

FUNDAMENTALS OF DATABASE SYSTEMS

7TH Edition

ELMASRI • NAVATHE

FUNDAMENTALS OF

Database Systems

SEVENTH EDITION

This page intentionally left blank

FUNDAMENTALS OF

Database Systems

SEVENTH EDITION

Ramez Elmasri

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

Shamkant B. Navathe

*College of Computing
Georgia Institute of Technology*

PEARSON

Boston Columbus Indianapolis New York San Francisco Hoboken
Amsterdam Cape Town Dubai London Madrid Milan Munich Paris Montreal Toronto
Delhi Mexico City São Paulo Sydney Hong Kong Seoul Singapore Taipei Tokyo

Vice President and Editorial Director, ECS:

Marcia J. Horton

Acquisitions Editor: *Matt Goldstein*

Editorial Assistant: *Kelsey Loanes*

Marketing Managers: *Bram Van Kempen, Demetrius Hall*

Marketing Assistant: *Jon Bryant*

Senior Managing Editor: *Scott Disanno*

Production Project Manager: *Rose Kernan*

Program Manager: *Carole Snyder*

Global HE Director of Vendor Sourcing

and Procurement: *Diane Hynes*

Director of Operations: *Nick Sklitsis*

Operations Specialist: *Maura Zaldivar-Garcia*

Cover Designer: *Black Horse Designs*

Manager, Rights and Permissions: *Rachel Youdelman*

Associate Project Manager, Rights and Permissions:

Timothy Nicholls

Full-Service Project Management: *Rashmi Tickyani,*
iEnergizer Aptara®, Ltd.

Composition: *iEnergizer Aptara®, Ltd.*

Printer/Binder: *Edwards Brothers Malloy*

Cover Printer: *Phoenix Color/Hagerstown*

Cover Image: *Micha Pawlitzki/Terra/Corbis*

Typeface: *10.5/12 Minion Pro*

Copyright © 2016, 2011, 2007 by Ramez Elmasri and Shamkant B. Navathe. All rights reserved. Manufactured in the United States of America. This publication is protected by Copyright and permissions should be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. To obtain permission(s) to use materials from this work, please submit a written request to Pearson Higher Education, Permissions Department, 221 River Street, Hoboken, NJ 07030.

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

The author and publisher of this book have used their best efforts in preparing this book. These efforts include the development, research, and testing of theories and programs to determine their effectiveness. The author and publisher make no warranty of any kind, expressed or implied, with regard to these programs or the documentation contained in this book. The author and publisher shall not be liable in any event for incidental or consequential damages with, or arising out of, the furnishing, performance, or use of these programs.

Microsoft and/or its respective suppliers make no representations about the suitability of the information contained in the documents and related graphics published as part of the services for any purpose. All such documents and related graphics are provided “as is” without warranty of any kind. Microsoft and/or its respective suppliers hereby disclaim all warranties and conditions with regard to this information, including all warranties and conditions of merchantability. Whether express, implied or statutory, fitness for a particular purpose, title and non-infringement. In no event shall microsoft and/or its respective suppliers be liable for any special, indirect or consequential damages or any damages whatsoever resulting from loss of use, data or profits, whether in an action of contract. Negligence or other tortious action, arising out of or in connection with the use or performance of information available from the services.

The documents and related graphics contained herein could include technical inaccuracies or typographical errors. Changes are periodically added to the information herein. Microsoft and/or its respective suppliers may make improvements and/or changes in the product(s) and/or the program(s) described herein at any time. Partial screen shots may be viewed in full within the software version specified.

Library of Congress Cataloging-in-Publication Data on File

10 9 8 7 6 5 4 3 2 1

PEARSON

ISBN-10: 0-13-397077-9

ISBN-13: 978-0-13-397077-7

*To Amalia
and
to Ramy, Riyad, Katrina, and Thomas
R. E.*

*To my wife Aruna for her love, support, and understanding
and
to Rohan, Maya, and Ayush for bringing so much joy into our lives
S.B.N.*

This page intentionally left blank

Preface

This book introduces the fundamental concepts necessary for designing, using, and implementing database systems and database applications. Our presentation stresses the fundamentals of database modeling and design, the languages and models provided by the database management systems, and database 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. Our goal is to provide an in-depth and up-to-date presentation of the most important aspects of database systems and applications, and related technologies. We assume that readers are familiar with elementary programming and data-structuring concepts and that they have had some exposure to the basics of computer organization.

New to This Edition

The following key features have been added in the seventh edition:

- A reorganization of the chapter ordering (this was based on a survey of the instructors who use the textbook); however, the book is still organized so that the individual instructor can choose to follow the new chapter ordering or *choose a different ordering of chapters* (for example, follow the chapter order from the sixth edition) when presenting the materials.
- There are two new chapters on recent advances in database systems and big data processing; one new chapter (Chapter 24) covers an introduction to the newer class of database systems known as **NOSQL databases**, and the other new chapter (Chapter 25) covers technologies for processing **big data**, including **MapReduce** and **Hadoop**.
- The chapter on query processing and optimization has been expanded and reorganized into two chapters; Chapter 18 focuses on strategies and algorithms for query processing whereas Chapter 19 focuses on query optimization techniques.
- A second UNIVERSITY database example has been added to the early chapters (Chapters 3 through 8) in addition to our COMPANY database example from the previous editions.
- Many of the individual chapters have been updated to varying degrees to include newer techniques and methods; rather than discuss these enhancements here,

we will describe them later in the preface when we discuss the organization of the seventh edition.

The following are key features of the book:

- A self-contained, flexible organization that can be tailored to individual needs; in particular, *the chapters can be used in different orders* depending on the instructor's preference.
- A companion website (<http://www.pearsonhighered.com/cs-resources>) includes data to be loaded into various types of relational databases for more realistic student laboratory exercises.
- A dependency chart (shown later in this preface) to show which chapters depend on other earlier chapters; this can guide the instructor who wants to tailor the *order of presentation of the chapters*.
- A collection of supplements, including a robust set of materials for instructors and students such as PowerPoint slides, figures from the text, and an instructor's guide with solutions.

Organization and Contents of the Seventh Edition

There are some organizational changes in the seventh edition as well as improvement to the individual chapters. The book is now divided into 12 parts as follows:

- Part 1 (Chapters 1 and 2) describes the basic introductory concepts necessary for a good understanding of database models, systems, and languages. Chapters 1 and 2 introduce databases, typical users, and DBMS concepts, terminology, and architecture, as well as a discussion of the progression of database technologies over time and a brief history of data models. These chapters have been updated to introduce some of the newer technologies such as NOSQL systems.
- Part 2 (Chapters 3 and 4) includes the presentation on entity-relationship modeling and database design; however, it is *important to note* that instructors can cover the relational model chapters (Chapters 5 through 8) *before Chapters 3 and 4* if that is their preferred order of presenting the course materials. 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 shows how the basic ER model can be extended to incorporate additional modeling concepts such as subclasses, specialization, generalization, union types (categories) and inheritance, leading to the enhanced-ER (EER) data model and EER diagrams. The notation for the class diagrams of UML are also introduced in Chapters 7 and 8 as an alternative model and diagrammatic notation for ER/EER diagrams.
- Part 3 (Chapters 5 through 8) includes a detailed presentation on relational databases and SQL with some additional new material in the SQL chapters to cover a few SQL constructs that were not in the previous edition. Chapter 5

describes the basic relational model, its integrity constraints, and update operations. Chapter 6 describes some of the basic parts of the SQL standard for relational databases, including data definition, data modification operations, and simple SQL queries. Chapter 7 presents more complex SQL queries, as well as the SQL concepts of triggers, assertions, views, and schema modification. Chapter 8 describes the formal operations of the relational algebra and introduces the relational calculus. The material on SQL (Chapters 6 and 7) is presented before our presentation on relational algebra and calculus in Chapter 8 to allow instructors to start SQL projects early in a course if they wish (it is possible to cover Chapter 8 before Chapters 6 and 7 if the instructor desires this order). The final chapter in Part 2, Chapter 9, covers ER- and EER-to-relational mapping, which are algorithms that can be used for designing a relational database schema from a conceptual ER/EER schema design.

- Part 4 (Chapters 10 and 11) are the chapters on database programming techniques; these chapters can be assigned as reading materials and augmented with materials on the particular language used in the course for programming projects (much of this documentation is readily available on the Web). Chapter 10 covers traditional SQL programming topics, such as embedded SQL, dynamic SQL, ODBC, SQLJ, JDBC, and SQL/CLI. Chapter 11 introduces Web database programming, using the PHP scripting language in our examples, and includes new material that discusses Java technologies for Web database programming.
- Part 5 (Chapters 12 and 13) covers the updated material on object-relational and object-oriented databases (Chapter 12) and XML (Chapter 13); both of these chapters now include a presentation of how the SQL standard incorporates object concepts and XML concepts into more recent versions of the SQL standard. Chapter 12 first introduces the concepts for object databases, and then shows how they have been incorporated into the SQL standard in order to add object capabilities to relational database systems. It then covers the ODMG object model standard, and its object definition and query languages. Chapter 13 covers the XML (eXtensible Markup Language) model and languages, and discusses how XML is related to database systems. It presents XML concepts and languages, and compares the XML model to traditional database models. We also show how data can be converted between the XML and relational representations, and the SQL commands for extracting XML documents from relational tables.
- Part 6 (Chapters 14 and 15) are the normalization and relational design theory chapters (we moved all the formal aspects of normalization algorithms to Chapter 15). Chapter 14 defines functional dependencies, and the normal forms that are based on functional dependencies. Chapter 14 also develops a step-by-step intuitive normalization approach, and includes the definitions of multivalued dependencies and join dependencies. Chapter 15 covers normalization theory, and the formalisms, theories,

and algorithms developed for relational database design by normalization, including the relational decomposition algorithms and the relational synthesis algorithms.

- Part 7 (Chapters 16 and 17) contains the chapters on file organizations on disk (Chapter 16) and indexing of database files (Chapter 17). Chapter 16 describes primary methods of organizing files of records on disk, including ordered (sorted), unordered (heap), and hashed files; both static and dynamic hashing techniques for disk files are covered. Chapter 16 has been updated to include materials on buffer management strategies for DBMSs as well as an overview of new storage devices and standards for files and modern storage architectures. Chapter 17 describes indexing techniques for files, including B-tree and B⁺-tree data structures and grid files, and has been updated with new examples and an enhanced discussion on indexing, including how to choose appropriate indexes and index creation during physical design.
- Part 8 (Chapters 18 and 19) includes the chapters on query processing algorithms (Chapter 18) and optimization techniques (Chapter 19); these two chapters have been updated and reorganized from the single chapter that covered both topics in the previous editions and include some of the newer techniques that are used in commercial DBMSs. Chapter 18 presents algorithms for searching for records on disk files, and for joining records from two files (tables), as well as for other relational operations. Chapter 18 contains new material, including a discussion of the semi-join and anti-join operations with examples of how they are used in query processing, as well as a discussion of techniques for selectivity estimation. Chapter 19 covers techniques for query optimization using cost estimation and heuristic rules; it includes new material on nested subquery optimization, use of histograms, physical optimization, and join ordering methods and optimization of typical queries in data warehouses.
- Part 9 (Chapters 20, 21, and 22) covers transaction processing concepts; concurrency control; and database recovery from failures. These chapters have been updated to include some of the newer techniques that are used in some commercial and open source DBMSs. Chapter 20 introduces the techniques needed for transaction processing systems, and defines the concepts of recoverability and serializability of schedules; it has a new section on buffer replacement policies for DBMSs and a new discussion on the concept of snapshot isolation. Chapter 21 gives an overview of the various types of concurrency control protocols, with a focus on two-phase locking. We also discuss timestamp ordering and optimistic concurrency control techniques, as well as multiple-granularity locking. Chapter 21 includes a new presentation of concurrency control methods that are based on the snapshot isolation concept. Finally, Chapter 23 focuses on database recovery protocols, and gives an overview of the concepts and techniques that are used in recovery.

- Part 10 (Chapters 23, 24, and 25) includes the chapter on distributed databases (Chapter 23), plus the two new chapters on NOSQL storage systems for big data (Chapter 24) and big data technologies based on Hadoop and MapReduce (Chapter 25). Chapter 23 introduces distributed database concepts, including availability and scalability, replication and fragmentation of data, maintaining data consistency among replicas, and many other concepts and techniques. In Chapter 24, NOSQL systems are categorized into four general categories with an example system in each category used for our examples, and the data models, operations, as well as the replication/distribution/scalability strategies of each type of NOSQL system are discussed and compared. In Chapter 25, the MapReduce programming model for distributed processing of big data is introduced, and then we have presentations of the Hadoop system and HDFS (Hadoop Distributed File System), as well as the Pig and Hive high-level interfaces, and the YARN architecture.
- Part 11 (Chapters 26 through 29) is entitled Advanced Database Models, Systems, and Applications and includes the following materials: Chapter 26 introduces several advanced data models including active databases/triggers (Section 26.1), temporal databases (Section 26.2), spatial databases (Section 26.3), multimedia databases (Section 26.4), and deductive databases (Section 26.5). Chapter 27 discusses information retrieval (IR) and Web search, and includes topics such as IR and keyword-based search, comparing DB with IR, retrieval models, search evaluation, and ranking algorithms. Chapter 28 is an introduction to data mining including overviews of various data mining methods such as associate rule mining, clustering, classification, and sequential pattern discovery. Chapter 29 is an overview of data warehousing including topics such as data warehousing models and operations, and the process of building a data warehouse.
- Part 12 (Chapter 30) includes one chapter on database security, which includes a discussion of SQL commands for discretionary access control (GRANT, REVOKE), as well as mandatory security levels and models for including mandatory access control in relational databases, and a discussion of threats such as SQL injection attacks, as well as other techniques and methods related to data security and privacy.

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 prefers. Appendix B gives some important physical parameters of disks. Appendix C gives an overview of the QBE graphical query language, and Appendixes D and E (available on the book's Companion Website located at <http://www.pearsonhighered.com/elmasri>) cover legacy database systems, based on the hierarchical and network database models. They have been used for more than thirty years as a basis for many commercial database applications and transaction-processing systems.

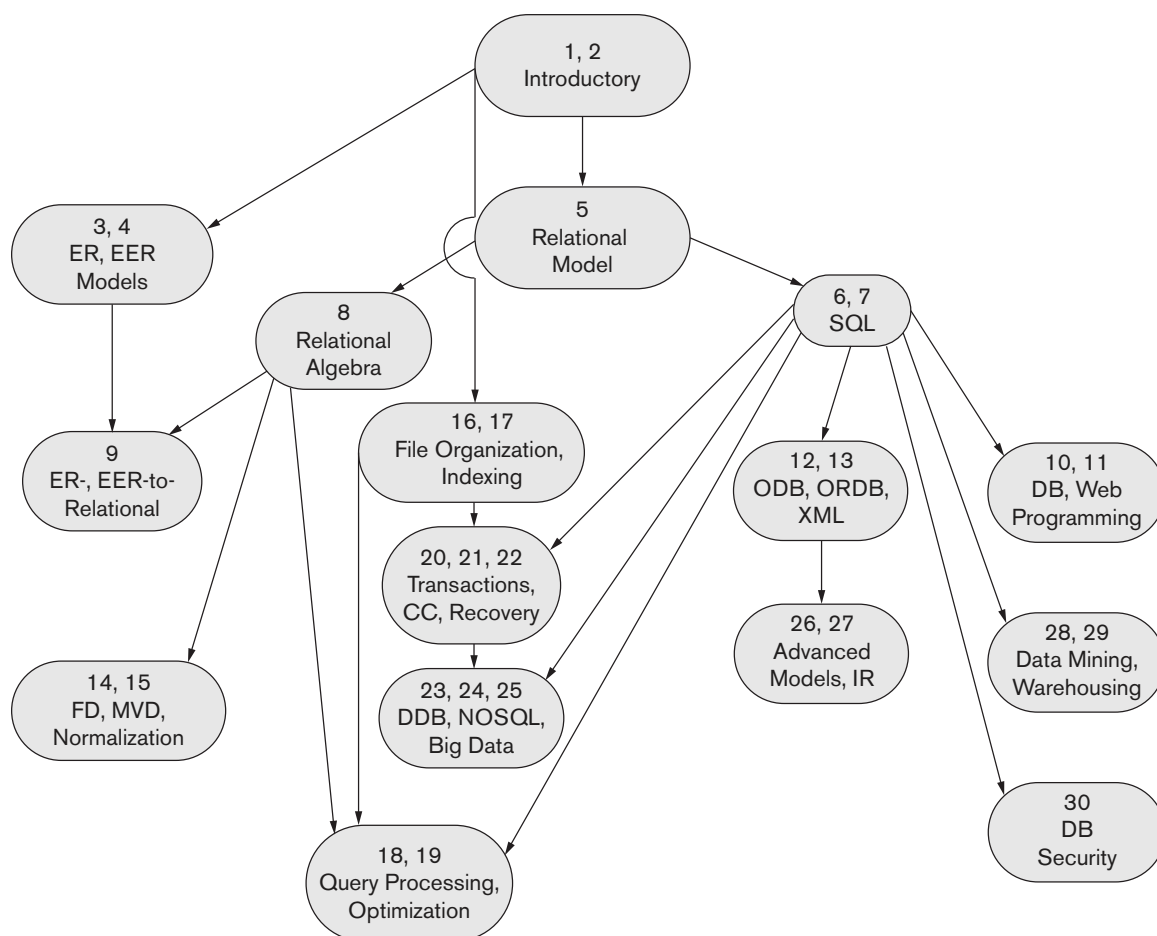
Guidelines for Using This Book

There are many different ways to teach a database course. The chapters in Parts 1 through 7 can be used in an introductory course on database systems in the order that they are given or in the preferred order of individual instructors. 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 the opening section of some of the book's chapters, we list sections that are candidates for being left out whenever a less-detailed discussion of the topic is desired. We suggest covering up to Chapter 15 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 7, 8, and 9 should replace some of the earlier chapters.

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 partially covered, covered later in a course, or even left out if the emphasis is on DBMS implementation. Chapters 16 and 17 on file organizations and indexing may also be covered early, later, or even left out if the emphasis is on database models and languages. For students who have completed a course on file organization, parts of these chapters can be assigned as reading material or some exercises can be assigned as a review for these concepts.

If the emphasis of a course is on database design, then the instructor should cover Chapters 3 and 4 early on, followed by the presentation of relational databases. A total life-cycle database design and implementation project would cover conceptual design (Chapters 3 and 4), relational databases (Chapters 5, 6, and 7), data model mapping (Chapter 9), normalization (Chapter 14), and application programs implementation with SQL (Chapter 10). Chapter 11 also should be covered if the emphasis is on Web database programming and applications. Additional documentation on the specific programming languages and RDBMS used would be required. The book is written so that it is possible to cover topics in various sequences. The following chapter dependency chart shows the major dependencies among 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 one-semester course based on this book, selected chapters can be assigned as reading material. The book also can be used for a two-semester course sequence. The first course, *Introduction to Database Design and Database Systems*, at the sophomore, junior, or senior level, can cover most of Chapters 1 through 15. The second course, *Database Models and Implementation Techniques*, at the senior or first-year graduate level, can cover most of Chapters 16 through 30. The two-semester sequence can also be designed in various other ways, depending on the preferences of the instructors.



Supplemental Materials

Support material is available to qualified instructors at Pearson's instructor resource center (<http://www.pearsonhighered.com/irc>). For access, contact your local Pearson representative.

- PowerPoint lecture notes and figures.
- A solutions manual.

Acknowledgments

It is a great pleasure to acknowledge the assistance and contributions of many individuals to this effort. First, we would like to thank our editor, Matt Goldstein, for his guidance, encouragement, and support. We would like to acknowledge the excellent work of Rose Kernan for production management, Patricia Daly for a

thorough copy editing of the book, Martha McMaster for her diligence in proofing the pages, and Scott Disanno, Managing Editor of the production team. We also wish to thank Kelsey Loanes from Pearson for her continued help with the project, and reviewers Michael Doherty, Deborah Dunn, Imad Rahal, Karen Davis, Gilliean Lee, Leo Mark, Monisha Pulimood, Hassan Reza, Susan Vrbsky, Li Da Xu, Weining Zhang and Vincent Oria.

Ramez Elmasri would like to thank Kulsawasd Jitkajornwanich, Vivek Sharma, and Surya Swaminathan for their help with preparing some of the material in Chapter 24. Sham Navathe would like to acknowledge the following individuals who helped in critically reviewing and revising various topics. Dan Forsythe and Satish Damle for discussion of storage systems; Rafi Ahmed for detailed re-organization of the material on query processing and optimization; Harish Butani, Balaji Palanisamy, and Prajakta Kalmegh for their help with the Hadoop and MapReduce technology material; Vic Ghorpadey and Nenad Jukic for revision of the Data Warehousing material; and finally, Frank Rietta for newer techniques in database security, Kunal Malhotra for various discussions, and Saurav Sahay for advances in information retrieval systems.

We would like to repeat our thanks to those who have reviewed and contributed to previous editions of *Fundamentals of Database Systems*.

- **First edition.** Alan Apt (editor), Don Batory, Scott Downing, Dennis Heimbinger, Julia Hodges, Yannis Ioannidis, Jim Larson, Per-Ake Larson, Dennis McLeod, Rahul Patel, Nicholas Roussopoulos, David Stemple, Michael Stonebraker, Frank Tompa, and Kyu-Young Whang.
- **Second edition.** Dan Joraanstad (editor), Rafi Ahmed, Antonio Albano, David Beech, Jose Blakeley, Panos Chrysanthis, Suzanne Dietrich, Vic Ghorpadey, Goetz 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.
- **Third edition.** Maite Suarez-Rivas and Katherine Harutunian (editors); 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.
- **Fourth edition.** Maite Suarez-Rivas, Katherine Harutunian, Daniel Rausch, and Juliet Silveri (editors); Phil Bernhard, Zhengxin Chen, Jan Chomicki, Hakan Ferhatosmanoglu, Len Fisk, William Hankley, Ali R. Hurson, Vijay Kumar, Peretz Shoval, Jason T. L. Wang (reviewers); Ed Omiecinski (who contributed to Chapter 27). Contributors from the University of Texas at

Arlington are Jack Fu, Hyoil Han, Babak Hojabri, Charley Li, Ande Swathi, and Steven Wu; Contributors from Georgia Tech are Weimin Feng, Dan Forsythe, Angshuman Guin, Abrar Ul-Haque, Bin Liu, Ying Liu, Wanxia Xie, and Waigen Yee.

- **Fifth edition.** Matt Goldstein and Katherine Harutunian (editors); Michelle Brown, Gillian Hall, Patty Mahtani, Maite Suarez-Rivas, Bethany Tidd, and Joyce Cosentino Wells (from Addison-Wesley); Hani Abu-Salem, Jamal R. Alsabbagh, Ramzi Bualuan, Soon Chung, Sumali Conlon, Hasan Davulcu, James Geller, Le Gruenwald, Latifur Khan, Herman Lam, Byung S. Lee, Donald Sanderson, Jamil Saquer, Costas Tsatsoulis, and Jack C. Wileden (reviewers); Raj Sunderraman (who contributed the laboratory projects); Salman Azar (who contributed some new exercises); Gaurav Bhatia, Fariborz Farahmand, Ying Liu, Ed Omiecinski, Nalini Polavarapu, Liora Sahar, Saurav Sahay, and Wanxia Xie (from Georgia Tech).
- **Sixth edition.** Matt Goldstein (editor); Gillian Hall (production management); Rebecca Greenberg (copy editing); Jeff Holcomb, Marilyn Lloyd, Margaret Waples, and Chelsea Bell (from Pearson); Rafi Ahmed, Venu Dasigi, Neha Deodhar, Fariborz Farahmand, Hariprasad Kumar, Leo Mark, Ed Omiecinski, Balaji Palanisamy, Nalini Polavarapu, Parimala R. Pranesh, Bharath Rengarajan, Liora Sahar, Saurav Sahay, Narsi Srinivasan, and Wanxia Xie.

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

R. E.

S.B.N.

This page intentionally left blank

Contents

Preface vii

About the Authors xxx

■ part 1

Introduction to Databases ■

chapter 1 Databases and Database Users 3

- 1.1 Introduction 4
- 1.2 An Example 6
- 1.3 Characteristics of the Database Approach 10
- 1.4 Actors on the Scene 15
- 1.5 Workers behind the Scene 17
- 1.6 Advantages of Using the DBMS Approach 17
- 1.7 A Brief History of Database Applications 23
- 1.8 When Not to Use a DBMS 27
- 1.9 Summary 27
- Review Questions 28
- Exercises 28
- Selected Bibliography 29

chapter 2 Database System Concepts and Architecture 31

- 2.1 Data Models, Schemas, and Instances 32
- 2.2 Three-Schema Architecture and Data Independence 36
- 2.3 Database Languages and Interfaces 38
- 2.4 The Database System Environment 42
- 2.5 Centralized and Client/Server Architectures for DBMSs 46
- 2.6 Classification of Database Management Systems 51
- 2.7 Summary 54
- Review Questions 55
- Exercises 55
- Selected Bibliography 56

■ part 2

Conceptual Data Modeling and Database Design ■

chapter 3 Data Modeling Using the Entity–Relationship (ER) Model 59

- 3.1 Using High-Level Conceptual Data Models for Database Design 60
- 3.2 A Sample Database Application 62
- 3.3 Entity Types, Entity Sets, Attributes, and Keys 63
- 3.4 Relationship Types, Relationship Sets, Roles, and Structural Constraints 72
- 3.5 Weak Entity Types 79
- 3.6 Refining the ER Design for the COMPANY Database 80
- 3.7 ER Diagrams, Naming Conventions, and Design Issues 81
- 3.8 Example of Other Notation: UML Class Diagrams 85
- 3.9 Relationship Types of Degree Higher than Two 88
- 3.10 Another Example: A UNIVERSITY Database 92
- 3.11 Summary 94
- Review Questions 96
- Exercises 96
- Laboratory Exercises 103
- Selected Bibliography 104

chapter 4 The Enhanced Entity–Relationship (EER) Model 107

- 4.1 Subclasses, Superclasses, and Inheritance 108
- 4.2 Specialization and Generalization 110
- 4.3 Constraints and Characteristics of Specialization and Generalization Hierarchies 113
- 4.4 Modeling of UNION Types Using Categories 120
- 4.5 A Sample UNIVERSITY EER Schema, Design Choices, and Formal Definitions 122
- 4.6 Example of Other Notation: Representing Specialization and Generalization in UML Class Diagrams 127
- 4.7 Data Abstraction, Knowledge Representation, and Ontology Concepts 128
- 4.8 Summary 135
- Review Questions 135
- Exercises 136
- Laboratory Exercises 143
- Selected Bibliography 146

■ part 3

The Relational Data Model and SQL ■

chapter	5	The Relational Data Model and Relational Database Constraints	149
	5.1	Relational Model Concepts	150
	5.2	Relational Model Constraints and Relational Database Schemas	157
	5.3	Update Operations, Transactions, and Dealing with Constraint Violations	165
	5.4	Summary	169
		Review Questions	170
		Exercises	170
		Selected Bibliography	175
chapter	6	Basic SQL	177
	6.1	SQL Data Definition and Data Types	179
	6.2	Specifying Constraints in SQL	184
	6.3	Basic Retrieval Queries in SQL	187
	6.4	INSERT, DELETE, and UPDATE Statements in SQL	198
	6.5	Additional Features of SQL	201
	6.6	Summary	202
		Review Questions	203
		Exercises	203
		Selected Bibliography	205
chapter	7	More SQL: Complex Queries, Triggers, Views, and Schema Modification	207
	7.1	More Complex SQL Retrieval Queries	207
	7.2	Specifying Constraints as Assertions and Actions as Triggers	225
	7.3	Views (Virtual Tables) in SQL	228
	7.4	Schema Change Statements in SQL	232
	7.5	Summary	234
		Review Questions	236
		Exercises	236
		Selected Bibliography	238
chapter	8	The Relational Algebra and Relational Calculus	239
	8.1	Unary Relational Operations: SELECT and PROJECT	241
	8.2	Relational Algebra Operations from Set Theory	246

8.3 Binary Relational Operations: JOIN and DIVISION	251
8.4 Additional Relational Operations	259
8.5 Examples of Queries in Relational Algebra	265
8.6 The Tuple Relational Calculus	268
8.7 The Domain Relational Calculus	277
8.8 Summary	279
Review Questions	280
Exercises	281
Laboratory Exercises	286
Selected Bibliography	288

chapter **9** Relational Database Design by ER- and EER-to-Relational Mapping 289

9.1 Relational Database Design Using ER-to-Relational Mapping	290
9.2 Mapping EER Model Constructs to Relations	298
9.3 Summary	303
Review Questions	303
Exercises	303
Laboratory Exercises	305
Selected Bibliography	306

■ part 4

Database Programming Techniques ■

chapter **10** Introduction to SQL Programming Techniques 309

10.1 Overview of Database Programming Techniques and Issues	310
10.2 Embedded SQL, Dynamic SQL, and SQLJ	314
10.3 Database Programming with Function Calls and Class Libraries: SQL/CLI and JDBC	326
10.4 Database Stored Procedures and SQL/PSM	335
10.5 Comparing the Three Approaches	338
10.6 Summary	339
Review Questions	340
Exercises	340
Selected Bibliography	341

chapter **11** Web Database Programming Using PHP 343

11.1 A Simple PHP Example	344
11.2 Overview of Basic Features of PHP	346

11.3 Overview of PHP Database Programming	353
11.4 Brief Overview of Java Technologies for Database Web Programming	358
11.5 Summary	358
Review Questions	359
Exercises	359
Selected Bibliography	359

■ part 5

Object, Object-Relational, and XML: Concepts, Models, Languages, and Standards ■

chapter **12** Object and Object-Relational Databases 363

12.1 Overview of Object Database Concepts	365
12.2 Object Database Extensions to SQL	379
12.3 The ODMG Object Model and the Object Definition Language ODL	386
12.4 Object Database Conceptual Design	405
12.5 The Object Query Language OQL	408
12.6 Overview of the C++ Language Binding in the ODMG Standard	417
12.7 Summary	418
Review Questions	420
Exercises	421
Selected Bibliography	422

chapter **13** XML: Extensible Markup Language 425

13.1 Structured, Semistructured, and Unstructured Data	426
13.2 XML Hierarchical (Tree) Data Model	430
13.3 XML Documents, DTD, and XML Schema	433
13.4 Storing and Extracting XML Documents from Databases	442
13.5 XML Languages	443
13.6 Extracting XML Documents from Relational Databases	447
13.7 XML/SQL: SQL Functions for Creating XML Data	453
13.8 Summary	455
Review Questions	456
Exercises	456
Selected Bibliography	456

■ part 6

Database Design Theory and Normalization ■

chapter **14** Basics of Functional Dependencies and Normalization for Relational Databases 459

- 14.1 Informal Design Guidelines for Relation Schemas 461
- 14.2 Functional Dependencies 471
- 14.3 Normal Forms Based on Primary Keys 474
- 14.4 General Definitions of Second and Third Normal Forms 483
- 14.5 Boyce-Codd Normal Form 487
- 14.6 Multivalued Dependency and Fourth Normal Form 491
- 14.7 Join Dependencies and Fifth Normal Form 494
- 14.8 Summary 495
- Review Questions 496
- Exercises 497
- Laboratory Exercises 501
- Selected Bibliography 502

chapter **15** Relational Database Design Algorithms and Further Dependencies 503

- 15.1 Further Topics in Functional Dependencies: Inference Rules, Equivalence, and Minimal Cover 505
- 15.2 Properties of Relational Decompositions 513
- 15.3 Algorithms for Relational Database Schema Design 519
- 15.4 About Nulls, Dangling Tuples, and Alternative Relational Designs 523
- 15.5 Further Discussion of Multivalued Dependencies and 4NF 527
- 15.6 Other Dependencies and Normal Forms 530
- 15.7 Summary 533
- Review Questions 534
- Exercises 535
- Laboratory Exercises 536
- Selected Bibliography 537

■ **part 7**

File Structures, Hashing, Indexing, and Physical Database Design ■

chapter 16 Disk Storage, Basic File Structures, Hashing, and Modern Storage Architectures 541

- 16.1 Introduction 542
- 16.2 Secondary Storage Devices 547
- 16.3 Buffering of Blocks 556
- 16.4 Placing File Records on Disk 560
- 16.5 Operations on Files 564
- 16.6 Files of Unordered Records (Heap Files) 567
- 16.7 Files of Ordered Records (Sorted Files) 568
- 16.8 Hashing Techniques 572
- 16.9 Other Primary File Organizations 582
- 16.10 Parallelizing Disk Access Using RAID Technology 584
- 16.11 Modern Storage Architectures 588
- 16.12 Summary 592
- Review Questions 593
- Exercises 595
- Selected Bibliography 598

chapter 17 Indexing Structures for Files and Physical Database Design 601

- 17.1 Types of Single-Level Ordered Indexes 602
- 17.2 Multilevel Indexes 613
- 17.3 Dynamic Multilevel Indexes Using B-Trees and B⁺-Trees 617
- 17.4 Indexes on Multiple Keys 631
- 17.5 Other Types of Indexes 633
- 17.6 Some General Issues Concerning Indexing 638
- 17.7 Physical Database Design in Relational Databases 643
- 17.8 Summary 646
- Review Questions 647
- Exercises 648
- Selected Bibliography 650

■ part 8

Query Processing and Optimization ■

chapter **18** Strategies for Query Processing 655

- 18.1 Translating SQL Queries into Relational Algebra and Other Operators 657
- 18.2 Algorithms for External Sorting 660
- 18.3 Algorithms for SELECT Operation 663
- 18.4 Implementing the JOIN Operation 668
- 18.5 Algorithms for PROJECT and Set Operations 676
- 18.6 Implementing Aggregate Operations and Different Types of JOINS 678
- 18.7 Combining Operations Using Pipelining 681
- 18.8 Parallel Algorithms for Query Processing 683
- 18.9 Summary 688
- Review Questions 688
- Exercises 689
- Selected Bibliography 689

chapter **19** Query Optimization 691

- 19.1 Query Trees and Heuristics for Query Optimization 692
- 19.2 Choice of Query Execution Plans 701
- 19.3 Use of Selectivities in Cost-Based Optimization 710
- 19.4 Cost Functions for SELECT Operation 714
- 19.5 Cost Functions for the JOIN Operation 717
- 19.6 Example to Illustrate Cost-Based Query Optimization 726
- 19.7 Additional Issues Related to Query Optimization 728
- 19.8 An Example of Query Optimization in Data Warehouses 731
- 19.9 Overview of Query Optimization in Oracle 733
- 19.10 Semantic Query Optimization 737
- 19.11 Summary 738
- Review Questions 739
- Exercises 740
- Selected Bibliography 740

■ part 9

Transaction Processing, Concurrency Control, and Recovery ■

chapter **20** Introduction to Transaction Processing Concepts and Theory 745

- 20.1 Introduction to Transaction Processing 746
- 20.2 Transaction and System Concepts 753
- 20.3 Desirable Properties of Transactions 757
- 20.4 Characterizing Schedules Based on Recoverability 759
- 20.5 Characterizing Schedules Based on Serializability 763
- 20.6 Transaction Support in SQL 773
- 20.7 Summary 776
- Review Questions 777
- Exercises 777
- Selected Bibliography 779

chapter **21** Concurrency Control Techniques 781

- 21.1 Two-Phase Locking Techniques for Concurrency Control 782
- 21.2 Concurrency Control Based on Timestamp Ordering 792
- 21.3 Multiversion Concurrency Control Techniques 795
- 21.4 Validation (Optimistic) Techniques and Snapshot Isolation Concurrency Control 798
- 21.5 Granularity of Data Items and Multiple Granularity Locking 800
- 21.6 Using Locks for Concurrency Control in Indexes 805
- 21.7 Other Concurrency Control Issues 806
- 21.8 Summary 807
- Review Questions 808
- Exercises 809
- Selected Bibliography 810

chapter **22** Database Recovery Techniques 813

- 22.1 Recovery Concepts 814
- 22.2 NO-UNDO/REDO Recovery Based on Deferred Update 821
- 22.3 Recovery Techniques Based on Immediate Update 823

22.4 Shadow Paging	826
22.5 The ARIES Recovery Algorithm	827
22.6 Recovery in Multidatabase Systems	831
22.7 Database Backup and Recovery from Catastrophic Failures	832
22.8 Summary	833
Review Questions	834
Exercises	835
Selected Bibliography	838

■ part 10

Distributed Databases, NOSQL Systems, and Big Data ■

chapter **23** Distributed Database Concepts 841

23.1 Distributed Database Concepts	842
23.2 Data Fragmentation, Replication, and Allocation Techniques for Distributed Database Design	847
23.3 Overview of Concurrency Control and Recovery in Distributed Databases	854
23.4 Overview of Transaction Management in Distributed Databases	857
23.5 Query Processing and Optimization in Distributed Databases	859
23.6 Types of Distributed Database Systems	865
23.7 Distributed Database Architectures	868
23.8 Distributed Catalog Management	875
23.9 Summary	876
Review Questions	877
Exercises	878
Selected Bibliography	880

chapter **24** NOSQL Databases and Big Data Storage Systems 883

24.1 Introduction to NOSQL Systems	884
24.2 The CAP Theorem	888
24.3 Document-Based NOSQL Systems and MongoDB	890
24.4 NOSQL Key-Value Stores	895
24.5 Column-Based or Wide Column NOSQL Systems	900
24.6 NOSQL Graph Databases and Neo4j	903
24.7 Summary	909
Review Questions	909
Selected Bibliography	910

chapter **25** Big Data Technologies Based on MapReduce and Hadoop 911

- 25.1 What Is Big Data? 914
- 25.2 Introduction to MapReduce and Hadoop 916
- 25.3 Hadoop Distributed File System (HDFS) 921
- 25.4 MapReduce: Additional Details 926
- 25.5 Hadoop v2 alias YARN 936
- 25.6 General Discussion 944
- 25.7 Summary 953
- Review Questions 954
- Selected Bibliography 956

■ part **1**

Advanced Database Models, Systems, and Applications ■

chapter **26** Enhanced Data Models: Introduction to Active, Temporal, Spatial, Multimedia, and Deductive Databases 961

- 26.1 Active Database Concepts and Triggers 963
- 26.2 Temporal Database Concepts 974
- 26.3 Spatial Database Concepts 987
- 26.4 Multimedia Database Concepts 994
- 26.5 Introduction to Deductive Databases 999
- 26.6 Summary 1012
- Review Questions 1014
- Exercises 1015
- Selected Bibliography 1018

chapter **27** Introduction to Information Retrieval and Web Search 1021

- 27.1 Information Retrieval (IR) Concepts 1022
- 27.2 Retrieval Models 1029
- 27.3 Types of Queries in IR Systems 1035
- 27.4 Text Preprocessing 1037
- 27.5 Inverted Indexing 1040
- 27.6 Evaluation Measures of Search Relevance 1044
- 27.7 Web Search and Analysis 1047

27.8 Trends in Information Retrieval	1057
27.9 Summary	1063
Review Questions	1064
Selected Bibliography	1066

chapter **28** Data Mining Concepts 1069

28.1 Overview of Data Mining Technology	1070
28.2 Association Rules	1073
28.3 Classification	1085
28.4 Clustering	1088
28.5 Approaches to Other Data Mining Problems	1091
28.6 Applications of Data Mining	1094
28.7 Commercial Data Mining Tools	1094
28.8 Summary	1097
Review Questions	1097
Exercises	1098
Selected Bibliography	1099

chapter **29** Overview of Data Warehousing and OLAP 1101

29.1 Introduction, Definitions, and Terminology	1102
29.2 Characteristics of Data Warehouses	1103
29.3 Data Modeling for Data Warehouses	1105
29.4 Building a Data Warehouse	1111
29.5 Typical Functionality of a Data Warehouse	1114
29.6 Data Warehouse versus Views	1115
29.7 Difficulties of Implementing Data Warehouses	1116
29.8 Summary	1117
Review Questions	1117
Selected Bibliography	1118

■ part **12**

Additional Database Topics: Security ■

chapter **30** Database Security 1121

30.1 Introduction to Database Security Issues	1122
30.2 Discretionary Access Control Based on Granting and Revoking Privileges	1129
30.3 Mandatory Access Control and Role-Based Access Control for Multilevel Security	1134

30.4 SQL Injection	1143
30.5 Introduction to Statistical Database Security	1146
30.6 Introduction to Flow Control	1147
30.7 Encryption and Public Key Infrastructures	1149
30.8 Privacy Issues and Preservation	1153
30.9 Challenges to Maintaining Database Security	1154
30.10 Oracle Label-Based Security	1155
30.11 Summary	1158
Review Questions	1159
Exercises	1160
Selected Bibliography	1161

appendix **A** **Alternative Diagrammatic Notations for ER Models** 1163

appendix **B** **Parameters of Disks** 1167

appendix **C** **Overview of the QBE Language** 1171

C.1 Basic Retrievals in QBE	1171
C.2 Grouping, Aggregation, and Database Modification in QBE	1175

appendix **D** **Overview of the Hierarchical Data Model**
(located on the Companion Website at
<http://www.pearsonhighered.com/elmasri>)

appendix **E** **Overview of the Network Data Model**
(located on the Companion Website at
<http://www.pearsonhighered.com/elmasri>)

Selected Bibliography 1179

Index 1215

About the Authors

Ramez Elmasri is a professor and the associate chairperson of the Department of Computer Science and Engineering at the University of Texas at Arlington. He has over 140 refereed research publications, and has supervised 16 PhD students and over 100 MS students. His research has covered many areas of database management and big data, including conceptual modeling and data integration, query languages and indexing techniques, temporal and spatio-temporal databases, bio-informatics databases, data collection from sensor networks, and mining/analysis of spatial and spatio-temporal data. He has worked as a consultant to various companies, including Digital, Honeywell, Hewlett Packard, and Action Technologies, as well as consulting with law firms on patents. He was the Program Chair of the 1993 International Conference on Conceptual Modeling (ER conference) and program vice-chair of the 1994 IEEE International Conference on Data Engineering. He has served on the ER conference steering committee and has been on the program committees of many conferences. He has given several tutorials at the VLDB, ICDE, and ER conferences. He also co-authored the book “Operating Systems: A Spiral Approach” (McGraw-Hill, 2009) with Gil Carrick and David Levine. Elmasri is a recipient of the UTA College of Engineering Outstanding Teaching Award in 1999. He holds a BS degree in Engineering from Alexandria University, and MS and PhD degrees in Computer Science from Stanford University.

Shamkant B. Navathe is a professor and the founder of the database research group at the College of Computing, Georgia Institute of Technology, Atlanta. He has worked with IBM and Siemens in their research divisions and has been a consultant to various companies including Digital, Computer Corporation of America, Hewlett Packard, Equifax, and Persistent Systems. He was the General Co-chairman of the 1996 International VLDB (Very Large Data Base) conference in Bombay, India. He was also program co-chair of ACM SIGMOD 1985 International Conference and General Co-chair of the IFIP WG 2.6 Data Semantics Workshop in 1995. He has served on the VLDB foundation and has been on the steering committees of several conferences. He has been an associate editor of a number of journals including *ACM Computing Surveys*, and *IEEE Transactions on Knowledge and Data Engineering*. He also co-authored the book “Conceptual Design: An Entity Relationship Approach” (Addison Wesley, 1992) with Carlo Batini and Stefano Ceri. Navathe is a fellow of the Association for Computing Machinery (ACM) and recipient of the IEEE TCDE Computer Science, Engineering and Education Impact award in 2015. Navathe holds a PhD from the University of Michigan and has over 150 refereed publications in journals and conferences.

part 1

Introduction to Databases

This page intentionally left blank

chapter 1

Databases and Database Users

Databases and database systems are an essential component of life in modern society: most of us encounter several activities every day 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 purchase something online—such as a book, toy, or computer—chances are that our activities will involve someone or some computer program accessing a database. Even purchasing items at a supermarket often automatically updates the database that holds the inventory of grocery 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 led to exciting new applications of database systems. The proliferation of social media Web sites, such as Facebook, Twitter, and Flickr, among many others, has required the creation of huge databases that store nontraditional data, such as posts, tweets, images, and video clips. New types of database systems, often referred to as **big data** storage systems, or **NOSQL systems**, have been created to manage data for social media applications. These types of systems are also used by companies such as Google, Amazon, and Yahoo, to manage the data required in their Web search engines, as well as to provide **cloud storage**, whereby users are provided with storage capabilities on the Web for managing all types of data including documents, programs, images, videos and emails. We will give an overview of these new types of database systems in Chapter 24.

We now mention some other applications of databases. The wide availability of photo and video technology on cellphones and other devices has made it possible to

store images, audio clips, and video streams digitally. These types of files are becoming an important component of **multimedia databases**. **Geographic information systems (GISs)** 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 business information from very large databases to support decision making. **Real-time** and **active database technology** is used to control 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. In Section 1.1 we start by defining a database, and then we explain 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 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 have had 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, social media, engineering, medicine, genetics, law, education, and library science. The word *database* is so commonly used 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. Nowadays, this data is typically stored in mobile phones, which have their own simple database software. This data can also be recorded in an indexed address book or stored on a hard drive, using a personal computer and software such as Microsoft Access or Excel. This collection of related data with an implicit meaning 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

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

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.

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 its contents. The end users of a database may perform business transactions (for example, a customer buys a camera) or events may happen (for example, an employee has a baby) that cause the information in the database to change. In order for a database to be accurate and reliable at all times, it must be a true reflection of the miniworld that it represents; therefore, changes must be reflected in the database as soon as possible.

A database can be of any size and 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 alphabetically. A database of even greater size and complexity would be maintained by a social media company such as Facebook, which has more than a billion users. The database has to maintain information on which users are related to one another as *friends*, the postings of each user, which users are allowed to see each posting, and a vast amount of other types of information needed for the correct operation of their Web site. For such Web sites, a large number of databases are needed to keep track of the constantly changing information required by the social media Web site.

An example of a large commercial database is Amazon.com. It contains data for over 60 million active users, and millions of books, CDs, videos, DVDs, games, electronics, apparel, and other items. The database occupies over 42 terabytes (a terabyte is 10^{12} bytes worth of storage) and is stored on hundreds of computers (called servers). Millions of visitors access Amazon.com each day and use the database to make purchases. The database is continually updated as new books and other items are added to the inventory, and stock quantities are updated as purchases are transacted.

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 text.

A **database management system (DBMS)** is a computerized system that enables users to create and maintain a database. The DBMS is 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 of the data to be stored in the database. The database definition or descriptive information is also stored by the DBMS in the form of a database catalog or dictionary; it is called **meta-data**. **Constructing** the database is the process of storing the data on some storage medium that is controlled by the DBMS. **Manipulating** a database includes functions such 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 simultaneously.

An **application program** accesses the database by sending queries or requests for data to the DBMS. A **query**² typically causes some data to be retrieved; a **transaction** may cause some data to be read and some data to be written into the database.

Other important functions provided by the DBMS include *protecting* the database and *maintaining* it over a long period of time. **Protection** includes *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.

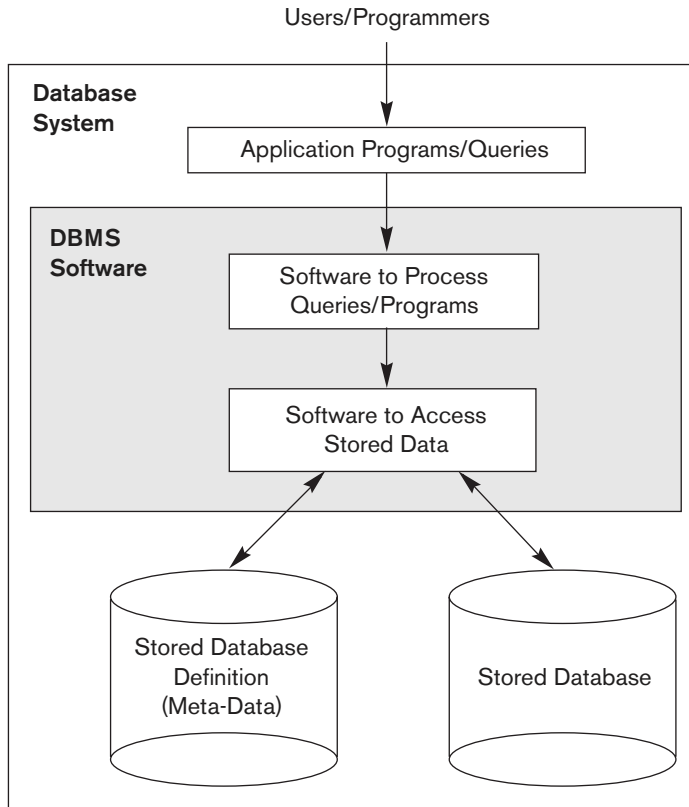
It is not absolutely necessary to use general-purpose DBMS software to implement a computerized database. It is possible to write a customized set of programs to create and maintain the database, in effect creating a *special-purpose* DBMS software for a specific application, such as airlines reservations. In either case—whether we use a general-purpose DBMS or not—a considerable amount of complex software is deployed. 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 have discussed so far.

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 records. The database is organized as five files, each of which

²The term *query*, originally meaning a question or an inquiry, is sometimes loosely used for all types of interactions with databases, including modifying the data.

**Figure 1.1**

A simplified database system environment.

stores **data records** of the same type.³ 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, Student_number, Class (such as freshman or '1', sophomore or '2', and so forth), and Major (such as mathematics or 'MATH' and computer science or 'CS'); each COURSE record includes data to represent the Course_name, Course_number, Credit_hours, 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, Student_number of STUDENT is an integer, and Grade of GRADE_REPORT is a

³We 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	Student_number	Class	Major
Smith	17	1	CS
Brown	8	2	CS

COURSE

Course_name	Course_number	Credit_hours	Department
Intro to Computer Science	CS1310	4	CS
Data Structures	CS3320	4	CS
Discrete Mathematics	MATH2410	3	MATH
Database	CS3380	3	CS

SECTION

Section_identifier	Course_number	Semester	Year	Instructor
85	MATH2410	Fall	07	King
92	CS1310	Fall	07	Anderson
102	CS3320	Spring	08	Knuth
112	MATH2410	Fall	08	Chang
119	CS1310	Fall	08	Anderson
135	CS3380	Fall	08	Stone

GRADE_REPORT

Student_number	Section_identifier	Grade
17	112	B
17	119	C
8	85	A
8	92	A
8	102	B
8	135	A

PREREQUISITE

Course_number	Prerequisite_number
CS3380	CS3320
CS3380	MATH2410
CS3320	CS1310

Figure 1.2

A database that stores student and course information.

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 as follows:

- 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 2008 and their grades in that section
- List the prerequisites of the 'Database' course

Examples of updates include the following:

- Change the class of 'Smith' to sophomore
- Create a new section for the 'Database' course for this semester
- 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.

At this stage, it is useful to describe the database as part of a larger undertaking known as an information system within an organization. The Information Technology (IT) department within an organization designs and maintains an information system consisting of various computers, storage systems, application software, and databases. Design of a new application for an existing database or design of a brand new database starts off with a phase called **requirements specification and analysis**. These requirements are documented in detail and transformed into a **conceptual design** that can be represented and manipulated using some computerized tools so that it can be easily maintained, modified, and transformed into a database implementation. (We will introduce a model called the Entity-Relationship model in Chapter 3 that is used for this purpose.) The design is then translated to a **logical design** that can be expressed in a data model implemented in a commercial DBMS. (Various types of DBMSs are discussed throughout the text, with an emphasis on relational DBMSs in Chapters 5 through 9.)

The final stage is **physical design**, during which further specifications are provided for storing and accessing the database. The database design is implemented, populated with actual data, and continuously maintained to reflect the state of the miniworld.

1.3 Characteristics of the Database Approach

A number of characteristics distinguish the database approach from the much older approach of writing customized programs to access data stored in 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 files on students and their grades. Programs to print a student's transcript and to enter new grades 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 up-to-date data.

In the database approach, a single repository maintains data that is defined once and then accessed by various users repeatedly through queries, transactions, and application programs. 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 describe each of these characteristics in a separate section. We will discuss additional characteristics of database systems 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). It is important to note that some newer types of database systems, known as NOSQL systems, do not require meta-data. Rather the data is stored as **self-describing data** that includes the data item names and data values together in one structure (see Chapter 24).

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. Therefore, 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. Whereas file-processing software can access only specific databases, DBMS software can access diverse databases by extracting the database definitions from the catalog and using these definitions.

For the example shown in Figure 1.2, the DBMS catalog will store the definitions of all the files shown. Figure 1.3 shows some entries in a database 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.

RELATIONS

Relation_name	No_of_columns
STUDENT	4
COURSE	4
SECTION	5
GRADE_REPORT	3
PREREQUISITE	2

Figure 1.3
An example of a database catalog for the database in Figure 1.2.

COLUMNS

Column_name	Data_type	Belongs_to_relation
Name	Character (30)	STUDENT
Student_number	Character (4)	STUDENT
Class	Integer (1)	STUDENT
Major	Major_type	STUDENT
Course_name	Character (10)	COURSE
Course_number	XXXXNNNN	COURSE
....
....
....
Prerequisite_number	XXXXNNNN	PREREQUISITE

Note: Major_type is defined as an enumerated type with all known majors. XXXXNNNN is used to define a type with four alphabetic characters followed by four numeric digits.

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 that 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 1.4. If we want to add another piece of data to each STUDENT record, say the Birth_date, such a program will no longer work and must be changed. By contrast, in a DBMS environment, we only need to change the *description* of STUDENT records in the catalog (Figure 1.3) to reflect the inclusion of the new data item Birth_date; 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.

In some types of database systems, such as object-oriented and object-relational systems (see Chapter 12), 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.

Looking at the example in Figures 1.2 and 1.3, the internal implementation of the STUDENT 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 1.4. But a typical database user is not concerned with the location of each data item within a record or its length; rather, the user is concerned 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

Data Item Name	Starting Position in Record	Length in Characters (bytes)
Name	1	30
Student_number	31	4
Class	35	1
Major	36	4

Figure 1.4

Internal storage format for a STUDENT record, based on the database catalog in Figure 1.3.

file—can be hidden from database users by the DBMS; we discuss storage details in Chapters 16 and 17.

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 text 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, 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.

1.3.3 Support of Multiple Views of the Data

A database typically has many types of 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 1.5(a). A second user, who is interested only in checking that students have taken all the prerequisites of each course for which the student registers, may require the view shown in Figure 1.5(b).

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

TRANSCRIPT

Student_name	Student_transcript				
	Course_number	Grade	Semester	Year	Section_id
Smith	CS1310	C	Fall	08	119
	MATH2410	B	Fall	08	112
Brown	MATH2410	A	Fall	07	85
	CS1310	A	Fall	07	92
	CS3320	B	Spring	08	102
	CS3380	A	Fall	08	135

(a)

COURSE_PREREQUISITES

Course_name	Course_number	Prerequisites
Database	CS3380	CS3320
		MATH2410
Data Structures	CS3320	CS1310

(b)

Figure 1.5

Two views derived from the database in Figure 1.2. (a) The TRANSCRIPT view.

(b) The COURSE_PREREQUISITES view.

do so in a controlled manner so that the result of the updates is correct. For example, when several reservation agents try to assign a seat on an airline flight, the DBMS should ensure that each seat can be accessed by only one agent 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 and efficiently.

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 9.

The preceding characteristics are 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 people 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, in large organizations, many people are involved in the design, use, and maintenance of a large database with hundreds or thousands 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 contents as part of their daily job.

1.4.1 Database Administrators

In any organization where many people 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, coordinating and monitoring its use, and acquiring software and hardware resources as needed. The DBA is accountable for problems such as security breaches and poor system response time. In large organizations, the DBA is assisted by a staff that carries 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 create 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 interface

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. Many of these tasks are now available as **mobile apps** for use with mobile devices. The tasks that such users perform are varied. A few examples are:
 - Bank customers and tellers check account balances and post withdrawals and deposits.
 - Reservation agents or customers for airlines, hotels, and car rental companies check availability for a given request and make reservations.
 - Employees at receiving stations for shipping companies enter package identifications via bar codes and descriptive information through buttons to update a central database of received and in-transit packages.
 - Social media users post and read items on social media Web sites.
- **Sophisticated end users** include engineers, scientists, business analysts, and others who thoroughly familiarize themselves with the facilities of the DBMS in order to implement their own applications to meet their complex requirements.
- **Standalone 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 financial software package that stores a variety of personal financial data.

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 simply have to understand the user interfaces of the mobile apps or 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. Standalone 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 standard 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 developers** or **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 content itself. We call them the *workers behind the scene*, and they include the following categories:

- **DBMS system designers and implementers** 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, query language processing, interface processing, 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** design and implement **tools**—the software packages that facilitate database modeling and design, database system design, and improved 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** (system administration personnel) 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 contents for their own purposes.

1.6 Advantages of Using the DBMS Approach

In this section we discuss some additional 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 keeps data on registration and related billing information, whereas the registration office keeps track of student courses and grades. Other 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 birth date erroneously as ‘JAN-19-1988’, whereas the other user groups may enter the correct value of ‘JAN-29-1988’.

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 is known as **data normalization**, and it ensures consistency and saves storage space (data normalization is described in Part 6 of the text).

However, in practice, it is sometimes necessary to use **controlled redundancy** to improve the performance of queries. For example, we may store Student_name and Course_number redundantly in a GRADE_REPORT file (Figure 1.6(a)) 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. This is known as **denormalization**. In such cases, the DBMS should

Figure 1.6
Redundant storage of Student_name and Course_name in GRADE_REPORT.
(a) Consistent data.
(b) Inconsistent record.

(a)	GRADE_REPORT				
	Student_number	Student_name	Section_identifier	Course_number	Grade
	17	Smith	112	MATH2410	B
	17	Smith	119	CS1310	C
	8	Brown	85	MATH2410	A
	8	Brown	92	CS1310	A
(b)	8	Brown	102	CS3320	B
	8	Brown	135	CS3380	A
(b)	GRADE_REPORT				
	Student_number	Student_name	Section_identifier	Course_number	Grade
	17	Brown	112	MATH2410	B

have the capability to *control* this redundancy in order to prohibit inconsistencies among the files. This may be done by automatically checking that the Student_name–Student_number values in any GRADE_REPORT record in Figure 1.6(a) match one of the Name–Student_number values of a STUDENT record (Figure 1.2). Similarly, the Section_identifier–Course_number values in 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 1.6(b) shows a GRADE_REPORT record that is inconsistent with the STUDENT file in Figure 1.2; this kind of error may be entered if the redundancy is *not controlled*. Can you tell which part is inconsistent?

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 such as salaries and bonuses is often considered confidential, and only authorized persons are allowed to access such data. In addition, some users may only be permitted to retrieve data, whereas others are allowed to retrieve and 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. Then, the DBMS should 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 pre-defined apps or canned transactions developed for their use. We discuss database security and authorization in Chapter 30.

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** (see Chapter 12). Programming languages typically have complex data structures, such as structs or class definitions in C++ or Java. The values of program variables or objects 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 or object 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 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 and Search Techniques 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 and search techniques to speed up disk search for the desired records. Auxiliary files called **indexes** are often used for this purpose. Indexes are typically based on tree data structures or hash data structures that are 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 main memory. Therefore, the DBMS often has a **buffering** or **caching** module that maintains parts of the database in main memory buffers. In general, the operating system is responsible for disk-to-memory buffering. However, because data buffering is crucial to the DBMS performance, most DBMSs do their own data buffering.

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. We discuss query processing and optimization in Part 8 of the text.

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. Disk backup is also necessary in case of a catastrophic disk failure. We discuss recovery and backup in Chapter 22.

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 apps for mobile users, 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 standalone 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 and 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, to define new relationships as they arise, and 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 1.3, we specified that the value of the Class data item within each STUDENT record must be a one-digit integer and that the value of Name must be a string of no more than 30 alphabetic characters. To restrict the value of Class between 1 and 5 would be an additional constraint that is not shown in the current catalog. 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*. This is known as a **referential integrity** constraint. Another type of constraint specifies uniqueness on data item values, such as *every course record must have a unique value for Course_number*. This is known as a **key** or **uniqueness** constraint. These constraints are derived from the meaning or **semantics** of the data and of the miniworld it represents. It is the responsibility of the database designers 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. For typical large applications, it is customary to call such constraints **business rules**.

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’ would be rejected automatically by the DBMS because ‘Z’ is not a valid value for the Grade data type. When we discuss each data model in subsequent chapters, we will introduce rules that pertain to

that model implicitly. For example, in the Entity-Relationship model in Chapter 3, a relationship must involve at least two entities. Rules that pertain to a specific data model are called **inherent rules** of the data model.

1.6.9 Permitting Inferencing and Actions Using Rules and Triggers

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. In today's relational database systems, it is possible to associate **triggers** with tables. A trigger is a form of a rule activated by updates to the table, which results in performing some additional operations to some other tables, sending messages, and so on. More involved procedures to enforce rules are popularly called **stored procedures**; they become a part of the overall database definition and are invoked appropriately when certain conditions are met. More powerful functionality is provided by **active database systems**, which provide active rules that can automatically initiate actions when certain events and conditions occur (see Chapter 26 for introductions to active databases in Section 26.1 and deductive databases in Section 26.5).

1.6.10 Additional Implications of Using the Database Approach

This section discusses a few 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 data 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 large multiuser 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 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 as well as redundancies among applications. This enables the whole organization to invest in more powerful processors, storage devices, or networking gear, rather than having each department purchase its own (lower performance) 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. Hence, these systems did not provide sufficient *data abstraction* and *program-data independence* capabilities. 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 laborious to reorganize the database when changes were made to the application's requirements.

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 continuing 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 Data Abstraction and 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 data representation and querying. The relational data model also introduced high-level query languages that provided an alternative to programming language interfaces, making it much faster 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, and provided flexibility to develop new queries quickly and to reorganize the database as requirements changed. Hence, *data abstraction* and *program-data independence* were much improved when compared to earlier systems.

Early experimental relational systems developed in the late 1970s and the commercial relational database management systems (RDBMS) 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 system 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 (OODBs). Initially, OODBs 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. Despite expectations that they will make a big impact, their overall penetration into the database products market remains low. In addition, many object-oriented concepts were incorporated into the newer versions of relational DBMSs, leading to object-relational database management systems, known as ORDBMSs.

1.7.4 Interchanging Data on the Web for E-Commerce Using XML

The World Wide Web provides a large network of interconnected computers. Users can create static Web pages using a Web publishing language, such as Hyper-Text Markup Language (HTML), and store these documents on Web servers where other users (clients) can access them and view them through Web browsers. Documents can be linked through **hyperlinks**, which are pointers to other documents. Starting in the 1990s, electronic commerce (e-commerce) emerged as a major application on the Web. Much of the critical information on e-commerce Web pages is dynamically extracted data from DBMSs, such as flight information, product prices, and product availability. A variety of techniques were developed to allow the interchange of dynamically extracted data on the Web for display on Web pages. The eXtended Markup Language (XML) is one 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. Chapter 13 is devoted to an overview of XML.

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 software and file and data structures. Database systems now offer extensions to better support the specialized requirements for some of these applications. The following are some examples of these applications:

- **Scientific** applications that store large amounts of data resulting from scientific experiments in areas such as high-energy physics, the mapping of the human genome, and the discovery of protein structures
- Storage and retrieval of **images**, including scanned news or personal photographs, satellite photographic images, and images from medical procedures such as x-rays and MRI (magnetic resonance imaging) tests

- Storage and retrieval of **videos**, such as movies, and **video clips** from news or personal digital cameras
- **Data mining** applications that analyze large amounts of data to search for the occurrences of specific patterns or relationships, and for identifying unusual patterns in areas such as credit card fraud detection
- **Spatial** applications that store and analyze spatial locations of data, such as weather information, maps used in geographical information systems, and automobile navigational systems
- **Time series** applications that store information such as economic data at regular points in time, such as daily sales and 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 for efficient searching on the new data types.

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.

1.7.6 Emergence of Big Data Storage Systems and NOSQL Databases

In the first decade of the twenty-first century, the proliferation of applications and platforms such as social media Web sites, large e-commerce companies, Web search indexes, and cloud storage/backup led to a surge in the amount of data stored on large databases and massive servers. New types of database systems were necessary to manage these huge databases—systems that would provide fast search and retrieval as well as reliable and safe storage of nontraditional types of data, such as social media posts and tweets. Some of the requirements of these new systems were not compatible with SQL relational DBMSs (SQL is the standard data model and language for relational databases). The term *NOSQL* is generally interpreted as Not Only SQL, meaning that in systems that manage large amounts of data, some of the data is stored using SQL systems, whereas other data would be stored using NOSQL, depending on the application requirements.

1.8 When Not to Use a DBMS

In spite of the advantages of using a DBMS, there are a few situations in which a DBMS 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

Therefore, it may be more desirable to develop customized database applications under the following circumstances:

- Simple, well-defined database applications that are not expected to change at all
- Stringent, real-time requirements for some application programs that may not be met because of DBMS overhead
- Embedded systems with limited storage capacity, where a general-purpose DBMS would not fit
- No multiple-user access to data

Certain industries and applications have elected not to use general-purpose DBMSs. For example, many computer-aided design (CAD) tools used by mechanical and civil engineers have proprietary file and data management software that is geared for the internal manipulations of drawings and 3D objects. Similarly, communication and switching systems designed by companies like AT&T were early manifestations of database software that was made to run very fast with hierarchically organized data for quick access and routing of calls. GIS implementations often implement their own data organization schemes for efficiently implementing functions related to processing maps, physical contours, lines, polygons, and so on.

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, and we 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 presented a list of capabilities that should be provided by the DBMS software to the DBA, database designers, and end users to help them design, administer, and use a database. Then we gave a brief historical perspective on the evolution of database applications. We pointed out the recent rapid growth of the amounts and types of data that must be stored in databases, and we discussed the emergence of new systems for handling “big data” applications. Finally, we discussed the overhead costs of using a DBMS and discussed some situations in which it may not be advantageous to use one.

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*, and *transaction-processing application*.
- 1.2. What four 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.
- 1.7. Discuss the differences between database systems and information retrieval systems.

Exercises

- 1.8. Identify some informal queries and update operations that you would expect to apply to the database shown in Figure 1.2.
- 1.9. What is the difference between controlled and uncontrolled redundancy? Illustrate with examples.
- 1.10. Specify all the relationships among the records of the database shown in Figure 1.2.
- 1.11. Give some additional views that may be needed by other user groups for the database shown in Figure 1.2.
- 1.12. Cite some examples of integrity constraints that you think can apply to the database shown in Figure 1.2.
- 1.13. Give examples of systems in which it may make sense to use traditional file processing instead of a database approach.

1.14. Consider Figure 1.2.

- a. If the name of the 'CS' (Computer Science) Department changes to 'CSSE' (Computer Science and Software Engineering) Department and the corresponding prefix for the course number also changes, identify the columns in the database that would need to be updated.
- b. Can you restructure the columns in the COURSE, SECTION, and PREREQUISITE tables so that only one column will need to be updated?

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. We will include references to other concepts, systems, and applications introduced in this chapter in the later text chapters that discuss each topic in more detail.

This page intentionally left blank