**SEVENTH EDITION**

**Database System Concepts**

Abraham Silberschatz 

Henry F. Korth

S. Sudarshan

**DATABASE** SYSTEM CONCEPTS

SIXTH EDITION

SEVENTH EDITION

Abraham Silberschatz

*Yale University*

Henry F. Korth

*Lehigh University*

S. Sudarshan

*Indian Institute of Technology, Bombay*

TM

DATABASE SYSTEM CONCEPTS, SEVENTH EDITION

Published by McGraw-Hill Education, 2 Penn Plaza, New York, NY 10121. Copyright © 2020 by McGraw-Hill Education. All rights reserved. Printed in the United States of America. Previous editions © 2011, 2006, and 2002. No part of this publication may be reproduced or distributed in any form or by any means, or stored in a database or retrieval system, without the prior written consent of McGraw-Hill Education, including, but not limited to, in any network or other electronic storage or transmission, or broadcast for distance learning.

Some ancillaries, including electronic and print components, may not be available to customers outside the United States.

This book is printed on acid-free paper.

1 2 3 4 5 6 7 8 9 LCR 21 20 19

ISBN 978-0-07-802215-9 (bound edition)

MHID 0-07-802215-0 (bound edition)

ISBN 978-1-260-51504-6 (loose-leaf edition)

MHID 1-260-51504-4 (loose-leaf edition)

Portfolio Manager: *Thomas Scaife Ph.D.*

Product Developers: *Tina Bower & Megan Platt*

Marketing Manager: *Shannon O’Donnell*

Content Project Managers: *Laura Bies & Sandra Schnee*

Buyer: *Susan K. Culbertson*

Design: *Egzon Shaqiri*

Content Licensing Specialists: *Shawntel Schmitt & Lorraine Buczek*

Cover Image: *© Pavel Nesvadba/Shutterstock*

Compositor: *Aptara®, Inc.*

All credits appearing on page or at the end of the book are considered to be an extension of the copyright page.

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

Names: Silberschatz, Abraham, author. | Korth, Henry F., author. | Sudarshan, S., author. Title: Database system concepts/Abraham Silberschatz, Yale University, Henry F. Korth, Lehigh University, S. Sudarshan, Indian Institute of Technology, Bombay. Description: Seventh edition. | New York, NY: McGraw-Hill, [2020] | Includes bibliographical references.

Identifiers: LCCN 2018060474 | ISBN 9780078022159 (alk. paper) | ISBN 0078022150 (alk. paper) Subjects: LCSH: Database management.

Classification: LCC QA76.9.D3 S5637 2020 | DDC 005.74—dc23 LC record available at https://lccn.loc.gov/2018060474

The Internet addresses listed in the text were accurate at the time of publication. The inclusion of a website does not indicate an endorsement by the authors or McGraw-Hill Education, and McGraw-Hill Education does not guarantee the accuracy of the information presented at these sites.

mheducation.com/highered

*To meine schatzi, Valerie*

*her parents and my dear friends, Steve and Mary Anne and in memory of my parents, Joseph and Vera*

*Avi Silberschatz*

*To my wife, Joan*

*my children, Abigail and Joseph*

*my mother, Frances*

*and in memory of my father, Henry*

*Hank Korth*

*To my wife, Sita*

*my children, Madhur and Advaith*

*and my mother, Indira*

*S. Sudarshan*

About the Authors

Abraham (Avi) Silberschatz is the Sidney J. Weinberg Professor of Computer Science at Yale University. Prior to coming to Yale in 2003, he was the vice president of the Information Sciences Research Center at Bell Labs. He previously held an endowed professorship at the University of Texas at Austin, where he taught until 1993. Silber schatz is a fellow of the ACM, a fellow of the IEEE, and a member of the Connecticut Academy of Science and Engineering. He received the 2002 IEEE Taylor L. Booth Ed ucation Award, the 1998 ACM Karl V. Karlstrom Outstanding Educator Award, and the 1997 ACM SIGMOD Contribution Award. Silberschatz was awarded the Bell Lab oratories President’s Award three times, in 1998, 1999 and 2004. His writings have appeared in numerous journals, conferences, workshops, and book chapters. He has obtained over 48 patents and over 24 grants. He is an author of the textbook Operating System Concepts.

Henry F. (Hank) Korth is a Professor of Computer Science and Engineering and co director of the Computer Science and Business program at Lehigh University. Prior to joining Lehigh, he was director of Database Principles Research at Bell Labs, a vice president of Panasonic Technologies, an associate professor at the University of Texas at Austin, and a research staff member at IBM Research. Korth is a fellow of the ACM and of the IEEE and a winner of the 10-Year Award at the VLDB Conference. His numer ous research publications span a wide range of aspects of database systems, including transaction management in parallel and distributed systems, real-time systems, query processing, and the influence on these areas from modern computing architectures. Most recently, his research has addressed issues in the application of blockchains in enterprise databases.

S. Sudarshan is currently the Subrao M. Nilekani Chair Professor at the Indian Insti tute of Technology, Bombay. He received his Ph.D. at the University of Wisconsin in 1992, and he was a member of the technical staff at Bell Labs before joining IIT Bom bay. Sudarshan is a fellow of the ACM. His research spans several areas of database systems, with a focus on query processing and query optimization. His paper on key word search in databases published in 2002 won the IEEE ICDE Most Influential Paper Award in 2012, and his work on main-memory databases received the Bell Laborato ries President’s Award in 1999. His current research areas include testing and grading of SQL queries, optimization of database applications by rewriting of imperative code, and query optimization for parallel databases. He has published over 100 papers and obtained 15 patents.

Contents

Chapter 1 Introduction

1.1 Database-System Applications 1

1.2 Purpose of Database Systems 5

1.3 View of Data 8

1.4 Database Languages 13

1.5 Database Design 17

1.6 Database Engine 18

1.7 Database and Application Architecture 21 1.8 Database Users and Administrators 24 1.9 History of Database Systems 25 1.10 Summary 29

Exercises 31

Further Reading 33

PART ONE RELATIONAL LANGUAGES Chapter 2 Introduction to the Relational Model

2.1 Structure of Relational Databases 37 2.2 Database Schema 41

2.3 Keys 43

2.4 Schema Diagrams 46

2.5 Relational Query Languages 47

Chapter 3 Introduction to SQL

3.1 Overview of the SQL Query Language 65 3.2 SQL Data Definition 66

3.3 Basic Structure of SQL Queries 71 3.4 Additional Basic Operations 79 3.5 Set Operations 85

3.6 Null Values 89



2.6 The Relational Algebra 48

2.7 Summary 58

Exercises 60

Further Reading 63

3.7 Aggregate Functions 91

3.8 Nested Subqueries 98

3.9 Modification of the Database 108 3.10 Summary 114

Exercises 115

Further Reading 124

vii

viii Contents

Chapter 4 Intermediate SQL

4.1 Join Expressions 125

4.2 Views 137

4.3 Transactions 143

4.4 Integrity Constraints 145

4.5 SQL Data Types and Schemas 153

Chapter 5 Advanced SQL

5.1 Accessing SQL from a Programming

Language 183

5.2 Functions and Procedures 198

5.3 Triggers 206

5.4 Recursive Queries 213

4.6 Index Definition in SQL 164 4.7 Authorization 165

4.8 Summary 173

Exercises 176

Further Reading 180

5.5 Advanced Aggregation Features 219 5.6 Summary 231

Exercises 232

Further Reading 238

PART TWO DATABASE DESIGN Chapter 6 Database Design Using the E-R Model

6.1 Overview of the Design Process 241 6.2 The Entity-Relationship Model 244 6.3 Complex Attributes 249

6.4 Mapping Cardinalities 252

6.5 Primary Key 256

6.6 Removing Redundant Attributes in Entity Sets 261

6.7 Reducing E-R Diagrams to Relational Schemas 264

Chapter 7 Relational Database Design

7.1 Features of Good Relational Designs 303 7.2 Decomposition Using Functional

Dependencies 308

7.3 Normal Forms 313

7.4 Functional-Dependency Theory 320 7.5 Algorithms for Decomposition Using Functional Dependencies 330

7.6 Decomposition Using Multivalued

Dependencies 336

6.8 Extended E-R Features 271

6.9 Entity-Relationship Design Issues 279 6.10 Alternative Notations for Modeling Data 285

6.11 Other Aspects of Database Design 291 6.12 Summary 292

Exercises 294

Further Reading 300

7.7 More Normal Forms 341

7.8 Atomic Domains and First Normal Form 342

7.9 Database-Design Process 343 7.10 Modeling Temporal Data 347 7.11 Summary 351

Exercises 353

Further Reading 360

Contents ix

PART THREE APPLICATION DESIGN AND DEVELOPMENT

Chapter 8 Complex Data Types

8.1 Semi-structured Data 365

8.2 Object Orientation 376

8.3 Textual Data 382

8.4 Spatial Data 387

Chapter 9 Application Development

9.1 Application Programs and User

Interfaces 403

9.2 Web Fundamentals 405

9.3 Servlets 411

9.4 Alternative Server-Side Frameworks 416 9.5 Client-Side Code and Web Services 421 9.6 Application Architectures 429

8.5 Summary 394

Exercises 397

Further Reading 401

9.7 Application Performance 434 9.8 Application Security 437

9.9 Encryption and Its Applications 447 9.10 Summary 453

Exercises 455

Further Reading 462

PART FOUR BIG DATA ANALYTICS Chapter 10 Big Data

10.1 Motivation 467

10.2 Big Data Storage Systems 472 10.3 The MapReduce Paradigm 483 10.4 Beyond MapReduce: Algebraic Operations 494

Chapter 11 Data Analytics

11.1 Overview of Analytics 519 11.2 Data Warehousing 521

11.3 Online Analytical Processing 527 11.4 Data Mining 540

10.5 Streaming Data 500 10.6 Graph Databases 508 10.7 Summary 511

Exercises 513

Further Reading 516

11.5 Summary 550

Exercises 552

Further Reading 555

x Contents

PART FIVE STORAGE MANAGEMENT AND INDEXING

Chapter 12 Physical Storage Systems

12.1 Overview of Physical Storage Media 559 12.2 Storage Interfaces 562

12.3 Magnetic Disks 563

12.4 Flash Memory 567

12.5 RAID 570

Chapter 13 Data Storage Structures

13.1 Database Storage Architecture 587 13.2 File Organization 588

13.3 Organization of Records in Files 595 13.4 Data-Dictionary Storage 602

13.5 Database Buffer 604

13.6 Column-Oriented Storage 611

Chapter 14 Indexing

14.1 Basic Concepts 623

14.2 Ordered Indices 625

14.3 B+-Tree Index Files 634

14.4 B+-Tree Extensions 650

14.5 Hash Indices 658

14.6 Multiple-Key Access 661

14.7 Creation of Indices 664

12.6 Disk-Block Access 577

12.7 Summary 580

Exercises 582

Further Reading 584

13.7 Storage Organization in Main-Memory Databases 615

13.8 Summary 617

Exercises 619

Further Reading 621

14.8 Write-Optimized Index Structures 665 14.9 Bitmap Indices 670

14.10 Indexing of Spatial and Temporal Data 672 14.11 Summary 677

Exercises 679

Further Reading 683

PART SIX QUERY PROCESSING AND OPTIMIZATION

Chapter 15 Query Processing

15.1 Overview 689

15.2 Measures of Query Cost 692 15.3 Selection Operation 695 15.4 Sorting 701

15.5 Join Operation 704

15.6 Other Operations 719

15.7 Evaluation of Expressions 724 15.8 Query Processing in Memory 731 15.9 Summary 734

Exercises 736

Further Reading 740

Contents xi

Chapter 16 Query Optimization

16.1 Overview 743

16.2 Transformation of Relational Expressions 747

16.3 Estimating Statistics of Expression Results 757

16.4 Choice of Evaluation Plans 766

16.5 Materialized Views 778 16.6 Advanced Topics in Query Optimization 783

16.7 Summary 787

Exercises 789

Further Reading 794

PART SEVEN TRANSACTION MANAGEMENT Chapter 17 Transactions

17.1 Transaction Concept 799

17.2 A Simple Transaction Model 801 17.3 Storage Structure 804

17.4 Transaction Atomicity and Durability 805 17.5 Transaction Isolation 807

17.6 Serializability 812

17.7 Transaction Isolation and Atomicity 819

Chapter 18 Concurrency Control

18.1 Lock-Based Protocols 835

18.2 Deadlock Handling 849

18.3 Multiple Granularity 853

18.4 Insert Operations, Delete Operations, and Predicate Reads 857

18.5 Timestamp-Based Protocols 861 18.6 Validation-Based Protocols 866 18.7 Multiversion Schemes 869

Chapter 19 Recovery System

19.1 Failure Classification 907

19.2 Storage 908

19.3 Recovery and Atomicity 912

19.4 Recovery Algorithm 922

19.5 Buffer Management 926

19.6 Failure with Loss of Non-Volatile Storage 930

19.7 High Availability Using Remote Backup Systems 931

17.8 Transaction Isolation Levels 821 17.9 Implementation of Isolation Levels 823 17.10 Transactions as SQL Statements 826 17.11 Summary 828

Exercises 831

Further Reading 834

18.8 Snapshot Isolation 872

18.9 Weak Levels of Consistency in

Practice 880

18.10 Advanced Topics in Concurrency Control 883

18.11 Summary 894

Exercises 899

Further Reading 904

19.8 Early Lock Release and Logical Undo Operations 935

19.9 ARIES 941

19.10 Recovery in Main-Memory Databases 947 19.11 Summary 948

Exercises 952

Further Reading 956

xii Contents

PART EIGHT PARALLEL AND DISTRIBUTED DATABASES

Chapter 20 Database-System Architectures

20.1 Overview 961

20.2 Centralized Database Systems 962 20.3 Server System Architectures 963 20.4 Parallel Systems 970

20.5 Distributed Systems 986

20.6 Transaction Processing in Parallel and Distributed Systems 989

20.7 Cloud-Based Services 990 20.8 Summary 995

Exercises 998

Further Reading 1001

Chapter 21 Parallel and Distributed Storage

21.1 Overview 1003

21.2 Data Partitioning 1004

21.3 Dealing with Skew in Partitioning 1007 21.4 Replication 1013

21.5 Parallel Indexing 1017

21.6 Distributed File Systems 1019 21.7 Parallel Key-Value Stores 1023 21.8 Summary 1032

Exercises 1033

Further Reading 1036

Chapter 22 Parallel and Distributed Query Processing

22.1 Overview 1039

22.2 Parallel Sort 1041

22.3 Parallel Join 1043

22.4 Other Operations 1048

22.5 Parallel Evaluation of Query Plans 1052 22.6 Query Processing on Shared-Memory Architectures 1061

22.7 Query Optimization for Parallel

Execution 1064

22.8 Parallel Processing of Streaming Data 1070 22.9 Distributed Query Processing 1076 22.10 Summary 1086

Exercises 1089

Further Reading 1093

Chapter 23 Parallel and Distributed Transaction Processing

23.1 Distributed Transactions 1098 23.2 Commit Protocols 1100

23.3 Concurrency Control in Distributed Databases 1111

23.4 Replication 1121

23.5 Extended Concurrency Control Protocols 1129

23.6 Replication with Weak Degrees of Consistency 1133

23.7 Coordinator Selection 1146

23.8 Consensus in Distributed Systems 1150 23.9 Summary 1162

Exercises 1165

Further Reading 1168

Contents xiii

PART NINE ADVANCED TOPICS

Chapter 24 Advanced Indexing Techniques

24.1 Bloom Filter 1175

24.2 Log-Structured Merge Tree and Variants 1176

24.3 Bitmap Indices 1182 24.4 Indexing of Spatial Data 1186

24.5 Hash Indices 1190 24.6 Summary 1203 Exercises 1205

Further Reading 1206

Chapter 25 Advanced Application Development

25.1 Performance Tuning 1210

25.2 Performance Benchmarks 1230

25.3 Other Issues in Application

Development 1234

25.4 Standardization 1237

Chapter 26 Blockchain Databases

26.1 Overview 1252

26.2 Blockchain Properties 1254

26.3 Achieving Blockchain Properties via Cryptographic Hash Functions 1259

26.4 Consensus 1263

26.5 Data Management in a Blockchain 1267 PART TEN APPENDIX A

25.5 Distributed Directory Systems 1240 25.6 Summary 1243

Exercises 1245

Further Reading 1248

26.6 Smart Contracts 1269

26.7 Performance Enhancement 1274 26.8 Emerging Applications 1276 26.9 Summary 1279

Exercises 1280

Further Reading 1282

Appendix A Detailed University Schema 1287 Index 1299

PART ELEVEN ONLINE CHAPTERS Chapter 27 Formal Relational Query Languages Chapter 28 Advanced Relational Database Design Chapter 29 Object-Based Databases

Chapter 30 XML

Chapter 31 Information Retrieval

Chapter 32 PostgreSQL

Preface

Database management has evolved from a specialized computer application to a cen tral component of virtually all enterprises, and, as a result, knowledge about database systems has become an essential part of an education in computer science. In this text, we present the fundamental concepts of database management. These concepts include aspects of database design, database languages, and database-system implementation.

This text is intended for a first course in databases at the junior or senior under graduate, or first-year graduate, level. In addition to basic material for a first course, the text contains advanced material that can be used for course supplements, or as introductory material for an advanced course.

We assume only a familiarity with basic data structures, computer organization, and a high-level programming language such as Java, C, C++, or Python. We present concepts as intuitive descriptions, many of which are based on our running example of a university. Important theoretical results are covered, but formal proofs are omitted. In place of proofs, figures and examples are used to suggest why a result is true. Formal descriptions and proofs of theoretical results may be found in research papers and advanced texts that are referenced in the bibliographical notes.

The fundamental concepts and algorithms covered in the book are often based on those used in existing commercial or experimental database systems. Our aim is to present these concepts and algorithms in a general setting that is not tied to one particular database system, though we do provide references to specific systems where appropriate.

In this, the seventh edition of *Database System Concepts*, we have retained the over all style of the prior editions while evolving the content and organization to reflect the changes that are occurring in the way databases are designed, managed, and used. One such major change is the extensive use of “Big Data” systems. We have also taken into account trends in the teaching of database concepts and made adaptations to facilitate these trends where appropriate.

xv

xvi Preface

Among the notable changes in this edition are:

• Extensive coverage of Big Data systems, from the user perspective (Chapter 10), as well as from an internal perspective (Chapter 20 through Chapter 23), with extensive additions and modifications compared to the sixth edition.

• A new chapter entitled “Blockchain Databases” (Chapter 26) that introduces blockchain technology and its growing role in enterprise applications. An im portant focus in this chapter is the interaction between blockchain systems and database systems.

• Updates to all chapters covering database internals (Chapter 12 through Chap ter 19) to reflect current-generation technology, such as solid-state disks, main memory databases, multi-core systems, and column-stores.

• Enhanced coverage of semi-structured data management using JSON, RDF, and SPARQL (Section 8.1).

• Updated coverage of temporal data (in Section 7.10), data analytics (Chapter 11), and advanced indexing techniques such as write-optimized indices (Section 14.8 and Section 24.2).

• Reorganization and update of chapters to better support courses with a significant hands-on component (which we strongly recommend for any database course), including use of current-generation application development tools and Big Data systems such as Apache Hadoop and Spark.

These and other updates have arisen from the many comments and suggestions we have received from readers of the sixth edition, our students at Yale University, Lehigh University, and IIT Bombay, and our own observations and analyses of developments in database technology.

**Content of This Book**

The text is organized in eleven major parts.

• Overview (Chapter 1). Chapter 1 provides a general overview of the nature and pur pose of database systems. We explain how the concept of a database system has developed, what the common features of database systems are, what a database system does for the user, and how a database system interfaces with operating systems. We also introduce an example database application: a university organi zation consisting of multiple departments, instructors, students, and courses. This application is used as a running example throughout the book. This chapter is motivational, historical, and explanatory in nature.

Preface xvii

• Part 1: Relational Model and SQL (Chapter 2 through Chapter 5). Chapter 2 in troduces the relational model of data, covering basic concepts such as the struc ture of relational databases, database schemas, keys, schema diagrams, relational query languages, relational operations, and the relational algebra. Chapter 3, Chap ter 4, and Chapter 5 focus on the most influential of the user-oriented relational languages: SQL. The chapters in this part describe data manipulation: queries, updates, insertions, and deletions, assuming a schema design has been provided. Although data-definition syntax is covered in detail here, schema design issues are deferred to Part 2.

• Part 2: Database Design (Chapter 6 and Chapter 7). Chapter 6 provides an overview of the database-design process and a detailed description of the entity relationship data model. The entity-relationship data model provides a high-level view of the issues in database design and of the problems encountered in capturing the semantics of realistic applications within the constraints of a data model. UML class-diagram notation is also covered in this chapter. Chapter 7 introduces rela tional database design. The theory of functional dependencies and normalization is covered, with emphasis on the motivation and intuitive understanding of each normal form. This chapter begins with an overview of relational design and relies on an intuitive understanding of logical implication of functional dependencies. This allows the concept of normalization to be introduced prior to full coverage of functional-dependency theory, which is presented later in the chapter. Instructors may choose to use only this initial coverage without loss of continuity. Instructors covering the entire chapter will benefit from students having a good understand ing of normalization concepts to motivate them to learn some of the challenging concepts of functional-dependency theory. The chapter ends with a section on modeling of temporal data.

• Part 3: Application Design and Development (Chapter 8 and Chapter 9). Chapter 8 discusses several complex data types that are particularly important for appli cation design and development, including semi-structured data, object-based data, textual data, and spatial data. Although the popularity of XML in a database con text has been diminishing, we retain an introduction to XML, while adding coverage of JSON, RDF, and SPARQL. Chapter 9 discusses tools and technologies that are used to build interactive web-based and mobile database applications. This chap ter includes detailed coverage on both the server side and the client side. Among the topics covered are servlets, JSP, Django, JavaScript, and web services. Also discussed are application architecture, object-relational mapping systems includ ing Hibernate and Django, performance (including caching using memcached and Redis), and the unique challenges in ensuring web-application security.

• Part 4: Big Data Analytics (Chapter 10 and Chapter 11). Chapter 10 provides an overview of large-scale data-analytic applications, with a focus on how those applications place distinct demands on data management compared with the de-

xviii Preface

mands of traditional database applications. The chapter then discusses how those demands are addressed. Among the topics covered are Big Data storage systems including distributed file systems, key-value stores and NoSQL systems, MapRe duce, Apache Spark, streaming data, and graph databases. The connection of these systems and concepts with database concepts introduced earlier is emphasized. Chapter 11 discusses the structure and use of systems designed for large-scale data analysis. After first explaining the concepts of data analytics, business intelligence, and decision support, the chapter discusses the structure of a data warehouse and the process of gathering data into a warehouse. The chapter next covers usage of warehouse data in OLAP applications followed by a survey of data-mining algo rithms and techniques.

• Part 5: Storage Management and Indexing (Chapter 12 through Chapter 14). Chap ter 12 deals with storage devices and how the properties of those devices influ ence database physical organization and performance. Chapter 13 deals with data storage structures, including file organization and buffer management. A variety of data-access techniques are presented in Chapter 14. Multilevel index-based access is described, culminating in detailed coverage of B+-trees. The chapter then covers index structures for applications where the B+-tree structure is less appropriate, in cluding write-optimized indices such as LSM trees and buffer trees, bitmap indices, and the indexing of spatial data using k-d trees, quadtrees and R-trees.

• Part 6: Query Processing and Optimization (Chapter 15 and Chapter 16). Chap ter 15 and Chapter 16 address query-evaluation algorithms and query optimiza tion. Chapter 15 focuses on algorithms for the implementation of database opera tions, particularly the wide range of join algorithms, which are designed to work on very large data that may not fit in main-memory. Query processing techniques for main-memory databases are also covered in this chapter. Chapter 16 covers query optimization, starting by showing how query plans can be transformed to other equivalent plans by using transformation rules. The chapter then describes how to generate estimates of query execution costs, and how to efficiently find query execution plans with the lowest cost.

• Part 7: Transaction Management (Chapter 17 through Chapter 19). Chapter 17 focuses on the fundamentals of a transaction-processing system: atomicity, con sistency, isolation, and durability. It provides an overview of the methods used to ensure these properties, including log-based recovery and concurrency control using locking, timestamp-based techniques, and snapshot isolation. Courses re quiring only a survey of the transaction concept can use Chapter 17 on its own without the other chapters in this part; those chapters provide significantly greater depth. Chapter 18 focuses on concurrency control and presents several techniques for ensuring serializability, including locking, timestamping, and optimistic (vali dation) techniques. Multiversion concurrency control techniques, including the widely used snapshot isolation technique, and an extension of the technique that

Preface xix

guarantees serializability, are also covered. This chapter also includes discussion of weak levels of consistency, concurrency on index structures, concurrency in main-memory database systems, long-duration transactions, operation-level con currency, and real-time transaction processing. Chapter 19 covers the primary techniques for ensuring correct transaction execution despite system crashes and storage failures. These techniques include logs, checkpoints, and database dumps, as well as high availability using remote backup systems. Recovery with early lock release, and the widely used ARIES algorithm are also presented. This chapter in cludes discussion of recovery in main-memory database systems and the use of NVRAM.

• Part 8: Parallel and Distributed Databases (Chapter 20 through Chapter 23). Chapter 20 covers computer-system architecture, and describes the influence of the underlying computer system on the database system. We discuss centralized systems, client–server systems, parallel and distributed architectures, and cloud based systems in this chapter. The remaining three chapters in this part address distinct aspects of parallel and distributed databases, with Chapter 21 covering storage and indexing, Chapter 22 covering query processing, and Chapter 23 cov ering transaction management. Chapter 21 includes discussion of partitioning and data skew, replication, parallel indexing, distributed file systems (including the Hadoop file system), and parallel key-value stores. Chapter 22 includes discussion of parallelism both among multiple queries and within a single query. It covers par allel and distributed sort and join, MapReduce, pipelining, the Volcano exchange operator model, thread-level parallelism, streaming data, and the optimization of geographically distributed queries. Chapter 23 includes discussion of traditional distributed consensus such as two-phase commit and more sophisticated solutions including Paxos and Raft. It covers a variety of algorithms for distributed concur rency control, including replica management and weaker degrees of consistency. The trade-offs implied by the CAP theorem are discussed along with the means of detecting inconsistency using version vectors and Merkle trees.

• Part 9: Advanced Topics (Chapter 24 through Chapter 26). Chapter 24 expands upon the coverage of indexing in Chapter 14 with detailed coverage of the LSM tree and its variants, bitmap indices, spatial indexing, and dynamic hashing tech niques. Chapter 25 expands upon the coverage of Chapter 9 with a discussion of performance tuning, benchmarking, testing, and migration from legacy systems, standardization, and distributed directory systems. Chapter 26 looks at blockchain technology from a database perspective. It describes blockchain data structures and the use of cryptographic hash functions and public-key encryption to ensure the blockchain properties of anonymity, irrefutability, and tamper resistance. It describes and compares the distributed consensus algorithms used to ensure de centralization, including proof-of-work, proof-of-stake, and Byzantine consensus. Much of the chapter focuses on the features that make blockchain an important database concept, including the role of permisssioned blockchains, the encoding

xx Preface

of business logic and agreements in smart contracts, and interoperability across blockchains. Techniques for achieving database-scale transaction-processing per formance are discussed. A final section surveys current and contemplated enter prise blockchain applications.

• Part 10: Appendix. Appendix A presents details of our university schema, including the full schema, DDL, and all the tables.

• Part 11: Online Chapters (Chapter 27 through Chapter 32) available online at db-book.com. We provide six chapters that cover material that is of historical nature or is advanced; these chapters are available only online. Chapter 27 cov ers “pure” query languages: the tuple and domain relational calculus and Data log, which has a syntax modeled after the Prolog language. Chapter 28 covers advanced topics in relational database design, including the theory of multivalued dependencies and fourth normal form, as well as higher normal forms. Chapter 29 covers object-based databases and more complex data types such as array, and multiset types, as well as tables that are not in 1NF. Chapter 30 expands on the cov erage in Chapter 8 of XML. Chapter 31 covers information retrieval, which deals with querying of unstructured textual data. Chapter 32 provides an overview of the PostgreSQL database system, and is targeted at courses focusing on database inter nals. The chapter is likely to be particularly useful for supporting student projects that work with the open-source code base of the PostgreSQL database.

At the end of each chapter we provide references in a section titled *Further Reading*. This section is intentionally abbreviated and provides references that allow students to continue their study of the material covered in the chapter or to learn about new developments in the area covered by the chapter. On occasion, the further reading section includes original source papers that have become classics of which everyone should be aware. Detailed bibliographical notes for each chapter are available online, and provide references for readers who wish to go into further depth on any of the topics covered in the chapter.

**The Seventh Edition**

The production of this seventh edition has been guided by the many comments and suggestions we received concerning the earlier editions, by our own observations while teaching at Yale University, Lehigh University, and IIT Bombay, and by our analysis of the directions in which database technology is evolving.

We provided a list of the major new features of this edition earlier in this preface; these include coverage of extensive coverage of Big Data, updates to all chapters to reflect current generation hardware technology, semi-structured data management, ad vanced indexing techniques, and a new chapter on blockchain databases. Beyond these major changes, we revised the material in each chapter, bringing the older material

Preface xxi

up-to-date, adding discussions on recent developments in database technology, and im proving descriptions of topics that students found difficult to understand. We have also added new exercises and updated references.

For instructors who previously used the sixth edition, we list the more significant changes below:

• Relational algebra has been moved into Chapter 2, to help students better under stand relational operations that form the basis of query languages such as SQL. Deeper coverage of relational algebra also aids in understanding the algebraic op erators needed for discussion later of query processing and optimization. The two variants of the relational calculus are now in an online chapter, since we believe they are now of value only to more theoretically oriented courses, and can be omit ted by most database courses.

• The SQL chapters now include more details of database-system specific SQL vari ations, to aid students carrying out practical assignments. Connections between SQL and the multiset relational algebra are also covered in more detail. Chapter 4 now covers all the material concerning joins, whereas previously natural join was in the preceding chapter. Coverage of sequences used to generate unique key values, and coverage of row-level security have also been added to this chapter. Recent extensions to the JDBC API that are particularly useful are now covered in Chapter 5; coverage of OLAP has been moved from this chapter to Chapter 11.

• Chapter 6 has been modified to cover E-R diagrams along with E-R concepts, in stead of first covering the concepts and then introducing E-R diagrams as was done in earlier editions. We believe this will help students better comprehend the E-R model.

• Chapter 7 now has improved coverage of temporal data modeling, including SQL:2011 temporal database features.

• Chapter 8 is a new chapter that covers complex data types, including semi structured data, such as XML, JSON, RDF, and SPARQL, object-based data, textual data, and spatial data. Object-based databases, XML, and information retrieval on textual data were covered in detail in the sixth edition; these topics have been ab breviated and covered in Chapter 8, while the original chapters from the sixth edition have now been made available online.

• Chapter 9 has been significantly updated to reflect modern application devel opment tools and techniques, including extended coverage of JavaScript and JavaScript libraries for building dynamic web interfaces, application development in Python using the Django framework, coverage of web services, and disconnec tion operations using HTML5. Object-relation mapping using Django has been added, as also discussion of techniques for developing high-performance applica tions that can handle large transaction loads.

xxii Preface

• Chapter 10 is a new chapter on Big Data, covering Big Data concepts and tools from a user perspective. Big Data storage systems, the MapReduce paradigm, Apache Hadoop and Apache Spark, and streaming and graph databases are cov ered in this chapter. The goal is to enable readers to use Big Data systems, with only a summary coverage of what happens behind the scenes. Big Data internals are covered in detail in later chapters.

• The chapter on storage and file structure has been split into two chapters. Chap ter 12 which covers storage has been updated with new technology, including ex panded coverage of flash memory, column-oriented storage, and storage organiza tion in main-memory databases. Chapter 13, which covers data storage structures has been expanded, and now covers details such as free-space maps, partitioning, and most importantly column-oriented storage.

• Chapter 14 on indexing now covers write-optimized index structures including the LSM tree and its variants, and the buffer tree, which are seeing increasing usage. Spatial indices are now covered briefly in this chapter. More detailed coverage of LSM trees and spatial indices is provided in Chapter 24, which covers advanced indexing techniques. Bitmap indices are now covered in brief in Chapter 14, while more detailed coverage has been moved to Chapter 24. Dynamic hashing tech niques have been moved into Chapter 24, since they are of limited practical im portance today.

• Chapter 15 on query processing has significantly expanded coverage of pipelining in query processing, new material on query processing in main-memory, including query compilation, as well as brief coverage of spatial joins. Chapter 16 on query optimization has more examples of equivalence rules for operators such as outer joins and aggregates, has updated material on statistics for cost estimation, an improved presentation of the join-order optimization algorithm. Techniques for decorrelating nested subqueries using semijoin and antijoin operations have also been added.

• Chapter 18 on concurrency control has new material on concurrency control in main-memory. Chapter 19 on recovery now gives more importance to high avail ability using remote backup systems.

• Our coverage of parallel and distributed databases has been completely revamped. Because of the evolution of these two areas into a continuum from low-level paral lelism to geographically distributed systems, we now present these topics together.

° Chapter 20 on database architectures has been significantly updated from the earlier edition, including new material on practical interconnection networks like the tree-like (or fat-tree) architecture, and significantly expanded and up dated material on shared-memory architectures and cache coherency. There is an entirely new section on cloud-based services, covering virtual machines and containers, platform-as-a-service, software-as-a-service, and elasticity.

Preface xxiii

° Chapter 21 covers parallel and distributed storage; while a few parts of this chapter were present in the sixth edition, such as partitioning techniques, ev erything else in this chapter is new.

° Chapter 22 covers parallel and distributed query processing. Again only a few sections of this chapter, such as parallel algorithms for sorting, join, and a few other relational operations, were present in the sixth edition, almost everything else in this chapter is new.

° Chapter 23 covers parallel and distributed transaction processing. A few parts of this chapter, such as the sections on 2PC, persistent messaging, and concur rency control in distributed databases, are new but almost everything else in this chapter is new.

As in the sixth edition, we facilitate the following of our running example by listing the database schema and the sample relation instances for our university database to gether in Appendix A as well as where they are used in the various regular chapters. In addition, we provide, on our web site db-book.com, SQL data-definition statements for the entire example, along with SQL statements to create our example relation instances. This encourages students to run example queries directly on a database system and to experiment with modifying those queries. All topics not listed above are updated from the sixth edition, though their overall organization is relatively unchanged.

**End of Chapter Material**

Each chapter has a list of review terms, in addition to a summary, which can help readers review key topics covered in the chapter.

As in the sixth edition, the exercises are divided into two sets: practice exercises and exercises. The solutions for the practice exercises are publicly available on the web site of the book. Students are encouraged to solve the practice exercises on their own and later use the solutions on the web site to check their own solutions. Solutions to the other exercises are available only to instructors (see “Instructor’s Note,” below, for information on how to get the solutions).

Many chapters have a tools section at the end of the chapter that provides infor mation on software tools related to the topic of the chapter; some of these tools can be used for laboratory exercises. SQL DDL and sample data for the university database and other relations used in the exercises are available on the web site of the book and can be used for laboratory exercises.

**Instructor’s Note**

It is possible to design courses by using various subsets of the chapters. Some of the chapters can also be covered in an order different from their order in the book. We outline some of the possibilities here:

xxiv Preface

• Chapter 5 (Advanced SQL). This chapter can be skipped or deferred to later with out loss of continuity. We expect most courses will cover at least Section 5.1.1 early, as JDBC is likely to be a useful tool in student projects.

• Chapter 6 (E-R Model). This chapter can be covered ahead of Chapter 3, Chapter 4, and Chapter 5 if you so desire, since Chapter 6 does not have any dependency on SQL. However, for courses with a programming emphasis, a richer variety of laboratory exercises is possible after studying SQL, and we recommend that SQL be covered before database design for such courses.

• Chapter 15 (Query Processing) and Chapter 16 (Query Optimization). These chapters can be omitted from an introductory course without affecting coverage of any other chapter.

• Part 7 (Transaction Management). Our coverage consists of an overview (Chapter 17) followed by chapters with details. You might choose to use Chapter 17 while omitting Chapter 18 and Chapter 19, if you defer these latter chapters to an ad vanced course.

• Part 8 (Parallel and Distributed Databases). Our coverage consists of an overview (Chapter 20), followed by chapters on the topics of storage, query processing, and transactions. You might choose to use Chapter 20 while omitting Chapter 21 through Chapter 23 if you defer these latter chapters to an advanced course.

• Part 11 (Online chapters). Chapter 27 (Formal-Relational Query Languages). This chapter can be covered immediately after Chapter 2, ahead of SQL. Alternatively, this chapter may be omitted from an introductory course. The five other online chapters (Advanced Relational Database Design, Object-Based Databases, XML, Information Retrieval, and PostgreSQL) can be used as self-study material or omit ted from an introductory course.

Model course syllabi, based on the text, can be found on the web site of the book.

**Web Site and Teaching Supplements**

A web site for the book is available at the URL: db-book.com. The web site contains: • Slides covering all the chapters of the book.

• Answers to the practice exercises.

• The six online chapters.

• Laboratory material, including SQL DDL and sample data for the university schema and other relations used in exercises, and instructions for setting up and using various database systems and tools.

• An up-to-date errata list.

Preface xxv

The following additional material is available only to faculty:

• An instructor’s manual containing solutions to all exercises in the book. • A question bank containing extra exercises.

For more information about how to get a copy of the instructor’s manual and the question bank, please send an email message to sem@mheducation.com. In the United States, you may call 800-338-3987. The McGraw-Hill web site for this book is www.mhhe.com/silberschatz.

**Contacting Us**

We have endeavored to eliminate typos, bugs, and the like from the text. But, as in new releases of software, bugs almost surely remain; an up-to-date errata list is accessible from the book’s web site. We would appreciate it if you would notify us of any errors or omissions in the book that are not on the current list of errata.

We would be glad to receive suggestions on improvements to the book. We also welcome any contributions to the book web site that could be of use to other read ers, such as programming exercises, project suggestions, online labs and tutorials, and teaching tips.

Email should be addressed to db-book-authors@cs.yale.edu. Any other corre spondence should be sent to Avi Silberschatz, Department of Computer Science, Yale University, 51 Prospect Street, P.O. Box 208285, New Haven, CT 06520-8285 USA.

**Acknowledgments**

Many people have helped us with this seventh edition, as well as with the previous six editions from which it is derived, and we are indebted to all of them.

Seventh Edition

• Ioannis Alagiannis and Renata Borovica-Gajic for writing Chapter 32 on the PostgreSQL database, which is available online. The chapter is a complete rewrite of the PostgreSQL chapter in the 6th edition, which was authored by Anastasia Ailamaki, Sailesh Krishnamurthy, Spiros Papadimitriou, Bianca Schroeder, Karl Schnaitter, and Gavin Sherry.

• Judi Paige for her help in generating figures, presentation slides, and with handling the copy-editing material.

• Mark Wogahn for making sure that the software to produce the book, including LaTeX macros and fonts, worked properly.

xxvi Preface

• Sriram Srinivasan for discussions and feedback that have immensely benefited the chapters on parallel and distributed databases.

• N. L. Sarda for his insightful feedback on the sixth edition, and on some sections of the seventh edition.

• Bikash Chandra and Venkatesh Emani for their help with updates to the applica tion development chapter, including creation of sample code.

• Students at IIT Bombay, particularly Ashish Mithole, for their feedback on draft versions of the chapters on parallel and distributed databases.

• Students at Yale, Lehigh, and IIT Bombay, for their comments on the sixth edition. • Jeffrey Anthony, partner and CTO, Synaptic; and Lehigh students Corey Ca plan (now co-founder, Leavitt Innovations); Gregory Cheng; Timothy LaRowe; and Aaron Rotem for comments and suggestions that have benefited the new blockchain chapter.

Previous Editions

• Hakan Jakobsson (Oracle), for writing the chapter on the Oracle database sys tem in the sixth edition; Sriram Padmanabhan (IBM), for writing the chapter de scribing the IBM DB2 database system in the sixth edition; and Sameet Agarwal, Jos´e A. Blakeley, Thierry D’Hers, Gerald Hinson, Dirk Myers, Vaqar Pirzada, Bill Ramos, Balaji Rathakrishnan, Michael Rys, Florian Waas, and Michael Zwilling for writing the chapter describing the Microsoft SQL Server database system in the sixth edition; and in particular Jos´e Blakeley, who sadly is no longer amongst us, for coordinating and editing the chapter; and C´esar Galindo-Legaria, Goetz Graefe, Kalen Delaney, and Thomas Casey for their contributions to the previous edition of the Microsoft SQL Server chapter. These chapters, however, are not part of the seventh edition.

• Anastasia Ailamaki, Sailesh Krishnamurthy, Spiros Papadimitriou, Bianca Schroeder, Karl Schnaitter, and Gavin Sherry for writing the chapter on PostgreSQL in the sixth edition.

• Daniel Abadi for reviewing the table of contents of the fifth edition and helping with the new organization.

• Steve Dolins, University of Florida; Rolando Fernanez, George Washington Uni versity; Frantisek Franek, McMaster University; Latifur Khan, University of Texas at Dallas; Sanjay Madria, Missouri University of Science and Technology; Aris Ouksel, University of Illinois; and Richard Snodgrass, University of Waterloo; who served as reviewers of the book and whose comments helped us greatly in formu lating the sixth edition.

Preface xxvii

• Judi Paige for her help in generating figures and presentation slides. • Mark Wogahn for making sure that the software to produce the book, including LaTeX macros and fonts, worked properly.

• N. L. Sarda for feedback that helped us improve several chapters. Vikram Pudi for motivating us to replace the earlier bank schema; and Shetal Shah for feedback on several chapters.

• Students at Yale, Lehigh, and IIT Bombay, for their comments on the fifth edition, as well as on preprints of the sixth edition.

• Chen Li and Sharad Mehrotra for providing material on JDBC and security for the fifth edition.

• Marilyn Turnamian and Nandprasad Joshi provided secretarial assistance for the fifth edition, and Marilyn also prepared an early draft of the cover design for the fifth edition.

• Lyn Dupr´e copyedited the third edition and Sara Strandtman edited the text of the third edition.

• Nilesh Dalvi, Sumit Sanghai, Gaurav Bhalotia, Arvind Hulgeri K. V. Raghavan, Prateek Kapadia, Sara Strandtman, Greg Speegle, and Dawn Bezviner helped to prepare the instructor’s manual for earlier editions.

• The idea of using ships as part of the cover concept was originally suggested to us by Bruce Stephan.

• The following people offered suggestions and comments for the fifth and earlier editions of the book. R. B. Abhyankar, Hani Abu-Salem, Jamel R. Alsabbagh, Raj Ashar, Don Batory, Phil Bernhard, Christian Breimann, Gavin M. Bierman, Janek Bogucki, Haran Boral, Paul Bourgeois, Phil Bohannon, Robert Brazile, Yuri Breitbart, Ramzi Bualuan, Michael Carey, Soumen Chakrabarti, Tom Chappell, Zhengxin Chen, Y. C. Chin, Jan Chomicki, Laurens Damen, Prasanna Dhan dapani, Qin Ding, Valentin Dinu, J. Edwards, Christos Faloutsos, Homma Far ian, Alan Fekete, Frantisek Franek, Shashi Gadia, Hector Garcia-Molina, Goetz Graefe, Jim Gray, Le Gruenwald, Eitan M. Gurari, William Hankley, Bruce Hillyer, Ron Hitchens, Chad Hogg, Arvind Hulgeri, Yannis Ioannidis, Zheng Ji aping, Randy M. Kaplan, Graham J. L. Kemp, Rami Khouri, Hyoung-Joo Kim, Won Kim, Henry Korth (father of Henry F.), Carol Kroll, Hae Choon Lee, Sang Won Lee, Irwin Levinstein, Mark Llewellyn, Gary Lindstrom, Ling Liu, Dave Maier, Keith Marzullo, Marty Maskarinec, Fletcher Mattox, Sharad Mehrotra, Jim Melton, Alberto Mendelzon, Ami Motro, Bhagirath Narahari, Yiu-Kai Dennis Ng, Thanh-Duy Nguyen, Anil Nigam, Cyril Orji, Meral Ozsoyoglu, D. B. Phatak, Juan Altmayer Pizzorno, Bruce Porter, Sunil Prabhakar, Jim Peterson, K. V. Raghavan, Nahid Rahman, Rajarshi Rakshit, Krithi Ramamritham, Mike Reiter, Greg Ric-

xxviii Preface

cardi, Odinaldo Rodriguez, Mark Roth, Marek Rusinkiewicz, Michael Rys, Sunita Sarawagi, N. L. Sarda, Patrick Schmid, Nikhil Sethi, S. Seshadri, Stewart Shen, Shashi Shekhar, Amit Sheth, Max Smolens, Nandit Soparkar, Greg Speegle, Jeff Storey, Dilys Thomas, Prem Thomas, Tim Wahls, Anita Whitehall, Christopher Wilson, Marianne Winslett, Weining Zhang, and Liu Zhenming.

Personal Notes

Sudarshan would like to acknowledge his wife, Sita, for her love, patience, and support, and children Madhur and Advaith for their love and joie de vivre. Hank would like to acknowledge his wife, Joan, and his children, Abby and Joe, for their love and under standing. Avi would like to acknowledge Valerie for her love, patience, and support during the revision of this book.

A. S.

H. F. K.

S. S.

***CHAPTER*** 1

Introduction

A database-management system (DBMS) is a collection of interrelated data and a set of programs to access those data. The collection of data, usually referred to as the database, contains information relevant to an enterprise. The primary goal of a DBMS is to provide a way to store and retrieve database information that is both *convenient* and *efficient*.

Database systems are designed to manage large bodies of information. Manage ment of data involves both defining structures for storage of information and provid ing mechanisms for the manipulation of information. In addition, the database system must ensure the safety of the information stored, despite system crashes or attempts at unauthorized access. If data are to be shared among several users, the system must avoid possible anomalous results.

Because information is so important in most organizations, computer scientists have developed a large body of concepts and techniques for managing data. These concepts and techniques form the focus of this book. This chapter briefly introduces the principles of database systems.

**1.1 Database-System Applications**

The earliest database systems arose in the 1960s in response to the computerized man agement of commercial data. Those earlier applications were relatively simple com pared to modern database applications. Modern applications include highly sophisti cated, worldwide enterprises.

All database applications, old and new, share important common elements. The central aspect of the application is not a program performing some calculation, but rather the data themselves. Today, some of the most valuable corporations are valuable not because of their physical assets, but rather because of the information they own. Imagine a bank without its data on accounts and customers or a social-network site that loses the connections among its users. Such companies’ value would be almost totally lost under such circumstances.

1

2 Chapter 1 Introduction

Database systems are used to manage collections of data that:

• are highly valuable,

• are relatively large, and

• are accessed by multiple users and applications, often at the same time.

The first database applications had only simple, precisely formatted, structured data. Today, database applications may include data with complex relationships and a more variable structure. As an example of an application with structured data, consider a university’s records regarding courses, students, and course registration. The univer sity keeps the same type of information about each course: course-identifier, title, de partment, course number, etc., and similarly for students: student-identifier, name, ad dress, phone, etc. Course registration is a collection of pairs: one course identifier and one student identifier. Information of this sort has a standard, repeating structure and is representative of the type of database applications that go back to the 1960s. Con trast this simple university database application with a social-networking site. Users of the site post varying types of information about themselves ranging from simple items such as name or date of birth, to complex posts consisting of text, images, videos, and links to other users. There is only a limited amount of common structure among these data. Both of these applications, however, share the basic features of a database.

Modern database systems exploit commonalities in the structure of data to gain efficiency but also allow for weakly structured data and for data whose formats are highly variable. As a result, a database system is a large, complex software system whose task is to manage a large, complex collection of data.

Managing complexity is challenging, not only in the management of data but in any domain. Key to the management of complexity is the concept of *abstraction*. Ab straction allows a person to use a complex device or system without having to know the details of how that device or system is constructed. A person is able, for example, to drive a car by knowing how to operate its controls. However, the driver does not need to know how the motor was built nor how it operates. All the driver needs to know is an abstraction of what the motor does. Similarly, for a large, complex collection of data, a database system provides a simpler, abstract view of the information so that users and application programmers do not need to be aware of the underlying details of how data are stored and organized. By providing a high level of abstraction, a database sys tem makes it possible for an enterprise to combine data of various types into a unified repository of the information needed to run the enterprise.

Here are some representative applications:

• Enterprise Information

° Sales: For customer, product, and purchase information.

1.1 Database-System Applications 3

° Accounting: For payments, receipts, account balances, assets, and other ac counting information.

° Human resources: For information about employees, salaries, payroll taxes, and benefits, and for generation of paychecks.

• Manufacturing: For management of the supply chain and for tracking production of items in factories, inventories of items in warehouses and stores, and orders for items.

• Banking and Finance

° Banking: For customer information, accounts, loans, and banking transactions. ° Credit card transactions: For purchases on credit cards and generation of monthly statements.

° Finance: For storing information about holdings, sales, and purchases of finan cial instruments such as stocks and bonds; also for storing real-time market data to enable online trading by customers and automated trading by the firm. • Universities: For student information, course registrations, and grades (in addition

to standard enterprise information such as human resources and accounting). • Airlines: For reservations and schedule information. Airlines were among the first to use databases in a geographically distributed manner.

• Telecommunication: For keeping records of calls, texts, and data usage, generating monthly bills, maintaining balances on prepaid calling cards, and storing informa tion about the communication networks.

• Web-based services

° Social-media: For keeping records of users, connections between users (such as friend/follows information), posts made by users, rating/like information about posts, etc.

° Online retailers: For keeping records of sales data and orders as for any retailer, but also for tracking a user’s product views, search terms, etc., for the purpose of identifying the best items to recommend to that user.

° Online advertisements: For keeping records of click history to enable targeted advertisements, product suggestions, news articles, etc. People access such databases every time they do a web search, make an online purchase, or ac cess a social-networking site.

• Document databases: For maintaining collections of new articles, patents, pub lished research papers, etc.

• Navigation systems: For maintaining the locations of varies places of interest along with the exact routes of roads, train systems, buses, etc.

4 Chapter 1 Introduction

As this list illustrates, databases form an essential part not only of every enterprise but also of a large part of a person’s daily activities.

The ways in which people interact with databases has changed over time. Early databases were maintained as back-office systems with which users interacted via printed reports and paper forms for input. As database systems became more sophisti cated, better languages were developed for programmers to use in interacting with the data, along with user interfaces that allowed end users within the enterprise to query and update data.

As the support for programmer interaction with databases improved, and computer hardware performance increased even as hardware costs decreased, more sophisticated applications emerged that brought database data into more direct contact not only with end users within an enterprise but also with the general public. Whereas once bank customers had to interact with a teller for every transaction, automated-teller machines (ATMs) allowed direct customer interaction. Today, virtually every enterprise employs web applications or mobile applications to allow its customers to interact directly with the enterprise’s database, and, thus, with the enterprise itself.

The user, or customer, can focus on the product or service without being aware of the details of the large database that makes the interaction possible. For instance, when you read a social-media post, or access an online bookstore and browse a book or music collection, you are accessing data stored in a database. When you enter an order online, your order is stored in a database. When you access a bank web site and retrieve your bank balance and transaction information, the information is retrieved from the bank’s database system. When you access a web site, information about you may be retrieved from a database to select which advertisements you should see. Almost every interaction with a smartphone results in some sort of database access. Furthermore, data about your web accesses may be stored in a database.

Thus, although user interfaces hide details of access to a database, and most people are not even aware they are dealing with a database, accessing databases forms an essential part of almost everyone’s life today.

Broadly speaking, there are two modes in which databases are used.

• The first mode is to support online transaction processing, where a large number of users use the database, with each user retrieving relatively small amounts of data, and performing small updates. This is the primary mode of use for the vast majority of users of database applications such as those that we outlined earlier.

• The second mode is to support data analytics, that is, the processing of data to draw conclusions, and infer rules or decision procedures, which are then used to drive business decisions.

For example, banks need to decide whether to give a loan to a loan applicant, online advertisers need to decide which advertisement to show to a particular user. These tasks are addressed in two steps. First, data-analysis techniques attempt to automatically discover rules and patterns from data and create *predictive models*. These models take as input attributes (“features”) of individuals, and output pre-

1.2 Purpose of Database Systems 5

dictions such as likelihood of paying back a loan, or clicking on an advertisement, which are then used to make the business decision.

As another example, manufacturers and retailers need to make decisions on what items to manufacture or order in what quantities; these decisions are driven significantly by techniques for analyzing past data, and predicting trends. The cost of making wrong decisions can be very high, and organizations are therefore willing to invest a lot of money to gather or purchase required data, and build systems that can use the data to make accurate predictions.

The field of *data mining* combines knowledge-discovery techniques invented by artificial intelligence researchers and statistical analysts with efficient implemen tation techniques that enable them to be used on extremely large databases.

**1.2 Purpose of Database Systems**

To understand the purpose of database systems, consider part of a university organiza tion that, among other data, keeps information about all instructors, students, depart ments, and course offerings. One way to keep the information on a computer is to store it in operating-system files. To allow users to manipulate the information, the system has a number of application programs that manipulate the files, including programs to:

• Add new students, instructors, and courses.

• Register students for courses and generate class rosters.

• Assign grades to students, compute grade point averages (GPA), and generate tran scripts.

Programmers develop these application programs to meet the needs of the university. New application programs are added to the system as the need arises. For exam ple, suppose that a university decides to create a new major. As a result, the university creates a new department and creates new permanent files (or adds information to existing files) to record information about all the instructors in the department, stu dents in that major, course offerings, degree requirements, and so on. The university may have to write new application programs to deal with rules specific to the new ma jor. New application programs may also have to be written to handle new rules in the university. Thus, as time goes by, the system acquires more files and more application programs.

This typical file-processing system is supported by a conventional operating system. The system stores permanent records in various files, and it needs different application programs to extract records from, and add records to, the appropriate files.

Keeping organizational information in a file-processing system has a number of major disadvantages:

6 Chapter 1 Introduction

• Data redundancy and inconsistency. Since different programmers create the files and application programs over a long period, the various files are likely to have different structures, and the programs may be written in several programming lan guages. Moreover, the same information may be duplicated in several places (files). For example, if a student has a double major (say, music and mathematics), the address and telephone number of that student may appear in a file that consists of student records of students in the Music department and in a file that consists of student records of students in the Mathematics department. This redundancy leads to higher storage and access cost. In addition, it may lead to data inconsistency; that is, the various copies of the same data may no longer agree. For example, a changed student address may be reflected in the Music department records but not elsewhere in the system.

• Difficulty in accessing data. Suppose that one of the university clerks needs to find out the names of all students who live within a particular postal-code area. The clerk asks the data-processing department to generate such a list. Because the designers of the original system did not anticipate this request, there is no application program on hand to meet it. There is, however, an application program to generate the list of *all* students. The university clerk now has two choices: either obtain the list of all students and extract the needed information manually or ask a programmer to write the necessary application program. Both alternatives are obviously unsatisfactory. Suppose that such a program is written and that, several days later, the same clerk needs to trim that list to include only those students who have taken at least 60 credit hours. As expected, a program to generate such a list does not exist. Again, the clerk has the preceding two options, neither of which is satisfactory.

The point here is that conventional file-processing environments do not allow needed data to be retrieved in a convenient and efficient manner. More responsive data-retrieval systems are required for general use.

• Data isolation. Because data are scattered in various files, and files may be in dif ferent formats, writing new application programs to retrieve the appropriate data is difficult.

• Integrity problems. The data values stored in the database must satisfy certain types of consistency constraints. Suppose the university maintains an account for each department, and records the balance amount in each account. Suppose also that the university requires that the account balance of a department may never fall below zero. Developers enforce these constraints in the system by adding appro priate code in the various application programs. However, when new constraints are added, it is difficult to change the programs to enforce them. The problem is compounded when constraints involve several data items from different files.

• Atomicity problems. A computer system, like any other device, is subject to failure. In many applications, it is crucial that, if a failure occurs, the data be restored to the

1.2 Purpose of Database Systems 7

consistent state that existed prior to the failure. Consider a banking system with a program to transfer $500 from account *A* to account *B*. If a system failure occurs during the execution of the program, it is possible that the $500 was removed from the balance of account *A* but was not credited to the balance of account *B*, resulting in an inconsistent database state. Clearly, it is essential to database consistency that either both the credit and debit occur, or that neither occur. That is, the funds transfer must be *atomic*—it must happen in its entirety or not at all. It is difficult to ensure atomicity in a conventional file-processing system.

• Concurrent-access anomalies. For the sake of overall performance of the system and faster response, many systems allow multiple users to update the data simulta neously. Indeed, today, the largest internet retailers may have millions of accesses per day to their data by shoppers. In such an environment, interaction of concur rent updates is possible and may result in inconsistent data. Consider account *A*, with a balance of $10,000. If two bank clerks debit the account balance (by say $500 and $100, respectively) of account *A* at almost exactly the same time, the re sult of the concurrent executions may leave the account balance in an incorrect (or inconsistent) state. Suppose that the programs executing on behalf of each with drawal read the old balance, reduce that value by the amount being withdrawn, and write the result back. If the two programs run concurrently, they may both read the value $10,000, and write back $9500 and $9900, respectively. Depending on which one writes the value last, the balance of account *A* may contain either $9500 or $9900, rather than the correct value of $9400. To guard against this possibility, the system must maintain some form of supervision. But supervision is difficult to provide because data may be accessed by many different application programs that have not been coordinated previously.

As another example, suppose a registration program maintains a count of students registered for a course in order to enforce limits on the number of students registered. When a student registers, the program reads the current count for the courses, verifies that the count is not already at the limit, adds one to the count, and stores the count back in the database. Suppose two students register concurrently, with the count at 39. The two program executions may both read the value 39, and both would then write back 40, leading to an incorrect increase of only 1, even though two students successfully registered for the course and the count should be 41. Furthermore, suppose the course registration limit was 40; in the above case both students would be able to register, leading to a violation of the limit of 40 students.

• Security problems. Not every user of the database system should be able to access all the data. For example, in a university, payroll personnel need to see only that part of the database that has financial information. They do not need access to information about academic records. But since application programs are added to the file-processing system in an ad hoc manner, enforcing such security constraints is difficult.

8 Chapter 1 Introduction

These difficulties, among others, prompted both the initial development of database systems and the transition of file-based applications to database systems, back in the 1960s and 1970s.

In what follows, we shall see the concepts and algorithms that enable database systems to solve the problems with file-processing systems. In most of this book, we use a university organization as a running example of a typical data-processing application.

**1.3 View of Data**

A database system is a collection of interrelated data and a set of programs that allow users to access and modify these data. A major purpose of a database system is to provide users with an *abstract* view of the data. That is, the system hides certain details of how the data are stored and maintained.

**1.3.1 Data Models**

Underlying the structure of a database is the data model: a collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints. There are a number of different data models that we shall cover in the text. The data models can be classified into four different categories:

• Relational Model. The relational model uses a collection of tables to represent both data and the relationships among those data. Each table has multiple columns, and each column has a unique name. Tables are also known as relations. The relational model is an example of a record-based model. Record-based models are so named because the database is structured in fixed-format records of several types. Each table contains records of a particular type. Each record type defines a fixed number of fields, or attributes. The columns of the table correspond to the attributes of the record type. The relational data model is the most widely used data model, and a vast majority of current database systems are based on the relational model. Chapter 2 and Chapter 7 cover the relational model in detail.

• Entity-Relationship Model. The entity-relationship (E-R) data model uses a collec tion of basic objects, called *entities*, and *relationships* among these objects. An en tity is a “thing” or “object” in the real world that is distinguishable from other objects. The entity-relationship model is widely used in database design. Chapter 6 explores it in detail.

• Semi-structured Data Model. The semi-structured data model permits the specifi cation of data where individual data items of the same type may have different sets of attributes. This is in contrast to the data models mentioned earlier, where every data item of a particular type must have the same set of attributes. *JSON* and *Extensible Markup Language* (*XML*) are widely used semi-structured data represen tations. Semi-structured data models are explored in detail in Chapter 8.

1.3 View of Data 9

• Object-Based Data Model. Object-oriented programming (especially in Java, C++, or C#) has become the dominant software-development methodology. This led initially to the development of a distinct object-oriented data model, but today the concept of objects is well integrated into relational databases. Standards exist to store objects in relational tables. Database systems allow procedures to be stored in the database system and executed by the database system. This can be seen as extending the relational model with notions of encapsulation, methods, and object identity. Object-based data models are summarized in Chapter 8.

A large portion of this text is focused on the relational model because it serves as the foundation for most database applications.

**1.3.2 Relational Data Model**

In the relational model, data are represented in the form of tables. Each table has mul tiple columns, and each column has a unique name. Each row of the table represents one piece of information. Figure 1.1 presents a sample relational database comprising two tables: one shows details of university instructors and the other shows details of the various university departments.

The first table, the *instructor* table, shows, for example, that an instructor named Einstein with *ID* 22222 is a member of the Physics department and has an annual salary of $95,000. The second table, *department*, shows, for example, that the Biology department is located in the Watson building and has a budget of $90,000. Of course, a real-world university would have many more departments and instructors. We use small tables in the text to illustrate concepts. A larger example for the same schema is available online.

**1.3.3 Data Abstraction**

For the system to be usable, it must retrieve data efficiently. The need for efficiency has led database system developers to use complex data structures to represent data in the database. Since many database-system users are not computer trained, developers hide the complexity from users through several levels of data abstraction, to simplify users’ interactions with the system:

• Physical level. The lowest level of abstraction describes *how* the data are actually stored. The physical level describes complex low-level data structures in detail. • Logical level. The next-higher level of abstraction describes *what* data are stored

in the database, and what relationships exist among those data. The logical level thus describes the entire database in terms of a small number of relatively simple structures. Although implementation of the simple structures at the logical level may involve complex physical-level structures, the user of the logical level does not need to be aware of this complexity. This is referred to as physical data indepen-

10 Chapter 1 Introduction

*ID name dept name salary*

22222 Einstein Physics 95000

12121 Wu Finance 90000

32343 El Said History 60000

45565 Katz Comp. Sci. 75000

98345 Kim Elec. Eng. 80000

76766 Crick Biology 72000

10101 Srinivasan Comp. Sci. 65000

58583 Califieri History 62000

83821 Brandt Comp. Sci. 92000

15151 Mozart Music 40000

33456 Gold Physics 87000

76543 Singh Finance 80000

(a) The *instructor* table

*dept name building budget*

Comp. Sci. Taylor 100000

Biology Watson 90000

Elec. Eng. Taylor 85000

Music Packard 80000

Finance Painter 120000

History Painter 50000

Physics Watson 70000

(b) The *department* table

**Figure 1.1** A sample relational database.

dence. Database administrators, who must decide what information to keep in the database, use the logical level of abstraction.

• View level. The highest level of abstraction describes only part of the entire database. Even though the logical level uses simpler structures, complexity remains because of the variety of information stored in a large database. Many users of the database system do not need all this information; instead, they need to access only a part of the database. The view level of abstraction exists to simplify their interac tion with the system. The system may provide many views for the same database.

Figure 1.2 shows the relationship among the three levels of abstraction.

An important feature of data models, such as the relational model, is that they hide such low-level implementation details from not just database users, but even from

view level

view 1 view 2

logical

level

physical

level

1.3 View of Data 11

… view *n*

**Figure 1.2** The three levels of data abstraction.

database-application developers. The database system allows application developers to store and retrieve data using the abstractions of the data model, and converts the abstract operations into operations on the low-level implementation.

An analogy to the concept of data types in programming languages may clarify the distinction among levels of abstraction. Many high-level programming languages support the notion of a structured type. We may describe the type of a record abstractly as follows:1

type *instructor* = record

*ID* : char (5);

*name* : char (20);

*dept name* : char (20);

*salary* : numeric (8,2);

end;

This code defines a new record type called *instructor* with four fields. Each field has a name and a type associated with it. For example, char(20) specifies a string with 20 characters, while numeric(8,2) specifies a number with 8 digits, two of which are to the right of the decimal point. A university organization may have several such record types, including:

• *department*, with fields *dept name*, *building*, and *budget*.

• *course*, with fields *course id*, *title*, *dept name*, and *credits*.

• *student*, with fields *ID*, *name*, *dept name*, and *tot cred*.

1The actual type declaration depends on the language being used. C and C++ use struct declarations. Java does not have such a declaration, but a simple class can be defined to the same effect.

12 Chapter 1 Introduction

At the physical level, an *instructor*, *department*, or *student* record can be described as a block of consecutive bytes. The compiler hides this level of detail from program mers. Similarly, the database system hides many of the lowest-level storage details from database programmers. Database administrators, on the other hand, may be aware of certain details of the physical organization of the data. For example, there are many possible ways to store tables in files. One way is to store a table as a sequence of records in a file, with a special character (such as a comma) used to delimit the different at tributes of a record, and another special character (such as a new-line character) may be used to delimit records. If all attributes have fixed length, the lengths of attributes may be stored separately, and delimiters may be omitted from the file. Variable length attributes could be handled by storing the length, followed by the data. Databases use a type of data structure called an index to support efficient retrieval of records; these too form part of the physical level.

At the logical level, each such record is described by a type definition, as in the previous code segment. The interrelationship of these record types is also defined at the logical level; a requirement that the *dept name* value of an *instructor* record must appear in the *department* table is an example of such an interrelationship. Programmers using a programming language work at this level of abstraction. Similarly, database administrators usually work at this level of abstraction.

Finally, at the view level, computer users see a set of application programs that hide details of the data types. At the view level, several views of the database are defined, and a database user sees some or all of these views. In addition to hiding details of the logical level of the database, the views also provide a security mechanism to prevent users from accessing certain parts of the database. For example, clerks in the university registrar office can see only that part of the database that has information about students; they cannot access information about salaries of instructors.

**1.3.4 Instances and Schemas**

Databases change over time as information is inserted and deleted. The collection of information stored in the database at a particular moment is called an instance of the database. The overall design of the database is called the database schema. The con cept of database schemas and instances can be understood by analogy to a program written in a programming language. A database schema corresponds to the variable declarations (along with associated type definitions) in a program. Each variable has a particular value at a given instant. The values of the variables in a program at a point in time correspond to an *instance* of a database schema.

Database systems have several schemas, partitioned according to the levels of ab straction. The physical schema describes the database design at the physical level, while the logical schema describes the database design at the logical level. A database may also have several schemas at the view level, sometimes called subschemas, that describe different views of the database.

Of these, the logical schema is by far the most important in terms of its effect on application programs, since programmers construct applications by using the logical

1.4 Database Languages 13

schema. The physical schema is hidden beneath the logical schema and can usually be changed easily without affecting application programs. Application programs are said to exhibit physical data independence if they do not depend on the physical schema and thus need not be rewritten if the physical schema changes.

We also note that it is possible to create schemas that have problems, such as unnecessarily duplicated information. For example, suppose we store the department *budget* as an attribute of the *instructor* record. Then, whenever the value of the budget for a department (say the Physics department) changes, that change must be reflected in the records of all instructors associated with the department. In Chapter 7, we shall study how to distinguish good schema designs from bad schema designs.

Traditionally, logical schemas were changed infrequently, if at all. Many newer database applications, however, require more flexible logical schemas where, for ex ample, different records in a single relation may have different attributes.

**1.4 Database Languages**

A database system provides a data-definition language (DDL) to specify the database schema and a data-manipulation language (DML) to express database queries and up dates. In practice, the data-definition and data-manipulation languages are not two sep arate languages; instead they simply form parts of a single database language, such as the SQL language. Almost all relational database systems employ the SQL language, which we cover in great detail in Chapter 3, Chapter 4, and Chapter 5.

**1.4.1 Data-Definition Language**

We specify a database schema by a set of definitions expressed by a special language called a data-definition language (DDL). The DDL is also used to specify additional properties of the data.

We specify the storage structure and access methods used by the database system by a set of statements in a special type of DDL called a data storage and definition language. These statements define the implementation details of the database schemas, which are usually hidden from the users.

The data values stored in the database must satisfy certain consistency constraints. For example, suppose the university requires that the account balance of a department must never be negative. The DDL provides facilities to specify such constraints. The database system checks these constraints every time the database is updated. In general, a constraint can be an arbitrary predicate pertaining to the database. However, arbitrary predicates may be costly to test. Thus, database systems implement only those integrity constraints that can be tested with minimal overhead:

• Domain Constraints. A domain of possible values must be associated with every attribute (for example, integer types, character types, date/time types). Declaring an attribute to be of a particular domain acts as a constraint on the values that it

14 Chapter 1 Introduction

can take. Domain constraints are the most elementary form of integrity constraint. They are tested easily by the system whenever a new data item is entered into the database.

• Referential Integrity. There are cases where we wish to ensure that a value that appears in one relation for a given set of attributes also appears in a certain set of attributes in another relation (referential integrity). For example, the depart ment listed for each course must be one that actually exists in the university. More precisely, the *dept name* value in a *course* record must appear in the *dept name* attribute of some record of the *department* relation. Database modifications can cause violations of referential integrity. When a referential-integrity constraint is violated, the normal procedure is to reject the action that caused the violation.

• Authorization. We may want to differentiate among the users as far as the type of access they are permitted on various data values in the database. These differentia tions are expressed in terms of authorization, the most common being: read autho rization, which allows reading, but not modification, of data; insert authorization, which allows insertion of new data, but not modification of existing data; update authorization, which allows modification, but not deletion, of data; and delete au thorization, which allows deletion of data. We may assign the user all, none, or a combination of these types of authorization.

The processing of DDL statements, just like those of any other programming lan guage, generates some output. The output of the DDL is placed in the data dictionary, which contains metadata—that is, data about data. The data dictionary is considered to be a special type of table that can be accessed and updated only by the database sys tem itself (not a regular user). The database system consults the data dictionary before reading or modifying actual data.

**1.4.2 The SQL Data-Definition Language**

SQL provides a rich DDL that allows one to define tables with data types and integrity constraints.

For instance, the following SQL DDL statement defines the *department* table:

create table *department*

(*dept name* char (20),

*building* char (15),

*budget* numeric (12,2));

Execution of the preceding DDL statement creates the *department* table with three columns: *dept name*, *building*, and *budget*, each of which has a specific data type asso ciated with it. We discuss data types in more detail in Chapter 3.

The SQL DDL also supports a number of types of integrity constraints. For exam ple, one can specify that the *dept name* attribute value is a *primary key*, ensuring that no

1.4 Database Languages 15

two departments can have the same department name. As another example, one can specify that the *dept name* attribute value appearing in any *instructor* record must also appear in the *dept name* attribute of some record of the *department* table. We discuss SQL support for integrity constraints and authorizations in Chapter 3 and Chapter 4.

**1.4.3 Data-Manipulation Language**

A data-manipulation language (DML) is a language that enables users to access or ma nipulate data as organized by the appropriate data model. The types of access are:

• Retrieval of information stored in the database.

• Insertion of new information into the database.

• Deletion of information from the database.

• Modification of information stored in the database.

There are basically two types of data-manipulation language:

• Procedural DMLs require a user to specify *what* data are needed and *how* to get those data.

• Declarative DMLs (also referred to as nonprocedural DMLs) require a user to spec ify *what* data are needed *without* specifying how to get those data.

Declarative DMLs are usually easier to learn and use than are procedural DMLs. However, since a user does not have to specify how to get the data, the database system has to figure out an efficient means of accessing data.

A query is a statement requesting the retrieval of information. The portion of a DML that involves information retrieval is called a query language. Although technically incorrect, it is common practice to use the terms *query language* and *data-manipulation language* synonymously.

There are a number of database query languages in use, either commercially or experimentally. We study the most widely used query language, SQL, in Chapter 3 through Chapter 5.

The levels of abstraction that we discussed in Section 1.3 apply not only to defining or structuring data, but also to manipulating data. At the physical level, we must define algorithms that allow efficient access to data. At higher levels of abstraction, we em phasize ease of use. The goal is to allow humans to interact efficiently with the system. The query processor component of the database system (which we study in Chapter 15 and Chapter 16) translates DML queries into sequences of actions at the physical level of the database system. In Chapter 22, we study the processing of queries in the increasingly common parallel and distributed settings.

16 Chapter 1 Introduction

**1.4.4 The SQL Data-Manipulation Language**

The SQL query language is nonprocedural. A query takes as input several tables (pos sibly only one) and always returns a single table. Here is an example of an SQL query that finds the names of all instructors in the History department:

select *instructor*.*name*

from *instructor*

where *instructor*.*dept name* = 'History';

The query specifies that those rows from the table *instructor* where the *dept name* is History must be retrieved, and the *name* attribute of these rows must be displayed. The result of executing this query is a table with a single column labeled *name* and a set of rows, each of which contains the name of an instructor whose *dept name* is History. If the query is run on the table in Figure 1.1, the result consists of two rows, one with the name El Said and the other with the name Califieri.

Queries may involve information from more than one table. For instance, the fol lowing query finds the instructor ID and department name of all instructors associated with a department with a budget of more than $95,000.

select *instructor*.*ID*, *department*.*dept name*

from *instructor*, *department*

where *instructor*.*dept name*= *department*.*dept name* and

*department*.*budget >* 95000;

If the preceding query were run on the tables in Figure 1.1, the system would find that there are two departments with a budget of greater than $95,000—Computer Science and Finance; there are five instructors in these departments. Thus, the result consists of a table with two columns (*ID*, *dept name*) and five rows: (12121, Finance), (45565, Com puter Science), (10101, Computer Science), (83821, Computer Science), and (76543, Finance).

**1.4.5 Database Access from Application Programs**

Non-procedural query languages such as SQL are not as powerful as a universal Turing machine; that is, there are some computations that are possible using a general-purpose programming language but are not possible using SQL. SQL also does not support ac tions such as input from users, output to displays, or communication over the network. Such computations and actions must be written in a *host* language, such as C/C++, Java, or Python, with embedded SQL queries that access the data in the database. Application programs are programs that are used to interact with the database in this fashion. Examples in a university system are programs that allow students to register for courses, generate class rosters, calculate student GPA, generate payroll checks, and perform other tasks.

1.5 Database Design 17

To access the database, DML statements need to be sent from the host to the database where they will be executed. This is most commonly done by using an application-program interface (set of procedures) that can be used to send DML and DDL statements to the database and retrieve the results. The Open Database Con nectivity (ODBC) standard defines application program interfaces for use with C and several other languages. The Java Database Connectivity (JDBC) standard defines a corresponding interface for the Java language.

**1.5 Database Design**

Database systems are designed to manage large bodies of information. These large bodies of information do not exist in isolation. They are part of the operation of some enterprise whose end product may be information from the database or may be some device or service for which the database plays only a supporting role.

Database design mainly involves the design of the database schema. The design of a complete database application environment that meets the needs of the enterprise being modeled requires attention to a broader set of issues. In this text, we focus on the writing of database queries and the design of database schemas, but discuss application design later, in Chapter 9.

A high-level data model provides the database designer with a conceptual frame work in which to specify the data requirements of the database users and how the database will be structured to fulfill these requirements. The initial phase of database design, then, is to characterize fully the data needs of the prospective database users. The database designer needs to interact extensively with domain experts and users to carry out this task. The outcome of this phase is a specification of user requirements.

Next, the designer chooses a data model, and by applying the concepts of the cho sen data model, translates these requirements into a conceptual schema of the database. The schema developed at this conceptual-design phase provides a detailed overview of the enterprise. The designer reviews the schema to confirm that all data requirements are indeed satisfied and are not in conflict with one another. The designer can also examine the design to remove any redundant features. The focus at this point is on describing the data and their relationships, rather than on specifying physical storage details.

In terms of the relational model, the conceptual-design process involves decisions on *what* attributes we want to capture in the database and *how to group* these attributes to form the various tables. The “what” part is basically a business decision, and we shall not discuss it further in this text. The “how” part is mainly a computer-science problem. There are principally two ways to tackle the problem. The first one is to use the entity-relationship model (Chapter 6); the other is to employ a set of algorithms (collectively known as normalization that takes as input the set of all attributes and generates a set of tables (Chapter 7).

A fully developed conceptual schema indicates the functional requirements of the enterprise. In a specification of functional requirements, users describe the kinds of oper-

18 Chapter 1 Introduction

ations (or transactions) that will be performed on the data. Example operations include modifying or updating data, searching for and retrieving specific data, and deleting data. At this stage of conceptual design, the designer can review the schema to ensure it meets functional requirements.

The process of moving from an abstract data model to the implementation of the database proceeds in two final design phases. In the logical-design phase, the de signer maps the high-level conceptual schema onto the implementation data model of the database system that will be used. The designer uses the resulting system-specific database schema in the subsequent physical-design phase, in which the physical features of the database are specified. These features include the form of file organization and the internal storage structures; they are discussed in Chapter 13.

**1.6 Database Engine**

A database system is partitioned into modules that deal with each of the responsibilities of the overall system. The functional components of a database system can be broadly divided into the storage manager, the query processor components, and the transaction management component.

The storage manager is important because databases typically require a large amount of storage space. Corporate databases commonly range in size from hundreds of gigabytes to terabytes of data. A gigabyte is approximately 1 billion bytes, or 1000 megabytes (more precisely, 1024 megabytes), while a terabyte is approximately 1 tril lion bytes or 1 million megabytes (more precisely, 1024 gigabytes). The largest enter prises have databases that reach into the multi-petabyte range (a petabyte is 1024 ter abytes). Since the main memory of computers cannot store this much information, and since the contents of main memory are lost in a system crash, the information is stored on disks. Data are moved between disk storage and main memory as needed. Since the movement of data to and from disk is slow relative to the speed of the central process ing unit, it is imperative that the database system structure the data so as to minimize the need to move data between disk and main memory. Increasingly, solid-state disks (SSDs) are being used for database storage. SSDs are faster than traditional disks but also more costly.

The query processor is important because it helps the database system to simplify and facilitate access to data. The query processor allows database users to obtain good performance while being able to work at the view level and not be burdened with un derstanding the physical-level details of the implementation of the system. It is the job of the database system to translate updates and queries written in a nonprocedural language, at the logical level, into an efficient sequence of operations at the physical level.

The transaction manager is important because it allows application developers to treat a sequence of database accesses as if they were a single unit that either happens in its entirety or not at all. This permits application developers to think at a higher level of

1.6 Database Engine 19

abstraction about the application without needing to be concerned with the lower-level details of managing the effects of concurrent access to the data and of system failures. While database engines were traditionally centralized computer systems, today parallel processing is key for handling very large amounts of data efficiently. Modern database engines pay a lot of attention to parallel data storage and parallel query pro cessing.

**1.6.1 Storage Manager**

The storage manager is the component of a database system that provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system. The storage manager is responsible for the interaction with the file manager. The raw data are stored on the disk using the file system provided by the operating system. The storage manager translates the various DML statements into low-level file-system commands. Thus, the storage manager is responsible for stor ing, retrieving, and updating data in the database.

The storage manager components include:

• Authorization and integrity manager, which tests for the satisfaction of integrity constraints and checks the authority of users to access data.

• Transaction manager, which ensures that the database remains in a consistent (cor rect) state despite system failures, and that concurrent transaction executions pro ceed without conflicts.

• File manager, which manages the allocation of space on disk storage and the data structures used to represent information stored on disk.

• Buffer manager, which is responsible for fetching data from disk storage into main memory, and deciding what data to cache in main memory. The buffer manager is a critical part of the database system, since it enables the database to handle data sizes that are much larger than the size of main memory.

The storage manager implements several data structures as part of the physical system implementation:

• Data files, which store the database itself.

• Data dictionary, which stores metadata about the structure of the database, in particular the schema of the database.

• Indices, which can provide fast access to data items. Like the index in this textbook, a database index provides pointers to those data items that hold a particular value. For example, we could use an index to find the *instructor* record with a particular *ID*, or all *instructor* records with a particular *name*.

20 Chapter 1 Introduction

We discuss storage media, file structures, and buffer management in Chapter 12 and Chapter 13. Methods of accessing data efficiently are discussed in Chapter 14.

**1.6.2 The Query Processor**

The query processor components include:

• DDL interpreter, which interprets DDL statements and records the definitions in the data dictionary.

• DML compiler, which translates DML statements in a query language into an eval uation plan consisting of low-level instructions that the query-evaluation engine understands.

A query can usually be translated into any of a number of alternative evalua tion plans that all give the same result. The DML compiler also performs query optimization; that is, it picks the lowest cost evaluation plan from among the alter natives.

• Query evaluation engine, which executes low-level instructions generated by the DML compiler.

Query evaluation is covered in Chapter 15, while the methods by which the query opti mizer chooses from among the possible evaluation strategies are discussed in Chapter 16.

**1.6.3 Transaction Management**

Often, several operations on the database form a single logical unit of work. An exam ple is a funds transfer, as in Section 1.2, in which one account *A* is debited and another account *B* is credited. Clearly, it is essential that either both the credit and debit occur, or that neither occur. That is, the funds transfer must happen in its entirety or not at all. This all-or-none requirement is called atomicity. In addition, it is essential that the execution of the funds transfer preserves the consistency of the database. That is, the value of the sum of the balances of *A* and *B* must be preserved. This correctness require ment is called consistency. Finally, after the successful execution of a funds transfer, the new values of the balances of accounts *A* and *B* must persist, despite the possibility of system failure. This persistence requirement is called durability.

A transaction is a collection of operations that performs a single logical function in a database application. Each transaction is a unit of both atomicity and consistency. Thus, we require that transactions do not violate any database-consistency constraints. That is, if the database was consistent when a transaction started, the database must be consistent when the transaction successfully terminates. However, during the exe cution of a transaction, it may be necessary temporarily to allow inconsistency, since

1.7 Database and Application Architecture 21

either the debit of *A* or the credit of *B* must be done before the other. This temporary inconsistency, although necessary, may lead to difficulty if a failure occurs. It is the programmer’s responsibility to properly define the various transactions so that each preserves the consistency of the database. For example, the transaction to transfer funds from account *A* to account *B* could be defined to be composed of two separate programs: one that debits account *A* and another that credits account *B*. The execution of these two programs one after the other will indeed preserve consistency. However, each program by itself does not transform the database from a consistent state to a new consistent state. Thus, those programs are not transactions. Ensuring the atomicity and durability properties is the responsibility of the database system itself—specifically, of the recovery manager. In the absence of failures, all transactions complete successfully, and atomicity is achieved easily. However, be cause of various types of failure, a transaction may not always complete its execution successfully. If we are to ensure the atomicity property, a failed transaction must have no effect on the state of the database. Thus, the database must be restored to the state in which it was before the transaction in question started executing. The database sys tem must therefore perform failure recovery, that is, it must detect system failures and restore the database to the state that existed prior to the occurrence of the failure. Finally, when several transactions update the database concurrently, the consis tency of data may no longer be preserved, even though each individual transaction is correct. It is the responsibility of the concurrency-control manager to control the inter action among the concurrent transactions, to ensure the consistency of the database. The transaction manager consists of the concurrency-control manager and the recovery manager.

The basic concepts of transaction processing are covered in Chapter 17. The man agement of concurrent transactions is covered in Chapter 18. Chapter 19 covers failure recovery in detail.

The concept of a transaction has been applied broadly in database systems and applications. While the initial use of transactions was in financial applications, the concept is now used in real-time applications in telecommunication, as well as in the management of long-duration activities such as product design or administrative work flows.

**1.7 Database and Application Architecture**

We are now in a position to provide a single picture of the various components of a database system and the connections among them. Figure 1.3 shows the architecture of a database system that runs on a centralized server machine. The figure summarizes how different types of users interact with a database, and how the different components of a database engine are connected to each other.

The centralized architecture shown in Figure 1.3 is applicable to *shared-memory* server architectures, which have multiple CPUs and exploit parallel processing, but all

22 Chapter 1 Introduction

naive users

(tellers, agents,

web users)

application programmers

sophisticated users

(analysts)

database

administrators

use write use use

application

interfaces

application

program

object code

application

programs

compiler and linker

query evaluation engine

query

toolsadministration

tools

DML queries DDL interpreter

DML compiler

and organizer

query processor

buffer manager file manager authorization and integrity

manager

transaction

manager

storage manager disk storage

indices

data dictionary

data statistical data

**Figure 1.3** System structure.

the CPUs access a common shared memory. To scale up to even larger data volumes and even higher processing speeds, *parallel databases* are designed to run on a cluster consisting of multiple machines. Further, *distributed databases* allow data storage and query processing across multiple geographically separated machines.

1.7 Database and Application Architecture 23

In Chapter 20, we cover the general structure of modern computer systems, with a focus on parallel system architectures. Chapter 21 and Chapter 22 describe how query processing can be implemented to exploit parallel and distributed processing. Chapter 23 presents a number of issues that arise in processing transactions in a parallel or a distributed database and describes how to deal with each issue. The issues include how to store data, how to ensure atomicity of transactions that execute at multiple sites, how to perform concurrency control, and how to provide high availability in the presence of failures.

We now consider the architecture of applications that use databases as their back end. Database applications can be partitioned into two or three parts, as shown in Figure 1.4. Earlier-generation database applications used a two-tier architecture, where the application resides at the client machine, and invokes database system functionality at the server machine through query language statements.

In contrast, modern database applications use a three-tier architecture, where the client machine acts as merely a front end and does not contain any direct database calls; web browsers and mobile applications are the most commonly used application clients today. The front end communicates with an application server. The application server, in turn, communicates with a database system to access data. The business logic of the application, which says what actions to carry out under what conditions, is embedded in the application server, instead of being distributed across multiple clients. Three tier applications provide better security as well as better performance than two-tier applications.

user

application

network

database system

(a) Two-tier architecture

client server

user

application client

network

application server

database system

(b) Three-tier architecture

**Figure 1.4** Two-tier and three-tier architectures.

24 Chapter 1 Introduction

**1.8 Database Users and Administrators**

A primary goal of a database system is to retrieve information from and store new information in the database. People who work with a database can be categorized as database users or database administrators.

**1.8.1 Database Users and User Interfaces**

There are four different types of database-system users, differentiated by the way they expect to interact with the system. Different types of user interfaces have been designed for the different types of users.

• Na¨ıve users are unsophisticated users who interact with the system by using prede fined user interfaces, such as web or mobile applications. The typical user interface for na¨ıve users is a forms interface, where the user can fill in appropriate fields of the form. Na¨ıve users may also view read *reports* generated from the database.

As an example, consider a student, who during class registration period, wishes to register for a class by using a web interface. Such a user connects to a web application program that runs at a web server. The application first verifies the identity of the user and then allows her to access a form where she enters the desired information. The form information is sent back to the web application at the server, which then determines if there is room in the class (by retrieving information from the database) and if so adds the student information to the class roster in the database.

• Application programmers are computer professionals who write application pro grams. Application programmers can choose from many tools to develop user in terfaces.

• Sophisticated users interact with the system without writing programs. Instead, they form their requests either using a database query language or by using tools such as data analysis software. Analysts who submit queries to explore data in the database fall in this category.

**1.8.2 Database Administrator**

One of the main reasons for using DBMSs is to have central control of both the data and the programs that access those data. A person who has such central control over the system is called a database administrator (DBA). The functions of a DBA include:

• Schema definition. The DBA creates the original database schema by executing a set of data definition statements in the DDL.

• Storage structure and access-method definition. The DBA may specify some param eters pertaining to the physical organization of the data and the indices to be cre ated.

1.9 History of Database Systems 25

• Schema and physical-organization modification. The DBA carries out changes to the schema and physical organization to reflect the changing needs of the organiza tion, or to alter the physical organization to improve performance.

• Granting of authorization for data access. By granting different types of authoriza tion, the database administrator can regulate which parts of the database various users can access. The authorization information is kept in a special system struc ture that the database system consults whenever a user tries to access the data in the system.

• Routine maintenance. Examples of the database administrator’s routine mainte nance activities are:

° Periodically backing up the database onto remote servers, to prevent loss of data in case of disasters such as flooding.

° Ensuring that enough free disk space is available for normal operations, and upgrading disk space as required.

° Monitoring jobs running on the database and ensuring that performance is not degraded by very expensive tasks submitted by some users.

**1.9 History of Database Systems**

Information processing drives the growth of computers, as it has from the earliest days of commercial computers. In fact, automation of data processing tasks predates com puters. Punched cards, invented by Herman Hollerith, were used at the very beginning of the twentieth century to record U.S. census data, and mechanical systems were used to process the cards and tabulate results. Punched cards were later widely used as a means of entering data into computers.

Techniques for data storage and processing have evolved over the years:

• 1950s and early 1960s: Magnetic tapes were developed for data storage. Data processing tasks such as payroll were automated, with data stored on tapes. Pro cessing of data consisted of reading data from one or more tapes and writing data to a new tape. Data could also be input from punched card decks and output to printers. For example, salary raises were processed by entering the raises on punched cards and reading the punched card deck in synchronization with a tape containing the master salary details. The records had to be in the same sorted or der. The salary raises would be added to the salary read from the master tape and written to a new tape; the new tape would become the new master tape.

Tapes (and card decks) could be read only sequentially, and data sizes were much larger than main memory; thus, data-processing programs were forced to

26 Chapter 1 Introduction

process data in a particular order by reading and merging data from tapes and card decks.

• Late 1960s and early 1970s: Widespread use of hard disks in the late 1960s changed the scenario for data processing greatly, since hard disks allowed direct access to data. The position of data on disk was immaterial, since any location on disk could be accessed in just tens of milliseconds. Data were thus freed from the tyranny of sequentiality. With the advent of disks, the network and hierarchical data models were developed, which allowed data structures such as lists and trees to be stored on disk. Programmers could construct and manipulate these data structures.

A landmark paper by Edgar Codd in 1970 defined the relational model and non procedural ways of querying data in the relational model, and relational databases were born. The simplicity of the relational model and the possibility of hiding im plementation details completely from the programmer were enticing indeed. Codd later won the prestigious Association of Computing Machinery Turing Award for his work.

• Late 1970s and 1980s: Although academically interesting, the relational model was not used in practice initially because of its perceived performance disadvantages; relational databases could not match the performance of existing network and hierarchical databases. That changed with System R, a groundbreaking project at IBM Research that developed techniques for the construction of an efficient relational database system. The fully functional System R prototype led to IBM’s first relational database product, SQL/DS. At the same time, the Ingres system was being developed at the University of California at Berkeley. It led to a commercial product of the same name. Also around this time, the first version of Oracle was released. Initial commercial relational database systems, such as IBM DB2, Oracle, Ingres, and DEC Rdb, played a major role in advancing techniques for efficient processing of declarative queries.

By the early 1980s, relational databases had become competitive with network and hierarchical database systems even in the area of performance. Relational databases were so easy to use that they eventually replaced network and hierar chical databases. Programmers using those older models were forced to deal with many low-level implementation details, and they had to code their queries in a procedural fashion. Most importantly, they had to keep efficiency in mind when designing their programs, which involved a lot of effort. In contrast, in a rela tional database, almost all these low-level tasks are carried out automatically by the database system, leaving the programmer free to work at a logical level. Since at taining dominance in the 1980s, the relational model has reigned supreme among data models.

The 1980s also saw much research on parallel and distributed databases, as well as initial work on object-oriented databases.

1.9 History of Database Systems 27

• 1990s: The SQL language was designed primarily for decision support applica tions, which are query-intensive, yet the mainstay of databases in the 1980s was transaction-processing applications, which are update-intensive.

In the early 1990s, decision support and querying re-emerged as a major ap plication area for databases. Tools for analyzing large amounts of data saw a large growth in usage. Many database vendors introduced parallel database products in this period. Database vendors also began to add object-relational support to their databases.

The major event of the 1990s was the explosive growth of the World Wide Web. Databases were deployed much more extensively than ever before. Database systems now had to support very high transaction-processing rates, as well as very high reliability and 24 × 7 availability (availability 24 hours a day, 7 days a week, meaning no downtime for scheduled maintenance activities). Database systems also had to support web interfaces to data.

• 2000s: The types of data stored in database systems evolved rapidly during this period. Semi-structured data became increasingly important. XML emerged as a data-exchange standard. JSON, a more compact data-exchange format well suited for storing objects from JavaScript or other programming languages subsequently grew increasingly important. Increasingly, such data were stored in relational database systems as support for the XML and JSON formats was added to the major commercial systems. Spatial data (that is, data that include geographic in formation) saw widespread use in navigation systems and advanced applications. Database systems added support for such data.

Open-source database systems, notably PostgreSQL and MySQL saw increased use. “Auto-admin” features were added to database systems in order to allow au tomatic reconfiguration to adapt to changing workloads. This helped reduce the human workload in administering a database.

Social network platforms grew at a rapid pace, creating a need to manage data about connections between people and their posted data, that did not fit well into a tabular row-and-column format. This led to the development of graph databases.

In the latter part of the decade, the use of data analytics and data mining in enterprises became ubiquitous. Database systems were developed specifically to serve this market. These systems featured physical data organizations suitable for analytic processing, such as “column-stores,” in which tables are stored by column rather than the traditional row-oriented storage of the major commercial database systems.

The huge volumes of data, as well as the fact that much of the data used for analytics was textual or semi-structured, led to the development of programming frameworks, such as *map-reduce*, to facilitate application programmers’ use of par allelism in analyzing data. In time, support for these features migrated into tradi tional database systems. Even in the late 2010s, debate continued in the database

28 Chapter 1 Introduction

research community over the relative merits of a single database system serving both traditional transaction processing applications and the newer data-analysis applications versus maintaining separate systems for these roles.

The variety of new data-intensive applications and the need for rapid devel opment, particularly by startup firms, led to “NoSQL” systems that provide a lightweight form of data management. The name was derived from those systems’ lack of support for the ubiquitous database query language SQL, though the name is now often viewed as meaning “not only SQL.” The lack of a high-level query lan guage based on the relational model gave programmers greater flexibility to work with new types of data. The lack of traditional database systems’ support for strict data consistency provided more flexibility in an application’s use of distributed data stores. The NoSQL model of “eventual consistency” allowed for distributed copies of data to be inconsistent as long they would eventually converge in the absence of further updates.

• 2010s: The limitations of NoSQL systems, such as lack of support for consistency, and lack of support for declarative querying, were found acceptable by many ap plications (e.g., social networks), in return for the benefits they provided such as scalability and availability. However, by the early 2010s it was clear that the lim itations made life significantly more complicated for programmers and database administrators. As a result, these systems evolved to provide features to support stricter notions of consistency, while continuing to support high scalability and availability. Additionally, these systems increasingly support higher levels of ab straction to avoid the need for programmers to have to reimplement features that are standard in a traditional database system.

Enterprises are increasingly outsourcing the storage and management of their data. Rather than maintaining in-house systems and expertise, enterprises may store their data in “cloud” services that host data for various clients in multiple, widely distributed server farms. Data are delivered to users via web-based services. Other enterprises are outsourcing not only the storage of their data but also whole applications. In such cases, termed “software as a service,” the vendor not only stores the data for an enterprise but also runs (and maintains) the application software. These trends result in significant savings in costs, but they create new issues not only in responsibility for security breaches, but also in data ownership, particularly in cases where a government requests access to data.

The huge influence of data and data analytics in daily life has made the man agement of data a frequent aspect of the news. There is an unresolved tradeoff between an individual’s right of privacy and society’s need to know. Various na tional governments have put regulations on privacy in place. High-profile security breaches have created a public awareness of the challenges in cybersecurity and the risks of storing data.

1.10 Summary 29

**1.10 Summary**

• A database-management system (DBMS) consists of a collection of interrelated data and a collection of programs to access those data. The data describe one particular enterprise.

• The primary goal of a DBMS is to provide an environment that is both convenient and efficient for people to use in retrieving and storing information.

• Database systems are ubiquitous today, and most people interact, either directly or indirectly, with databases many times every day.

• Database systems are designed to store large bodies of information. The manage ment of data involves both the definition of structures for the storage of infor mation and the provision of mechanisms for the manipulation of information. In addition, the database system must provide for the safety of the information stored in the face of system crashes or attempts at unauthorized access. If data are to be shared among several users, the system must avoid possible anomalous results.

• A major purpose of a database system is to provide users with an abstract view of the data. That is, the system hides certain details of how the data are stored and maintained.

• Underlying the structure of a database is the data model: a collection of conceptual tools for describing data, data relationships, data semantics, and data constraints. • The relational data model is the most widely deployed model for storing data in databases. Other data models are the object-oriented model, the object-relational model, and semi-structured data models.

• A data-manipulation language (DML) is a language that enables users to access or manipulate data. Nonprocedural DMLs, which require a user to specify only what data are needed, without specifying exactly how to get those data, are widely used today.

• A data-definition language (DDL) is a language for specifying the database schema and other properties of the data.

• Database design mainly involves the design of the database schema. The entity relationship (E-R) data model is a widely used model for database design. It pro vides a convenient graphical representation to view data, relationships, and con straints.

• A database system has several subsystems.

° The storage manager subsystem provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system.

30 Chapter 1 Introduction

° The query processor subsystem compiles and executes DDL and DML state ments. • Transaction management ensures that the database remains in a consistent (cor rect) state despite system failures. The transaction manager ensures that concur rent transaction executions proceed without conflicts.

• The architecture of a database system is greatly influenced by the underlying com puter system on which the database system runs. Database systems can be central ized, or parallel, involving multiple machines. Distributed databases span multiple geographically separated machines.

• Database applications are typically broken up into a front-end part that runs at client machines and a part that runs at the backend. In two-tier architectures, the front end directly communicates with a database running at the back end. In three tier architectures, the back end part is itself broken up into an application server and a database server.

• There are four different types of database-system users, differentiated by the way they expect to interact with the system. Different types of user interfaces have been designed for the different types of users.

• Data-analysis techniques attempt to automatically discover rules and patterns from data. The field of data mining combines knowledge-discovery techniques invented by artificial intelligence researchers and statistical analysts with efficient imple mentation techniques that enable them to be used on extremely large databases.

**Review Terms**

• Database-management system (DBMS)

• Database-system applications • Online transaction processing • Data analytics

• File-processing systems • Data inconsistency

• Consistency constraints • Data abstraction

° Physical level

° Logical level

° View level

• Instance

• Schema

° Physical schema

° Logical schema

° Subschema

• Physical data independence • Data models

° Entity-relationship model ° Relational data model

° Semi-structured data model ° Object-based data model

• Database languages

° Data-definition language

° Data-manipulation language

⋄ Procedural DML

⋄ Declarative DML

⋄ nonprocedural DML

° Query language

• Data-definition language

° Domain Constraints

° Referential Integrity

° Authorization

⋄ Read authorization

⋄ Insert authorization

⋄ Update authorization

⋄ Delete authorization

• Metadata

• Application program

• Database design

° Conceptual design

° Normalization

° Specification of functional re quirements

° Physical-design phase

• Database Engine

° Storage manager

⋄ Authorization and integrity

manager

**Practice Exercises**

Practice Exercises 31

⋄ Transaction manager

⋄ File manager

⋄ Buffer manager

⋄ Data files

⋄ Data dictionary

⋄ Indices

° Query processor

⋄ DDL interpreter

⋄ DML compiler

⋄ Query optimization

⋄ Query evaluation engine

° Transactions

⋄ Atomicity

⋄ Consistency

⋄ Durability

⋄ Recovery manager

⋄ Failure recovery

⋄ Concurrency-control manager • Database Architecture

° Centralized

° Parallel

° Distributed

• Database Application Architecture ° Two-tier

° Three-tier

° Application server

• Database administrator (DBA)

1.1 This chapter has described several major advantages of a database system. What are two disadvantages?

1.2 List five ways in which the type declaration system of a language such as Java or C++ differs from the data definition language used in a database.

32 Chapter 1 Introduction

1.3 List six major steps that you would take in setting up a database for a particular enterprise.

1.4 Suppose you want to build a video site similar to YouTube. Consider each of the points listed in Section 1.2 as disadvantages of keeping data in a file-processing system. Discuss the relevance of each of these points to the storage of actual video data, and to metadata about the video, such as title, the user who uploaded it, tags, and which users viewed it.

1.5 Keyword queries used in web search are quite different from database queries. List key differences between the two, in terms of the way the queries are specified and in terms of what is the result of a query.

**Exercises**

1.6 List four applications you have used that most likely employed a database system to store persistent data.

1.7 List four significant differences between a file-processing system and a DBMS.

1.8 Explain the concept of physical data independence and its importance in database systems.

1.9 List five responsibilities of a database-management system. For each responsi bility, explain the problems that would arise if the responsibility were not dis charged.

1.10 List at least two reasons why database systems support data manipulation using a declarative query language such as SQL, instead of just providing a library of C or C++ functions to carry out data manipulation.

1.11 Assume that two students are trying to register for a course in which there is only one open seat. What component of a database system prevents both students from being given that last seat?

1.12 Explain the difference between two-tier and three-tier application architectures. Which is better suited for web applications? Why?

1.13 List two features developed in the 2000s and that help database systems handle data-analytics workloads.

1.14 Explain why NoSQL systems emerged in the 2000s, and briefly contrast their features with traditional database systems.

1.15 Describe at least three tables that might be used to store information in a social networking system such as Facebook.

Further Reading 33

**Tools**

There are a large number of commercial database systems in use today. The major ones include: IBM DB2 (www.ibm.com/software/data/db2), Ora cle (www.oracle.com), Microsoft SQL Server (www.microsoft.com/sql), IBM In formix (www.ibm.com/software/data/informix), SAP Adaptive Server Enterprise (formerly Sybase) (www.sap.com/products/sybase-ase.html), and SAP HANA (www.sap.com/products/hana.html). Some of these systems are available free for personal or non-commercial use, or for development, but are not free for actual deploy ment.

There are also a number of free/public domain database systems; widely used ones include MySQL (www.mysql.com), PostgreSQL (www.postgresql.org), and the em bedded database SQLite (www.sqlite.org).

A more complete list of links to vendor web sites and other information is available from the home page of this book, at db-book.com.

**Further Reading**

[Codd (1970)] is the landmark paper that introduced the relational model. Textbook coverage of database systems is provided by [O’Neil and O’Neil (2000)], [Ramakrish nan and Gehrke (2002)], [Date (2003)], [Kifer et al. (2005)], [Garcia-Molina et al. (2008)], and [Elmasri and Navathe (2016)], in addition to this textbook,

A review of accomplishments in database management and an assessment of future research challenges appears in [Abadi et al. (2016)]. The home page of the ACM Special Interest Group on Management of Data (www.acm.org/sigmod) provides a wealth of information about database research. Database vendor web sites (see the Tools section above) provide details about their respective products.

**Bibliography**

[Abadi et al. (2016)] D. Abadi, R. Agrawal, A. Ailamaki, M. Balazinska, P. A. Bernstein, M. J. Carey, S. Chaudhuri, J. Dean, A. Doan, M. J. Franklin, J. Gehrke, L. M. Haas, A. Y. Halevy, J. M. Hellerstein, Y. E. Ioannidis, H. Jagadish, D. Kossmann, S. Madden, S. Mehro tra, T. Milo, J. F. Naughton, R. Ramakrishnan, V. Markl, C. Olston, B. C. Ooi, C. RÂ´e, D. Suciu, M. Stonebraker, T. Walter, and J. Widom, “The Beckman Report on Database Research”, *Communications of the ACM*, Volume 59, Number 2 (2016), pages 92–99.

[Codd (1970)] E. F. Codd, “A Relational Model for Large Shared Data Banks”, *Communi cations of the ACM*, Volume 13, Number 6 (1970), pages 377–387.

[Date (2003)] C. J. Date, *An Introduction to Database Systems*, 8th edition, Addison Wesley (2003).

34 Chapter 1 Introduction

[Elmasri and Navathe (2016)] R. Elmasri and S. B. Navathe, *Fundamentals of Database Sys tems*, 7th edition, Addison Wesley (2016).

[Garcia-Molina et al. (2008)] H. Garcia-Molina, J. D. Ullman, and J. D. Widom, *Database Systems: The Complete Book*, 2nd edition, Prentice Hall (2008).

[Kifer et al. (2005)] M. Kifer, A. Bernstein, and P. Lewis, *Database Systems: An Application Oriented Approach, Complete Version*, 2nd edition, Addison Wesley (2005).

[O’Neil and O’Neil (2000)] P. O’Neil and E. O’Neil, *Database: Principles, Programming, Per formance*, 2nd edition, Morgan Kaufmann (2000).

[Ramakrishnan and Gehrke (2002)] R. Ramakrishnan and J. Gehrke, *Database Management Systems*, 3rd edition, McGraw Hill (2002).

**Credits**

The photo of the sailboats in the beginning of the chapter is due to ©Pavel Nes vadba/Shutterstock.

PART 1

RELATIONAL LANGUAGES

A data model is a collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints. The relational model uses a collection of tables to represent both data and the relationships among those data. Its conceptual simplicity has led to its widespread adoption; today a vast majority of database products are based on the relational model. The relational model describes data at the logical and view levels, abstracting away low-level details of data storage.

To make data from a relational database available to users, we have to address how users specify requests for retrieving and updating data. Several query languages have been developed for this task, which are covered in this part.

Chapter 2 introduces the basic concepts underlying relational databases, including the coverage of relational algebra—a formal query language that forms the basis for SQL. The language SQL is the most widely used relational query language today and is covered in great detail in this part.

Chapter 3 provides an overview of the SQL query language, including the SQL data definition, the basic structure of SQL queries, set operations, aggregate functions, nested subqueries, and modification of the database.

Chapter 4 provides further details of SQL, including join expressions, views, trans actions, integrity constraints that are enforced by the database, and authorization mechanisms that control what access and update actions can be carried out by a user.

Chapter 5 covers advanced topics related to SQL including access to SQL from pro gramming languages, functions, procedures, triggers, recursive queries, and advanced aggregation features.

35

***CHAPTER*** 2

Introduction to the Relational Model

The relational model remains the primary data model for commercial data-processing applications. It attained its primary position because of its simplicity, which eases the job of the programmer, compared to earlier data models such as the network model or the hierarchical model. It has retained this position by incorporating various new features and capabilities over its half-century of existence. Among those additions are object-relational features such as complex data types and stored procedures, support for XML data, and various tools to support semi-structured data. The relational model’s independence from any specific underlying low-level data structures has allowed it to persist despite the advent of new approaches to data storage, including modern column stores that are designed for large-scale data mining.

In this chapter, we first study the fundamentals of the relational model. A substan tial theory exists for relational databases. In Chapter 6 and Chapter 7, we shall examine aspects of database theory that help in the design of relational database schemas, while in Chapter 15 and Chapter 16 we discuss aspects of the theory dealing with efficient processing of queries. In Chapter 27, we study aspects of formal relational languages beyond our basic coverage in this chapter.

**2.1 Structure of Relational Databases**

A relational database consists of a collection of tables, each of which is assigned a unique name. For example, consider the *instructor* table of Figure 2.1, which stores information about instructors. The table has four column headers: *ID*, *name*, *dept name*, and *salary*. Each row of this table records information about an instructor, consisting of the instructor’s *ID*, *name*, *dept name*, and *salary*. Similarly, the *course* table of Figure 2.2 stores information about courses, consisting of a *course id*, *title*, *dept name*, and *credits*, for each course. Note that each instructor is identified by the value of the column *ID*, while each course is identified by the value of the column *course id*.

37

38 Chapter 2 Introduction to the Relational Model

*ID name dept name salary*

10101 Srinivasan Comp. Sci. 65000

12121 Wu Finance 90000

15151 Mozart Music 40000

22222 Einstein Physics 95000

32343 El Said History 60000

33456 Gold Physics 87000

45565 Katz Comp. Sci. 75000

58583 Califieri History 62000

76543 Singh Finance 80000

76766 Crick Biology 72000

83821 Brandt Comp. Sci. 92000

98345 Kim Elec. Eng. 80000

**Figure 2.1** The *instructor* relation.

Figure 2.3 shows a third table, *prereq*, which stores the prerequisite courses for each course. The table has two columns, *course id* and *prereq id*. Each row consists of a pair of course identifiers such that the second course is a prerequisite for the first course.

Thus, a row in the *prereq* table indicates that two courses are *related* in the sense that one course is a prerequisite for the other. As another example, when we consider the table *instructor*, a row in the table can be thought of as representing the relationship

*course id title dept name credits*

BIO-101 Intro. to Biology Biology 4

BIO-301 Genetics Biology 4

BIO-399 Computational Biology Biology 3

CS-101 Intro. to Computer Science Comp. Sci. 4

CS-190 Game Design Comp. Sci. 4

CS-315 Robotics Comp. Sci. 3

CS-319 Image Processing Comp. Sci. 3

CS-347 Database System Concepts Comp. Sci. 3

EE-181 Intro. to Digital Systems Elec. Eng. 3

FIN-201 Investment Banking Finance 3

HIS-351 World History History 3

MU-199 Music Video Production Music 3

PHY-101 Physical Principles Physics 4

**Figure 2.2** The *course* relation.

2.1 Structure of Relational Databases 39

*course id prereq id*

BIO-301 BIO-101

BIO-399 BIO-101

CS-190 CS-101

CS-315 CS-101

CS-319 CS-101

CS-347 CS-101

EE-181 PHY-101

**Figure 2.3** The *prereq* relation.

between a specified *ID* and the corresponding values for *name*, *dept name*, and *salary* values.

In general, a row in a table represents a *relationship* among a set of values. Since a table is a collection of such relationships, there is a close correspondence between the concept of *table* and the mathematical concept of *relation*, from which the relational data model takes its name. In mathematical terminology, a *tuple* is simply a sequence (or list) of values. A relationship between *n* values is represented mathematically by an *n-tuple* of values, that is, a tuple with *n* values, which corresponds to a row in a table.

Thus, in the relational model the term relation is used to refer to a table, while the term tuple is used to refer to a row. Similarly, the term attribute refers to a column of a table.

Examining Figure 2.1, we can see that the relation *instructor* has four attributes: *ID*, *name*, *dept name*, and *salary*.

We use the term relation instance to refer to a specific instance of a relation, that is, containing a specific set of rows. The instance of *instructor* shown in Figure 2.1 has 12 tuples, corresponding to 12 instructors.

In this chapter, we shall be using a number of different relations to illustrate the various concepts underlying the relational data model. These relations represent part of a university. To simplify our presentation, we exclude much of the data an actual university database would contain. We shall discuss criteria for the appropriateness of relational structures in great detail in Chapter 6 and Chapter 7.

The order in which tuples appear in a relation is irrelevant, since a relation is a *set* of tuples. Thus, whether the tuples of a relation are listed in sorted order, as in Figure 2.1, or are unsorted, as in Figure 2.4, does not matter; the relations in the two figures are the same, since both contain the same set of tuples. For ease of exposition, we generally show the relations sorted by their first attribute.

For each attribute of a relation, there is a set of permitted values, called the domain of that attribute. Thus, the domain of the *salary* attribute of the *instructor* relation is the set of all possible salary values, while the domain of the *name* attribute is the set of all possible instructor names.

40 Chapter 2 Introduction to the Relational Model

*ID name dept name salary*

22222 Einstein Physics 95000

12121 Wu Finance 90000

32343 El Said History 60000

45565 Katz Comp. Sci. 75000

98345 Kim Elec. Eng. 80000

76766 Crick Biology 72000

10101 Srinivasan Comp. Sci. 65000

58583 Califieri History 62000

83821 Brandt Comp. Sci. 92000

15151 Mozart Music 40000

33456 Gold Physics 87000

76543 Singh Finance 80000

**Figure 2.4** Unsorted display of the *instructor* relation.

We require that, for all relations *r*, the domains of all attributes of *r* be atomic. A domain is atomic if elements of the domain are considered to be indivisible units. For example, suppose the table *instructor* had an attribute *phone number*, which can store a set of phone numbers corresponding to the instructor. Then the domain of *phone number* would not be atomic, since an element of the domain is a set of phone numbers, and it has subparts, namely, the individual phone numbers in the set.

The important issue is not what the domain itself is, but rather how we use domain elements in our database. Suppose now that the *phone number* attribute stores a single phone number. Even then, if we split the value from the phone number attribute into a country code, an area code, and a local number, we would be treating it as a non-atomic value. If we treat each phone number as a single indivisible unit, then the attribute *phone number* would have an atomic domain.

The null value is a special value that signifies that the value is unknown or does not exist. For example, suppose as before that we include the attribute *phone number* in the *instructor* relation. It may be that an instructor does not have a phone number at all, or that the telephone number is unlisted. We would then have to use the null value to signify that the value is unknown or does not exist. We shall see later that null values cause a number of difficulties when we access or update the database, and thus they should be eliminated if at all possible. We shall assume null values are absent initially, and in Section 3.6 we describe the effect of nulls on different operations.

The relatively strict structure of relations results in several important practical ad vantages in the storage and processing of data, as we shall see. That strict structure is suitable for well-defined and relatively static applications, but it is less suitable for applications where not only data but also the types and structure of those data change over time. A modern enterprise needs to find a good balance between the efficiencies of structured data and those situations where a predetermined structure is limiting.

2.2 Database Schema 41

**2.2 Database Schema**

When we talk about a database, we must differentiate between the database schema, which is the logical design of the database, and the database instance, which is a snap shot of the data in the database at a given instant in time.

The concept of a relation corresponds to the programming-language notion of a variable, while the concept of a relation schema corresponds to the programming language notion of type definition.

In general, a relation schema consists of a list of attributes and their corresponding domains. We shall not be concerned about the precise definition of the domain of each attribute until we discuss the SQL language in Chapter 3.

The concept of a relation instance corresponds to the programming-language no tion of a value of a variable. The value of a given variable may change with time; simi larly the contents of a relation instance may change with time as the relation is updated. In contrast, the schema of a relation does not generally change.

Although it is important to know the difference between a relation schema and a relation instance, we often use the same name, such as *instructor*, to refer to both the schema and the instance. Where required, we explicitly refer to the schema or to the instance, for example “the *instructor* schema,” or “an instance of the *instructor* relation.” However, where it is clear whether we mean the schema or the instance, we simply use the relation name.

Consider the *department* relation of Figure 2.5. The schema for that relation is: *department* (*dept name*, *building*, *budget*)

Note that the attribute *dept name* appears in both the *instructor* schema and the *department* schema. This duplication is not a coincidence. Rather, using common at tributes in relation schemas is one way of relating tuples of distinct relations. For ex ample, suppose we wish to find the information about all the instructors who work in the Watson building. We look first at the *department* relation to find the *dept name* of all the departments housed in Watson. Then, for each such department, we look in

*dept name building budget*

Biology Watson 90000

Comp. Sci. Taylor 100000

Elec. Eng. Taylor 85000

Finance Painter 120000

History Painter 50000

Music Packard 80000

Physics Watson 70000

**Figure 2.5** The *department* relation.

42 Chapter 2 Introduction to the Relational Model

*course id sec id semester year building room number time slot id*

BIO-101 1 Summer 2017 Painter 514 B BIO-301 1 Summer 2018 Painter 514 A CS-101 1 Fall 2017 Packard 101 H CS-101 1 Spring 2018 Packard 101 F CS-190 1 Spring 2017 Taylor 3128 E CS-190 2 Spring 2017 Taylor 3128 A CS-315 1 Spring 2018 Watson 120 D CS-319 1 Spring 2018 Watson 100 B CS-319 2 Spring 2018 Taylor 3128 C CS-347 1 Fall 2017 Taylor 3128 A EE-181 1 Spring 2017 Taylor 3128 C FIN-201 1 Spring 2018 Packard 101 B HIS-351 1 Spring 2018 Painter 514 C MU-199 1 Spring 2018 Packard 101 D PHY-101 1 Fall 2017 Watson 100 A

**Figure 2.6** The *section* relation.

the *instructor* relation to find the information about the instructor associated with the corresponding *dept name*.

Each course in a university may be offered multiple times, across different semesters, or even within a semester. We need a relation to describe each individual offering, or section, of the class. The schema is:

*section* (*course id*, *sec id*, *semester*, *year*, *building*, *room number*, *time slot id*)

Figure 2.6 shows a sample instance of the *section* relation.

We need a relation to describe the association between instructors and the class sections that they teach. The relation schema to describe this association is:

*teaches* (*ID*, *course id*, *sec id*, *semester*, *year*)

Figure 2.7 shows a sample instance of the *teaches* relation.

As you can imagine, there are many more relations maintained in a real university database. In addition to those relations we have listed already, *instructor*, *department*, *course*, *section*, *prereq*, and *teaches*, we use the following relations in this text:

• *student* (*ID*, *name*, *dept name*, *tot cred*)

• *advisor* (*s id*, *i id*)

• *takes* (*ID*, *course id*, *sec id*, *semester*, *year*, *grade*)

2.3 Keys 43

*ID course id sec id semester year*

10101 CS-101 1 Fall 2017

10101 CS-315 1 Spring 2018

10101 CS-347 1 Fall 2017

12121 FIN-201 1 Spring 2018

15151 MU-199 1 Spring 2018

22222 PHY-101 1 Fall 2017

32343 HIS-351 1 Spring 2018

45565 CS-101 1 Spring 2018

45565 CS-319 1 Spring 2018

76766 BIO-101 1 Summer 2017

76766 BIO-301 1 Summer 2018

83821 CS-190 1 Spring 2017

83821 CS-190 2 Spring 2017

83821 CS-319 2 Spring 2018

98345 EE-181 1 Spring 2017

**Figure 2.7** The *teaches* relation.

• *classroom* (*building*, *room number*, *capacity*)

• *time slot* (*time slot id*, *day*, *start time*, *end time*)

**2.3 Keys**

We must have a way to specify how tuples within a given relation are distinguished. This is expressed in terms of their attributes. That is, the values of the attribute values of a tuple must be such that they can *uniquely identify* the tuple. In other words, no two tuples in a relation are allowed to have exactly the same value for all attributes.1

A superkey is a set of one or more attributes that, taken collectively, allow us to identify uniquely a tuple in the relation. For example, the *ID* attribute of the relation *instructor* is sufficient to distinguish one instructor tuple from another. Thus, *ID* is a superkey. The *name* attribute of *instructor*, on the other hand, is not a superkey, because several instructors might have the same name.

Formally, let *R* denote the set of attributes in the schema of relation *r*. If we say that a subset *K* of *R* is a *superkey* for *r*, we are restricting consideration to instances of relations *r* in which no two distinct tuples have the same values on all attributes in *K*. That is, if *t*1 and *t*2 are in *r* and *t*1 ≠ *t*2, then *t*1*.K* ≠ *t*2*.K*.

1Commercial database systems relax the requirement that a relation is a set and instead allow duplicate tuples. This is discussed further in Chapter 3.

44 Chapter 2 Introduction to the Relational Model

A superkey may contain extraneous attributes. For example, the combination of *ID* and *name* is a superkey for the relation *instructor*. If *K* is a superkey, then so is any superset of *K*. We are often interested in superkeys for which no proper subset is a superkey. Such minimal superkeys are called candidate keys.

It is possible that several distinct sets of attributes could serve as a candidate key. Suppose that a combination of *name* and *dept name* is sufficient to distinguish among members of the *instructor* relation. Then, both {*ID*} and {*name*, *dept name*} are candidate keys. Although the attributes *ID* and *name* together can distinguish *instructor* tuples, their combination, {*ID*, *name*}, does not form a candidate key, since the attribute *ID* alone is a candidate key.

We shall use the term primary key to denote a candidate key that is chosen by the database designer as the principal means of identifying tuples within a relation. A key (whether primary, candidate, or super) is a property of the entire relation, rather than of the individual tuples. Any two individual tuples in the relation are prohibited from having the same value on the key attributes at the same time. The designation of a key represents a constraint in the real-world enterprise being modeled. Thus, primary keys are also referred to as primary key constraints.

It is customary to list the primary key attributes of a relation schema before the other attributes; for example, the *dept name* attribute of *department* is listed first, since it is the primary key. Primary key attributes are also underlined.

Consider the *classroom* relation:

*classroom* (*building*, *room number*, *capacity*)

Here the primary key consists of two attributes, *building* and *room number*, which are underlined to indicate they are part of the primary key. Neither attribute by itself can uniquely identify a classroom, although together they uniquely identify a classroom. Also consider the *time slot* relation:

*time slot* (*time slot id*, *day*, *start time*, *end time*)

Each section has an associated *time slot id*. The *time slot* relation provides information on which days of the week, and at what times, a particular *time slot id* meets. For ex ample, *time slot id* 'A' may meet from 8.00 AM to 8.50 AM on Mondays, Wednesdays, and Fridays. It is possible for a time slot to have multiple sessions within a single day, at different times, so the *time slot id* and *day* together do not uniquely identify the tuple. The primary key of the *time slot* relation thus consists of the attributes *time slot id*, *day*, and *start time*, since these three attributes together uniquely identify a time slot for a course.

Primary keys must be chosen with care. As we noted, the name of a person is insuffi cient, because there may be many people with the same name. In the United States, the social security number attribute of a person would be a candidate key. Since non-U.S. residents usually do not have social security numbers, international enterprises must

2.3 Keys 45

generate their own unique identifiers. An alternative is to use some unique combination of other attributes as a key.

The primary key should be chosen such that its attribute values are never, or are very rarely, changed. For instance, the address field of a person should not be part of the primary key, since it is likely to change. Social security numbers, on the other hand, are guaranteed never to change. Unique identifiers generated by enterprises generally do not change, except if two enterprises merge; in such a case the same identifier may have been issued by both enterprises, and a reallocation of identifiers may be required to make sure they are unique.

Figure 2.8 shows the complete set of relations that we use in our sample university schema, with primary-key attributes underlined.

Next, we consider another type of constraint on the contents of relations, called foreign-key constraints. Consider the attribute *dept name* of the *instructor* relation. It would not make sense for a tuple in *instructor* to have a value for *dept name* that does not correspond to a department in the *department* relation. Thus, in any database instance, given any tuple, say *ta*, from the *instructor* relation, there must be some tuple, say *tb*, in the *department* relation such that the value of the *dept name* attribute of *ta* is the same as the value of the primary key, *dept name*, of *tb*.

A foreign-key constraint from attribute(s) *A* of relation *r*1 to the primary-key *B* of relation *r*2 states that on any database instance, the value of *A* for each tuple in *r*1 must also be the value of *B* for some tuple in *r*2. Attribute set *A* is called a foreign key from *r*1, referencing *r*2. The relation *r*1 is also called the referencing relation of the foreign-key constraint, and *r*2 is called the referenced relation.

For example, the attribute *dept name* in *instructor* is a foreign key from *instructor*, referencing *department*; note that *dept name* is the primary key of *department*. Similarly,

*classroom*(*building*, *room number*, *capacity*)

*department*(*dept name*, *building*, *budget*)

*course*(*course id*, *title*, *dept name*, *credits*)

*instructor*(*ID*, *name*, *dept name*, *salary*)

*section*(*course id*, *sec id*, *semester*, *year*, *building*, *room number*, *time slot id*) *teaches*(*ID*, *course id*, *sec id*, *semester*, *year*)

*student*(*ID*, *name*, *dept name*, *tot cred*)

*takes*(*ID*, *course id*, *sec id*, *semester*, *year*, *grade*)

*advisor*(*s ID*, *i ID*)

*time slot*(*time slot id*, *day*, *start time*, *end time*)

*prereq*(*course id*, *prereq id*)

**Figure 2.8** Schema of the university database.

46 Chapter 2 Introduction to the Relational Model

the attributes *building* and *room number* of the *section* relation together form a foreign key referencing the *classroom* relation.

Note that in a foreign-key constraint, the referenced attribute(s) must be the pri mary key of the referenced relation. The more general case, a referential-integrity con straint, relaxes the requirement that the referenced attributes form the primary key of the referenced relation.

As an example, consider the values in the *time slot id* attribute of the *section* re lation. We require that these values must exist in the *time slot id* attribute of the *time slot* relation. Such a requirement is an example of a referential integrity constraint. In general, a referential integrity constraint requires that the values appearing in specified attributes of any tuple in the referencing relation also appear in specified attributes of at least one tuple in the referenced relation.

Note that *time slot* does not form a primary key of the *time slot* relation, although it is a part of the primary key; thus, we cannot use a foreign-key constraint to enforce the above constraint. In fact, foreign-key constraints are a *special case* of referential integrity constraints, where the referenced attributes form the primary key of the referenced relation. Database systems today typically support foreign-key constraints, but they do not support referential integrity constraints where the referenced attribute is not a primary key.

**2.4 Schema Diagrams**

A database schema, along with primary key and foreign-key constraints, can be de picted by schema diagrams. Figure 2.9 shows the schema diagram for our university organization. Each relation appears as a box, with the relation name at the top in blue and the attributes listed inside the box.

Primary-key attributes are shown underlined. Foreign-key constraints appear as arrows from the foreign-key attributes of the referencing relation to the primary key of the referenced relation. We use a two-headed arrow, instead of a single-headed arrow, to indicate a referential integrity constraint that is not a foreign-key constraints. In Figure 2.9, the line with a two-headed arrow from *time slot id* in the *section* relation to *time slot id* in the *time slot* relation represents the referential integrity constraint from *section*.*time slot id* to *time slot*.*time slot id*.

Many database systems provide design tools with a graphical user interface for creating schema diagrams.2 We shall discuss a different diagrammatic representation of schemas, called the entity-relationship diagram, at length in Chapter 6; although there are some similarities in appearance, these two notations are quite different, and should not be confused for one another.

2The two-headed arrow notation to represent referential integrity constraints has been introduced by us and is not supported by any tool as far as we know; the notations for primary and foreign keys, however, are widely used.

|  |
| --- |

*section*

*takes*

*ID*

*course\_id sec\_id*

*semester year*

*grade*

2.5 Relational Query Languages 47

*student*

*ID*

*name*

*dept\_name*

*tot\_cred*

*course*

*course\_id*

*sec\_id*

*semester*

*year*

*building*

*room\_number time\_slot\_id*

*classroom*

*building*

*room\_number capacity*

*time\_slot*

*time\_slot\_id*

*day*

*start\_time*

*end\_time*

*teaches*

*ID*

*course\_id*

*sec\_id*

*semester*

*year*

*course\_id title*

*dept\_name credits*

*prereq*

*course\_id prereq\_id*

*department dept\_name*

*building*

*budget*

*instructor*

*ID*

*name*

*dept\_name*

*salary*

*advisor*

*s\_id*

*i\_id*

**Figure 2.9** Schema diagram for the university database.

**2.5 Relational Query Languages**

A query language is a language in which a user requests information from the database. These languages are usually on a level higher than that of a standard programming language. Query languages can be categorized as imperative, functional, or declarative. In an imperative query language, the user instructs the system to perform a specific sequence of operations on the database to compute the desired result; such languages usually have a notion of state variables, which are updated in the course of the compu tation.

In a functional query language, the computation is expressed as the evaluation of functions that may operate on data in the database or on the results of other functions; functions are side-effect free, and they do not update the program state.3 In a declara tive query language, the user describes the desired information without giving a specific sequence of steps or function calls for obtaining that information; the desired informa tion is typically described using some form of mathematical logic. It is the job of the database system to figure out how to obtain the desired information.

3The term *procedural language* has been used in earlier editions of the book to refer to languages based on procedure invocations, which include functional languages; however, the term is also widely used to refer to imperative languages. To avoid confusion we no longer use the term.

48 Chapter 2 Introduction to the Relational Model

There are a number of “pure” query languages.

• The *relational algebra*, which we describe in Section 2.6, is a functional query language.4 The relational algebra forms the theoretical basis of the SQL query lan guage.

• The tuple relational calculus and domain relational calculus, which we describe in Chapter 27 (available online) are declarative.

These query languages are terse and formal, lacking the “syntactic sugar” of commercial languages, but they illustrate the fundamental techniques for extracting data from the database.

Query languages used in practice, such as the SQL query language, include ele ments of the imperative, functional, and declarative approaches. We study the very widely used query language SQL in Chapter 3 through Chapter 5.

**2.6 The Relational Algebra**

The relational algebra consists of a set of operations that take one or two relations as input and produce a new relation as their result.

Some of these operations, such as the select, project, and rename operations, are called *unary* operations because they operate on one relation. The other operations, such as union, Cartesian product, and set difference, operate on pairs of relations and are, therefore, called *binary* operations.

Although the relational algebra operations form the basis for the widely used SQL query language, database systems do not allow users to write queries in relational alge bra. However, there are implementations of relational algebra that have been built for students to practice relational algebra queries. The website of our book, db-book.com, under the link titled Laboratory Material, provides pointers to a few such implementa tions.

It is worth recalling at this point that since a relation is a set of tuples, relations cannot contain duplicate tuples. In practice, however, tables in database systems are permitted to contain duplicates unless a specific constraint prohibits it. But, in dis cussing the formal relational algebra, we require that duplicates be eliminated, as is required by the mathematical definition of a set. In Chapter 3 we discuss how rela tional algebra can be extended to work on multisets, which are sets that can contain duplicates.

4Unlike modern functional languages, relational algebra supports only a small number of predefined functions, which define an algebra on relations.

2.6 The Relational Algebra 49

*ID name dept name salary*

22222 Einstein Physics 95000

33456 Gold Physics 87000

**Figure 2.10** Result of σ*dept name* =“Physics” (*instructor*).

**2.6.1 The Select Operation**

The select operation selects tuples that satisfy a given predicate. We use the lowercase Greek letter sigma (σ) to denote selection. The predicate appears as a subscript to σ. The argument relation is in parentheses after the σ. Thus, to select those tuples of the *instructor* relation where the instructor is in the “Physics” department, we write:

σ*dept name* =“Physics” (*instructor*)

If the *instructor* relation is as shown in Figure 2.1, then the relation that results from the preceding query is as shown in Figure 2.10.

We can find all instructors with salary greater than $90,000 by writing: σ*salary>*90000 (*instructor*)

In general, we allow comparisons using =, ≠, *<*, ≤, *>*, and ≥ in the selection pred icate. Furthermore, we can combine several predicates into a larger predicate by using the connectives *and* (∧), *or* (∨), and *not* (¬). Thus, to find the instructors in Physics with a salary greater than $90,000, we write:

σ*dept name* =“Physics” ∧ *salary>*90000 (*instructor*)

The selection predicate may include comparisons between two attributes. To illus trate, consider the relation *department*. To find all departments whose name is the same as their building name, we can write:

σ*dept name* =*building*(*department*)

**2.6.2 The Project Operation**

Suppose we want to list all instructors’ *ID*, *name*, and *salary*, but we do not care about the *dept name*. The project operation allows us to produce this relation. The project operation is a unary operation that returns its argument relation, with certain attributes left out. Since a relation is a set, any duplicate rows are eliminated. Projection is denoted by the uppercase Greek letter pi (Π). We list those attributes that we wish to appear in the result as a subscript to Π. The argument relation follows in parentheses. We write the query to produce such a list as:

Π*ID*, *name*, *salary*(*instructor*)

Figure 2.11 shows the relation that results from this query.

50 Chapter 2 Introduction to the Relational Model

*ID name salary*

10101 Srinivasan 65000

12121 Wu 90000

15151 Mozart 40000

22222 Einstein 95000

32343 El Said 60000

33456 Gold 87000

45565 Katz 75000

58583 Califieri 62000

76543 Singh 80000

76766 Crick 72000

83821 Brandt 92000

98345 Kim 80000

**Figure 2.11** Result of Π*ID*, *name*, *salary* (*instructor*).

The basic version of the project operator Π*L*(*E*) allows only attribute names to be present in the list *L*. A generalized version of the operator allows expressions involving attributes to appear in the list *L*. For example, we could use:

Π*ID*,*name*,*salary*∕12(*instructor*)

to get the monthly salary of each instructor.

**2.6.3 Composition of Relational Operations**

The fact that the result of a relational operation is itself a relation is important. Con sider the more complicated query “Find the names of all instructors in the Physics department.” We write:

Π*name* (σ*dept name* =“Physics” (*instructor*))

Notice that, instead of giving the name of a relation as the argument of the projection operation, we give an expression that evaluates to a relation.

In general, since the result of a relational-algebra operation is of the same type (relation) as its inputs, relational-algebra operations can be composed together into a relational-algebra expression. Composing relational-algebra operations into relational algebra expressions is just like composing arithmetic operations (such as +, −, ∗, and ÷) into arithmetic expressions.

**2.6.4 The Cartesian-Product Operation**

The Cartesian-product operation, denoted by a cross (×), allows us to combine infor mation from any two relations. We write the Cartesian product of relations *r*1 and *r*2 as *r*1 × *r*2.

2.6 The Relational Algebra 51

*instructor.ID name dept name salary teaches*.*ID course id sec id semester year*

10101 Srinivasan Comp. Sci. 65000 10101 CS-101 1 Fall 2017 10101 Srinivasan Comp. Sci. 65000 10101 CS-315 1 Spring 2018 10101 Srinivasan Comp. Sci. 65000 10101 CS-347 1 Fall 2017 10101 Srinivasan Comp. Sci. 65000 12121 FIN-201 1 Spring 2018 10101 Srinivasan Comp. Sci. 65000 15151 MU-199 1 Spring 2018 10101 Srinivasan Comp. Sci. 65000 22222 PHY-101 1 Fall 2017

... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... 12121 Wu Finance 90000 10101 CS-101 1 Fall 2017 12121 Wu Finance 90000 10101 CS-315 1 Spring 2018 12121 Wu Finance 90000 10101 CS-347 1 Fall 2017 12121 Wu Finance 90000 12121 FIN-201 1 Spring 2018 12121 Wu Finance 90000 15151 MU-199 1 Spring 2018 12121 Wu Finance 90000 22222 PHY-101 1 Fall 2017

... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... 15151 Mozart Music 40000 10101 CS-101 1 Fall 2017 15151 Mozart Music 40000 10101 CS-315 1 Spring 2018 15151 Mozart Music 40000 10101 CS-347 1 Fall 2017 15151 Mozart Music 40000 12121 FIN-201 1 Spring 2018 15151 Mozart Music 40000 15151 MU-199 1 Spring 2018 15151 Mozart Music 40000 22222 PHY-101 1 Fall 2017 ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... 22222 Einstein Physics 95000 10101 CS-101 1 Fall 2017 22222 Einstein Physics 95000 10101 CS-315 1 Spring 2018 22222 Einstein Physics 95000 10101 CS-347 1 Fall 2017 22222 Einstein Physics 95000 12121 FIN-201 1 Spring 2018 22222 Einstein Physics 95000 15151 MU-199 1 Spring 2018 22222 Einstein Physics 95000 22222 PHY-101 1 Fall 2017 ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ...

**Figure 2.12** Result of the Cartesian product *instructor* × *teaches*.

A Cartesian product of database relations differs in its definition slightly from the mathematical definition of a Cartesian product of sets. Instead of *r*1 × *r*2 producing pairs (*t*1, *t*2) of tuples from *r*1 and *r*2, the relational algebra concatenates *t*1 and *t*2 into a single tuple, as shown in Figure 2.12.

Since the same attribute name may appear in the schemas of both *r*1 and *r*2, we need to devise a naming schema to distinguish between these attributes. We do so here by attaching to an attribute the name of the relation from which the attribute originally came. For example, the relation schema for *r* = *instructor* × *teaches* is:

(*instructor*.*ID*, *instructor*.*name*, *instructor*.*dept name*, *instructor*.*salary*, *teaches*.*ID*, *teaches*.*course id*, *teaches*.*sec id*, *teaches*.*semester*, *teaches*.*year*)