99	Knuth
	112	MATH2410	Fall	99	Chang
	119	CS1310	Fall	99	Anderson
	135	CS3380	Fall	99	Stone

GRADE_REPORT	StudentNumber	SectionIdentifier	Grade
	17	112	B
	17	119	C
	8	85	A
	8	92	A
	8	102	B
	8	135	A

PREREQUISITE	CourseNumber	PrerequisiteNumber
	CS3380	CS3320
	CS3380	MATH2410
	CS3320	CS1310

FIGURE 1.2 A database that stores student and course information.

(freshman or 1, sophomore or 2, . . .), and Major (mathematics or math, computer science or CS, . . .); each `COURSE` record includes data to represent the `CourseName`, `CourseNumber`, `CreditHours`, and `Department` (the department that offers the course); and so on. We must also specify a **data type** for each data element within a record. For example, we can specify that `Name` of `STUDENT` is a string of alphabetic characters, `StudentNumber` of `STUDENT` is an integer, and `Grade` of `GRADE_REPORT` is a single character from the set {A, B, C, D, F, I}. We may also use a coding scheme to represent the values of

a data item. For example, in Figure 1.2 we represent the Class of a STUDENT as 1 for freshman, 2 for sophomore, 3 for junior, 4 for senior, and 5 for graduate student.

To *construct* the UNIVERSITY database, we store data to represent each student, course, section, grade report, and prerequisite as a record in the appropriate file. Notice that records in the various files may be related. For example, the record for “Smith” in the STUDENT file is related to two records in the GRADE_REPORT file that specify Smith’s grades in two sections. Similarly, each record in the PREREQUISITE file relates two course records: one representing the course and the other representing the prerequisite. Most medium-size and large databases include many types of records and have *many relationships* among the records.

Database *manipulation* involves querying and updating. Examples of queries are “retrieve the transcript—a list of all courses and grades—of Smith,” “list the names of students who took the section of the Database course offered in fall 1999 and their grades in that section,” and “what are the prerequisites of the Database course?” Examples of updates are “change the class of Smith to Sophomore,” “create a new section for the Database course for this semester,” and “enter a grade of A for Smith in the Database section of last semester.” These informal queries and updates must be specified precisely in the query language of the DBMS before they can be processed.

1.3 CHARACTERISTICS OF THE DATABASE APPROACH

A number of characteristics distinguish the database approach from the traditional approach of programming with files. In traditional **file processing**, each user defines and implements the files needed for a specific software application as part of programming the application. For example, one user, the *grade reporting office*, may keep a file on students and their grades. Programs to print a student’s transcript and to enter new grades into the file are implemented as part of the application. A second user, the *accounting office*, may keep track of students’ fees and their payments. Although both users are interested in data about students, each user maintains separate files—and programs to manipulate these files—because each requires some data not available from the other user’s files. This redundancy in defining and storing data results in wasted storage space and in redundant efforts to maintain common data up to date.

In the database approach, a single repository of data is maintained that is defined once and then is accessed by various users. The main characteristics of the database approach versus the file-processing approach are the following:

- Self-describing nature of a database system
- Insulation between programs and data, and data abstraction
- Support of multiple views of the data
- Sharing of data and multiuser transaction processing

We next describe each of these characteristics in a separate section. Additional characteristics of database systems are discussed in Sections 1.6 through 1.8.

1.3.1 Self-Describing Nature of a Database System

A fundamental characteristic of the database approach is that the database system contains not only the database itself but also a complete definition or description of the database structure and constraints. This definition is stored in the DBMS **catalog**, which contains information such as the structure of each file, the type and storage format of each data item, and various constraints on the data. The information stored in the catalog is called **meta-data**, and it describes the structure of the primary database (Figure 1.1).

The catalog is used by the DBMS software and also by database users who need information about the database structure. A general-purpose DBMS software package is not written for a specific database application, and hence it must refer to the catalog to know the structure of the files in a specific database, such as the type and format of data it will access. The DBMS software must work equally well with *any number of database applications*—for example, a university database, a banking database, or a company database—as long as the database definition is stored in the catalog.

In traditional file processing, data definition is typically part of the application programs themselves. Hence, these programs are constrained to work with only *one specific database*, whose structure is declared in the application programs. For example, an application program written in C++ may have struct or class declarations, and a COBOL program has Data Division statements to define its files. Whereas file-processing software can access only specific databases, DBMS software can access diverse databases by extracting the database definitions from the catalog and then using these definitions.

In the example shown in Figure 1.2, the DBMS catalog will store the definitions of all the files shown. These definitions are specified by the database designer prior to creating the actual database and are stored in the catalog. Whenever a request is made to access, say, the Name of a STUDENT record, the DBMS software refers to the catalog to determine the structure of the STUDENT file and the position and size of the Name data item within a STUDENT record. By contrast, in a typical file-processing application, the file structure and, in the extreme case, the exact location of Name within a STUDENT record are already coded within each program that accesses this data item.

1.3.2 Insulation between Programs and Data, and Data Abstraction

In traditional file processing, the structure of data files is embedded in the application programs, so any changes to the structure of a file may require *changing all programs* that access this file. By contrast, DBMS access programs do not require such changes in most cases. The structure of data files is stored in the DBMS catalog separately from the access programs. We call this property **program-data independence**. For example, a file access program may be written in such a way that it can access only STUDENT records of the structure shown in Figure 1.3. If we want to add another piece of data to each STUDENT record, say the BirthDate, such a program will no longer work and must be changed. By contrast, in a DBMS environment, we just need to change the description of STUDENT records in the catalog to reflect the inclusion of the new data item BirthDate; no programs are changed. The next time a DBMS program refers to the catalog, the new structure of STUDENT records will be accessed and used.

<u>Data Item Name</u>	<u>Starting Position in Record</u>	<u>Length in Characters (bytes)</u>
Name	1	30
StudentNumber	31	4
Class	35	4
Major	39	4

FIGURE 1.3 Internal storage format for a STUDENT record.

In some types of database systems, such as object-oriented and object-relational systems (see Chapters 20 to 22), users can define operations on data as part of the database definitions. An **operation** (also called a *function* or *method*) is specified in two parts. The *interface* (or *signature*) of an operation includes the operation name and the data types of its arguments (or parameters). The *implementation* (or *method*) of the operation is specified separately and can be changed without affecting the interface. User application programs can operate on the data by invoking these operations through their names and arguments, regardless of how the operations are implemented. This may be termed **program-operation independence**.

The characteristic that allows program-data independence and program-operation independence is called **data abstraction**. A DBMS provides users with a **conceptual representation** of data that does not include many of the details of how the data is stored or how the operations are implemented. Informally, a **data model** is a type of data abstraction that is used to provide this conceptual representation. The data model uses logical concepts, such as objects, their properties, and their interrelationships, that may be easier for most users to understand than computer storage concepts. Hence, the data model *hides* storage and implementation details that are not of interest to most database users.

For example, consider again Figure 1.2. The internal implementation of a file may be defined by its record length—the number of characters (bytes) in each record—and each data item may be specified by its starting byte within a record and its length in bytes. The STUDENT record would thus be represented as shown in Figure 1.3. But a typical database user is not concerned with the location of each data item within a record or its length; rather, the concern is that when a reference is made to Name of STUDENT, the correct value is returned. A conceptual representation of the STUDENT records is shown in Figure 1.2. Many other details of file storage organization—such as the access paths specified on a file—can be hidden from database users by the DBMS; we discuss storage details in Chapters 13 and 14.

In the database approach, the detailed structure and organization of each file are stored in the catalog. Database users and application programs refer to the conceptual representation of the files, and the DBMS extracts the details of file storage from the catalog when these are needed by the DBMS file access modules. Many data models can be used to provide this data abstraction to database users. A major part of this book is devoted to presenting various data models and the concepts they use to abstract the representation of data.

In object-oriented and object-relational databases, the abstraction process includes not only the data structure but also the operations on the data. These operations provide an abstraction of miniworld activities commonly understood by the users. For example,

(a)

TRANSCRIPT	StudentName	Student Transcript				
		CourseNumber	Grade	Semester	Year	SectionId
Smith		CS1310	C	Fall	99	119
		MATH2410	B	Fall	99	112
Brown		MATH2410	A	Fall	98	85
		CS1310	A	Fall	98	92
		CS3320	B	Spring	99	102
		CS3380	A	Fall	99	135

(b)

PREREQUISITES	CourseName	CourseNumber	Prerequisites
Database		CS3380	CS3320
			MATH2410
Data Structures		CS3320	CS1310

FIGURE 1.4 Two views derived from the database in Figure 1.2. (a) The STUDENT TRANSCRIPT view. (b) The COURSE PREREQUISITES view.

an operation `CALCULATE_GPA` can be applied to a `STUDENT` object to calculate the grade point average. Such operations can be invoked by the user queries or application programs without having to know the details of how the operations are implemented. In that sense, an abstraction of the miniworld activity is made available to the user as an **abstract operation**.

1.3.3 Support of Multiple Views of the Data

A database typically has many users, each of whom may require a different perspective or **view** of the database. A view may be a subset of the database or it may contain **virtual data** that is derived from the database files but is not explicitly stored. Some users may not need to be aware of whether the data they refer to is stored or derived. A multiuser DBMS whose users have a variety of distinct applications must provide facilities for defining multiple views. For example, one user of the database of Figure 1.2 may be interested only in accessing and printing the transcript of each student; the view for this user is shown in Figure 1.4a. A second user, who is interested only in checking that students have taken all the prerequisites of each course for which they register, may require the view shown in Figure 1.4b.

1.3.4 Sharing of Data and Multiuser Transaction Processing

A multiuser DBMS, as its name implies, must allow multiple users to access the database at the same time. This is essential if data for multiple applications is to be integrated and

maintained in a single database. The DBMS must include **concurrency control** software to ensure that several users trying to update the same data do so in a controlled manner so that the result of the updates is correct. For example, when several reservation clerks try to assign a seat on an airline flight, the DBMS should ensure that each seat can be accessed by only one clerk at a time for assignment to a passenger. These types of applications are generally called **online transaction processing (OLTP)** applications. A fundamental role of multiuser DBMS software is to ensure that concurrent transactions operate correctly.

The concept of a **transaction** has become central to many database applications. A transaction is an *executing program* or *process* that includes one or more database accesses, such as reading or updating of database records. Each transaction is supposed to execute a logically correct database access if executed in its entirety without interference from other transactions. The DBMS must enforce several transaction properties. The **isolation** property ensures that each transaction appears to execute in isolation from other transactions, even though hundreds of transactions may be executing concurrently. The **atomicity** property ensures that either all the database operations in a transaction are executed or none are. We discuss transactions in detail in Part V of the textbook.

The preceding characteristics are most important in distinguishing a DBMS from traditional file-processing software. In Section 1.6 we discuss additional features that characterize a DBMS. First, however, we categorize the different types of persons who work in a database system environment.

1.4 ACTORS ON THE SCENE

For a small personal database, such as the list of addresses discussed in Section 1.1, one person typically defines, constructs, and manipulates the database, and there is no sharing. However, many persons are involved in the design, use, and maintenance of a large database with hundreds of users. In this section we identify the people whose jobs involve the day-to-day use of a large database; we call them the “actors on the scene.” In Section 1.5 we consider people who may be called “workers behind the scene”—those who work to maintain the database system environment but who are not actively interested in the database itself.

1.4.1 Database Administrators

In any organization where many persons use the same resources, there is a need for a chief administrator to oversee and manage these resources. In a database environment, the primary resource is the database itself, and the secondary resource is the DBMS and related software. Administering these resources is the responsibility of the **database administrator (DBA)**. The DBA is responsible for authorizing access to the database, for coordinating and monitoring its use, and for acquiring software and hardware resources as needed. The DBA is accountable for problems such as breach of security or poor system response time. In large organizations, the DBA is assisted by a staff that helps carry out these functions.

1.4.2 Database Designers

Database designers are responsible for identifying the data to be stored in the database and for choosing appropriate structures to represent and store this data. These tasks are mostly undertaken before the database is actually implemented and populated with data. It is the responsibility of database designers to communicate with all prospective database users in order to understand their requirements, and to come up with a design that meets these requirements. In many cases, the designers are on the staff of the DBA and may be assigned other staff responsibilities after the database design is completed. Database designers typically interact with each potential group of users and develop **views** of the database that meet the data and processing requirements of these groups. Each view is then analyzed and *integrated* with the views of other user groups. The final database design must be capable of supporting the requirements of all user groups.

1.4.3 End Users

End users are the people whose jobs require access to the database for querying, updating, and generating reports; the database primarily exists for their use. There are several categories of end users:

- **Casual end users** occasionally access the database, but they may need different information each time. They use a sophisticated database query language to specify their requests and are typically middle- or high-level managers or other occasional browsers.
- **Naive or parametric end users** make up a sizable portion of database end users. Their main job function revolves around constantly querying and updating the database, using standard types of queries and updates—called **canned transactions**—that have been carefully programmed and tested. The tasks that such users perform are varied:
 - Bank tellers check account balances and post withdrawals and deposits.
 - Reservation clerks for airlines, hotels, and car rental companies check availability for a given request and make reservations.
 - Clerks at receiving stations for courier mail enter package identifications via bar codes and descriptive information through buttons to update a central database of received and in-transit packages.
- **Sophisticated end users** include engineers, scientists, business analysts, and others who thoroughly familiarize themselves with the facilities of the DBMS so as to implement their applications to meet their complex requirements.
- **Stand-alone users** maintain personal databases by using ready-made program packages that provide easy-to-use menu-based or graphics-based interfaces. An example is the user of a tax package that stores a variety of personal financial data for tax purposes.

A typical DBMS provides multiple facilities to access a database. Naive end users need to learn very little about the facilities provided by the DBMS; they have to understand only the user interfaces of the standard transactions designed and implemented for their

use. Casual users learn only a few facilities that they may use repeatedly. Sophisticated users try to learn most of the DBMS facilities in order to achieve their complex requirements. Stand-alone users typically become very proficient in using a specific software package.

1.4.4 System Analysts and Application Programmers (Software Engineers)

System analysts determine the requirements of end users, especially naive and parametric end users, and develop specifications for canned transactions that meet these requirements. **Application programmers** implement these specifications as programs; then they test, debug, document, and maintain these canned transactions. Such analysts and programmers—commonly referred to as **software engineers**—should be familiar with the full range of capabilities provided by the DBMS to accomplish their tasks.

1.5 WORKERS BEHIND THE SCENE

In addition to those who design, use, and administer a database, others are associated with the design, development, and operation of the DBMS *software and system environment*. These persons are typically not interested in the database itself. We call them the “workers behind the scene,” and they include the following categories.

- **DBMS system designers and implementers** are persons who design and implement the DBMS modules and interfaces as a software package. A DBMS is a very complex software system that consists of many components, or **modules**, including modules for implementing the catalog, processing query language, processing the interface, accessing and buffering data, controlling concurrency, and handling data recovery and security. The DBMS must interface with other system software, such as the operating system and compilers for various programming languages.
- **Tool developers** include persons who design and implement **tools**—the software packages that facilitate database system design and use and that help improve performance. Tools are optional packages that are often purchased separately. They include packages for database design, performance monitoring, natural language or graphical interfaces, prototyping, simulation, and test data generation. In many cases, independent software vendors develop and market these tools.
- **Operators and maintenance personnel** are the system administration personnel who are responsible for the actual running and maintenance of the hardware and software environment for the database system.

Although these categories of workers behind the scene are instrumental in making the database system available to end users, they typically do not use the database for their own purposes.

1.6 ADVANTAGES OF USING THE DBMS APPROACH

In this section we discuss some of the advantages of using a DBMS and the capabilities that a good DBMS should possess. These capabilities are in addition to the four main characteristics discussed in Section 1.3. The DBA must utilize these capabilities to accomplish a variety of objectives related to the design, administration, and use of a large multiuser database.

1.6.1 Controlling Redundancy

In traditional software development utilizing file processing, every user group maintains its own files for handling its data-processing applications. For example, consider the UNIVERSITY database example of Section 1.2; here, two groups of users might be the course registration personnel and the accounting office. In the traditional approach, each group independently keeps files on students. The accounting office also keeps data on registration and related billing information, whereas the registration office keeps track of student courses and grades. Much of the data is stored twice: once in the files of each user group. Additional user groups may further duplicate some or all of the same data in their own files.

This **redundancy** in storing the same data multiple times leads to several problems. First, there is the need to perform a single logical update—such as entering data on a new student—multiple times: once for each file where student data is recorded. This leads to *duplication of effort*. Second, *storage space is wasted* when the same data is stored repeatedly, and this problem may be serious for large databases. Third, files that represent the same data may become *inconsistent*. This may happen because an update is applied to some of the files but not to others. Even if an update—such as adding a new student—is applied to all the appropriate files, the data concerning the student may still be *inconsistent* because the updates are applied independently by each user group. For example, one user group may enter a student's birthdate erroneously as JAN-19-1984, whereas the other user groups may enter the correct value of JAN-29-1984.

In the database approach, the views of different user groups are integrated during database design. Ideally, we should have a database design that stores each logical data item—such as a student's name or birth date—in *only one place* in the database. This ensures consistency, and it saves storage space. However, in practice, it is sometimes necessary to use **controlled redundancy** for improving the performance of queries. For example, we may store StudentName and CourseNumber redundantly in a GRADE_REPORT file (Figure 1.5a) because whenever we retrieve a GRADE_REPORT record, we want to retrieve the student name and course number along with the grade, student number, and section identifier. By placing all the data together, we do not have to search multiple files to collect this data. In such cases, the DBMS should have the capability to *control* this redundancy so as to prohibit inconsistencies among the files. This may be done by automatically checking that the StudentName-StudentNumber values in any GRADE_REPORT record in Figure 1.5a match one of the Name-StudentNumber values of a STUDENT record (Figure 1.2). Similarly, the SectionIdentifier-CourseNumber values in

(a)

GRADE_REPORT	StudentNumber	StudentName	SectionIdentifier	CourseNumber	Grade
	17	Smith	112	MATH2410	B
	17	Smith	119	CS1310	C
	8	Brown	85	MATH2410	A
	8	Brown	92	CS1310	A
	8	Brown	102	CS3320	B
	8	Brown	135	CS3380	A

(b)

GRADE_REPORT	StudentNumber	StudentName	SectionIdentifier	CourseNumber	Grade
	17	Brown	112	MATH2410	B

FIGURE 1.5 Redundant storage of StudentName and CourseNumber in GRADE_REPORT. (a) Consistent data. (b) Inconsistent record.

GRADE_REPORT can be checked against SECTION records. Such checks can be specified to the DBMS during database design and automatically enforced by the DBMS whenever the GRADE_REPORT file is updated. Figure 1.5b shows a GRADE_REPORT record that is inconsistent with the STUDENT file of Figure 1.2, which may be entered erroneously if the redundancy is *not controlled*.

1.6.2 Restricting Unauthorized Access

When multiple users share a large database, it is likely that most users will not be authorized to access all information in the database. For example, financial data is often considered confidential, and hence only authorized persons are allowed to access such data. In addition, some users may be permitted only to retrieve data, whereas others are allowed both to retrieve and to update. Hence, the *type of access operation*—retrieval or update—must also be controlled. Typically, users or user groups are given account numbers protected by passwords, which they can use to gain access to the database. A DBMS should provide a **security and authorization subsystem**, which the DBA uses to create accounts and to specify account restrictions. The DBMS should then enforce these restrictions automatically. Notice that we can apply similar controls to the DBMS software. For example, only the DBA's staff may be allowed to use certain **privileged software**, such as the software for creating new accounts. Similarly, parametric users may be allowed to access the database only through the canned transactions developed for their use.

1.6.3 Providing Persistent Storage for Program Objects

Databases can be used to provide **persistent storage** for program objects and data structures. This is one of the main reasons for **object-oriented database systems**. Programming languages typically have complex data structures, such as record types in Pascal or class

definitions in C++ or Java. The values of program variables are discarded once a program terminates, unless the programmer explicitly stores them in permanent files, which often involves converting these complex structures into a format suitable for file storage. When the need arises to read this data once more, the programmer must convert from the file format to the program variable structure. Object-oriented database systems are compatible with programming languages such as C++ and Java, and the DBMS software automatically performs any necessary conversions. Hence, a complex object in C++ can be stored permanently in an object-oriented DBMS. Such an object is said to be **persistent**, since it survives the termination of program execution and can later be directly retrieved by another C++ program.

The persistent storage of program objects and data structures is an important function of database systems. Traditional database systems often suffered from the so-called **impedance mismatch problem**, since the data structures provided by the DBMS were incompatible with the programming language's data structures. Object-oriented database systems typically offer data structure **compatibility** with one or more object-oriented programming languages.

1.6.4 Providing Storage Structures for Efficient Query Processing

Database systems must provide capabilities for *efficiently executing queries and updates*. Because the database is typically stored on disk, the DBMS must provide specialized data structures to speed up disk search for the desired records. Auxiliary files called **indexes** are used for this purpose. Indexes are typically based on tree data structures or hash data structures, suitably modified for disk search. In order to process the database records needed by a particular query, those records must be copied from disk to memory. Hence, the DBMS often has a **buffering** module that maintains parts of the database in main memory buffers. In other cases, the DBMS may use the operating system to do the buffering of disk data.

The **query processing and optimization** module of the DBMS is responsible for choosing an efficient query execution plan for each query based on the existing storage structures. The choice of which indexes to create and maintain is part of *physical database design and tuning*, which is one of the responsibilities of the DBA staff.

1.6.5 Providing Backup and Recovery

A DBMS must provide facilities for recovering from hardware or software failures. The **backup and recovery subsystem** of the DBMS is responsible for recovery. For example, if the computer system fails in the middle of a complex update transaction, the recovery subsystem is responsible for making sure that the database is restored to the state it was in before the transaction started executing. Alternatively, the recovery subsystem could ensure that the transaction is resumed from the point at which it was interrupted so that its full effect is recorded in the database.

1.6.6 Providing Multiple User Interfaces

Because many types of users with varying levels of technical knowledge use a database, a DBMS should provide a variety of user interfaces. These include query languages for casual users, programming language interfaces for application programmers, forms and command codes for parametric users, and menu-driven interfaces and natural language interfaces for stand-alone users. Both forms-style interfaces and menu-driven interfaces are commonly known as **graphical user interfaces (GUIs)**. Many specialized languages and environments exist for specifying GUIs. Capabilities for providing Web GUI interfaces to a database—or Web-enabling a database—are also quite common.

1.6.7 Representing Complex Relationships among Data

A database may include numerous varieties of data that are interrelated in many ways. Consider the example shown in Figure 1.2. The record for Brown in the `STUDENT` file is related to four records in the `GRADE_REPORT` file. Similarly, each section record is related to one course record as well as to a number of `GRADE_REPORT` records—one for each student who completed that section. A DBMS must have the capability to represent a variety of complex relationships among the data as well as to retrieve and update related data easily and efficiently.

1.6.8 Enforcing Integrity Constraints

Most database applications have certain **integrity constraints** that must hold for the data. A DBMS should provide capabilities for defining and enforcing these constraints. The simplest type of integrity constraint involves specifying a data type for each data item. For example, in Figure 1.2, we may specify that the value of the `Class` data item within each `STUDENT` record must be an integer between 1 and 5 and that the value of `Name` must be a string of no more than 30 alphabetic characters. A more complex type of constraint that frequently occurs involves specifying that a record in one file must be related to records in other files. For example, in Figure 1.2, we can specify that “every section record must be related to a course record.” Another type of constraint specifies uniqueness on data item values, such as “every course record must have a unique value for `CourseNumber`.” These constraints are derived from the meaning or **semantics** of the data and of the miniworld it represents. It is the database designers’ responsibility to identify integrity constraints during database design. Some constraints can be specified to the DBMS and automatically enforced. Other constraints may have to be checked by update programs or at the time of data entry.

A data item may be entered erroneously and still satisfy the specified integrity constraints. For example, if a student receives a grade of A but a grade of C is entered in the database, the DBMS *cannot* discover this error automatically, because C is a valid value for the `Grade` data type. Such data entry errors can only be discovered manually (when the student receives the grade and complains) and corrected later by updating the database. However, a grade of Z can be rejected automatically by the DBMS, because Z is not a valid value for the `Grade` data type.

1.6.9 Permitting Inferencing and Actions Using Rules

Some database systems provide capabilities for defining *deduction rules* for *inferencing* new information from the stored database facts. Such systems are called **deductive database systems**. For example, there may be complex rules in the miniworld application for determining when a student is on probation. These can be specified *declaratively* as **rules**, which when compiled and maintained by the DBMS can determine all students on probation. In a traditional DBMS, an explicit *procedural program code* would have to be written to support such applications. But if the miniworld rules change, it is generally more convenient to change the declared deduction rules than to recode procedural programs. More powerful functionality is provided by **active database systems**, which provide active rules that can automatically initiate actions when certain events and conditions occur.

1.6.10 Additional Implications of Using the Database Approach

This section discusses some additional implications of using the database approach that can benefit most organizations.

Potential for Enforcing Standards. The database approach permits the DBA to define and enforce standards among database users in a large organization. This facilitates communication and cooperation among various departments, projects, and users within the organization. Standards can be defined for names and formats of data elements, display formats, report structures, terminology, and so on. The DBA can enforce standards in a centralized database environment more easily than in an environment where each user group has control of its own files and software.

Reduced Application Development Time. A prime selling feature of the database approach is that developing a new application—such as the retrieval of certain data from the database for printing a new report—takes very little time. Designing and implementing a new database from scratch may take more time than writing a single specialized file application. However, once a database is up and running, substantially less time is generally required to create new applications using DBMS facilities. Development time using a DBMS is estimated to be one-sixth to one-fourth of that for a traditional file system.

Flexibility. It may be necessary to change the structure of a database as requirements change. For example, a new user group may emerge that needs information not currently in the database. In response, it may be necessary to add a file to the database or to extend the data elements in an existing file. Modern DBMSs allow certain types of evolutionary changes to the structure of the database without affecting the stored data and the existing application programs.

Availability of Up-to-Date Information. A DBMS makes the database available to all users. As soon as one user's update is applied to the database, all other users can

immediately see this update. This availability of up-to-date information is essential for many transaction-processing applications, such as reservation systems or banking databases, and it is made possible by the *concurrency control* and *recovery subsystems* of a DBMS.

Economies of Scale. The DBMS approach permits consolidation of data and applications, thus reducing the amount of wasteful overlap between activities of data-processing personnel in different projects or departments. This enables the whole organization to invest in more powerful processors, storage devices, or communication gear, rather than having each department purchase its own (weaker) equipment. This reduces overall costs of operation and management.

1.7 A BRIEF HISTORY OF DATABASE APPLICATIONS

We now give a brief historical overview of the applications that use DBMSs, and how these applications provided the impetus for new types of database systems.

1.7.1 Early Database Applications Using Hierarchical and Network Systems

Many early database applications maintained records in large organizations, such as corporations, universities, hospitals, and banks. In many of these applications, there were large numbers of records of similar structure. For example, in a university application, similar information would be kept for each student, each course, each grade record, and so on. There were also many types of records and many interrelationships among them.

One of the main problems with early database systems was the intermixing of conceptual relationships with the physical storage and placement of records on disk. For example, the grade records of a particular student could be physically stored next to the student record. Although this provided very efficient access for the original queries and transactions that the database was designed to handle, it did not provide enough flexibility to access records efficiently when new queries and transactions were identified. In particular, new queries that required a different storage organization for efficient processing were quite difficult to implement efficiently. It was also quite difficult to *reorganize the database when changes were made to the requirements of the application.*

Another shortcoming of early systems was that they provided only programming language interfaces. This made it time-consuming and expensive to implement new queries and transactions, since new programs had to be written, tested, and debugged. Most of these database systems were implemented on large and expensive mainframe computers starting in the mid-1960s and through the 1970s and 1980s. The main types of early systems were based on three main paradigms: hierarchical systems, network model based systems, and inverted file systems.

1.7.2 Providing Application Flexibility with Relational Databases

Relational databases were originally proposed to separate the physical storage of data from its conceptual representation and to provide a mathematical foundation for databases. The relational data model also introduced high-level query languages that provided an alternative to programming language interfaces; hence, it was a lot quicker to write new queries. Relational representation of data somewhat resembles the example we presented in Figure 1.2. Relational systems were initially targeted to the same applications as earlier systems, but were meant to provide flexibility to quickly develop new queries and to reorganize the database as requirements changed.

Early experimental relational systems developed in the late 1970s and the commercial RDBMSs (relational database management systems) introduced in the early 1980s were quite slow, since they did not use physical storage pointers or record placement to access related data records. With the development of new storage and indexing techniques and better query processing and optimization, their performance improved. Eventually, relational databases became the dominant type of database systems for traditional database applications. Relational databases now exist on almost all types of computers, from small personal computers to large servers.

1.7.3 Object-Oriented Applications and the Need for More Complex Databases

The emergence of object-oriented programming languages in the 1980s and the need to store and share complex-structured objects led to the development of object-oriented databases. Initially, they were considered a competitor to relational databases, since they provided more general data structures. They also incorporated many of the useful object-oriented paradigms, such as abstract data types, encapsulation of operations, inheritance, and object identity. However, the complexity of the model and the lack of an early standard contributed to their limited use. They are now mainly used in specialized applications, such as engineering design, multimedia publishing, and manufacturing systems.

1.7.4 Interchanging Data on the Web for E-Commerce

The World Wide Web provided a large network of interconnected computers. Users can create documents using a Web publishing language, such as HTML (HyperText Markup Language), and store these documents on Web servers where other users (clients) can access them. Documents can be linked together through **hyperlinks**, which are pointers to other documents. In the 1990s, electronic commerce (e-commerce) emerged as a major application on the Web. It quickly became apparent that parts of the information on e-commerce Web pages were often dynamically extracted data from DBMSs. A variety of techniques were developed to allow the interchange of data on the

Web. Currently, XML (eXtended Markup Language) is considered to be the primary standard for interchanging data among various types of databases and Web pages. XML combines concepts from the models used in document systems with database modeling concepts.

1.7.5 Extending Database Capabilities for New Applications

The success of database systems in traditional applications encouraged developers of other types of applications to attempt to use them. Such applications traditionally used their own specialized file and data structures. The following are examples of these applications:

- **Scientific** applications that store large amounts of data resulting from scientific experiments in areas such as high-energy physics or the mapping of the human genome.
- Storage and retrieval of **images**, from scanned news or personal photographs to satellite photograph images and images from medical procedures such as X-rays or MRI (magnetic resonance imaging).
- Storage and retrieval of **videos**, such as movies, or **video clips** from news or personal digital cameras.
- **Data mining** applications that analyze large amounts of data searching for the occurrences of specific patterns or relationships.
- **Spatial** applications that store spatial locations of data such as weather information or maps used in geographical information systems.
- **Time series** applications that store information such as economic data at regular points in time, for example, daily sales or monthly gross national product figures.

It was quickly apparent that basic relational systems were not very suitable for many of these applications, usually for one or more of the following reasons:

- More complex data structures were needed for modeling the application than the simple relational representation.
- New data types were needed in addition to the basic numeric and character string types.
- New operations and query language constructs were necessary to manipulate the new data types.
- New storage and indexing structures were needed.

This led DBMS developers to add functionality to their systems. Some functionality was general purpose, such as incorporating concepts from object-oriented databases into relational systems. Other functionality was special purpose, in the form of optional modules that could be used for specific applications. For example, users could buy a time series module to use with their relational DBMS for their time series application.

1.8 WHEN NOT TO USE A DBMS

In spite of the advantages of using a DBMS, there are a few situations in which such a system may involve unnecessary overhead costs that would not be incurred in traditional file processing. The overhead costs of using a DBMS are due to the following:

- High initial investment in hardware, software, and training
- The generality that a DBMS provides for defining and processing data
- Overhead for providing security, concurrency control, recovery, and integrity functions

Additional problems may arise if the database designers and DBA do not properly design the database or if the database systems applications are not implemented properly. Hence, it may be more desirable to use regular files under the following circumstances:

- The database and applications are simple, well defined, and not expected to change.
- There are stringent real-time requirements for some programs that may not be met because of DBMS overhead.
- Multiple-user access to data is not required.

1.9 SUMMARY

In this chapter we defined a database as a collection of related data, where *data* means recorded facts. A typical database represents some aspect of the real world and is used for specific purposes by one or more groups of users. A DBMS is a generalized software package for implementing and maintaining a computerized database. The database and software together form a database system. We identified several characteristics that distinguish the database approach from traditional file-processing applications. We then discussed the main categories of database users, or the “actors on the scene.” We noted that, in addition to database users, there are several categories of support personnel, or “workers behind the scene,” in a database environment.

We then presented a list of capabilities that should be provided by the DBMS software to the DBA, database designers, and users to help them design, administer, and use a database. Following this, we gave a brief historical perspective on the evolution of database applications. Finally, we discussed the overhead costs of using a DBMS and discussed some situations in which it may not be advantageous to use a DBMS.

Review Questions

- 1.1. Define the following terms: *data*, *database*, *DBMS*, *database system*, *database catalog*, *program-data independence*, *user view*, *DBA*, *end user*, *canned transaction*, *deductive database system*, *persistent object*, *meta-data*, *transaction-processing application*.
- 1.2. What three main types of actions involve databases? Briefly discuss each.

- 1.3. Discuss the main characteristics of the database approach and how it differs from traditional file systems.
- 1.4. What are the responsibilities of the DBA and the database designers?
- 1.5. What are the different types of database end users? Discuss the main activities of each.
- 1.6. Discuss the capabilities that should be provided by a DBMS.

Exercises

- 1.7. Identify some informal queries and update operations that you would expect to apply to the database shown in Figure 1.2.
- 1.8. What is the difference between controlled and uncontrolled redundancy? Illustrate with examples.
- 1.9. Name all the relationships among the records of the database shown in Figure 1.2.
- 1.10. Give some additional views that may be needed by other user groups for the database shown in Figure 1.2.
- 1.11. Cite some examples of integrity constraints that you think should hold on the database shown in Figure 1.2.

Selected Bibliography

The October 1991 issue of *Communications of the ACM* and Kim (1995) include several articles describing next-generation DBMSs; many of the database features discussed in the former are now commercially available. The March 1976 issue of *ACM Computing Surveys* offers an early introduction to database systems and may provide a historical perspective for the interested reader.