DATABASE SYSTEM CONCEPTS

SIXTH EDITION

Abraham Silberschatz
Yale University

Henry F. Korth
Lehigh University

S. Sudarshan
Indian Institute of Technology, Bombay
In memory of my father Joseph Silberschatz
my mother Vera Silberschatz
and my grandparents Stepha and Aaron Rosenblum

Avi Silberschatz

To my wife, Joan
my children, Abigail and Joseph
and my parents, Henry and Frances

Hank Korth

To my wife, Sita
my children, Madhur and Advaith
and my mother, Indira

S. Sudarshan
Contents

Chapter 1  Introduction
1.1 Database-System Applications  1
1.2 Purpose of Database Systems  3
1.3 View of Data  6
1.4 Database Languages  9
1.5 Relational Databases  12
1.6 Database Design  15
1.7 Data Storage and Querying  20
1.8 Transaction Management  22
1.9 Database Architecture  23
1.10 Data Mining and Information Retrieval  25
1.11 Specialty Databases  26
1.12 Database Users and Administrators  27
1.13 History of Database Systems  29
1.14 Summary  31
Exercises  33
Bibliographical Notes  35

PART ONE  ■  RELATIONAL DATABASES

Chapter 2  Introduction to the Relational Model
2.1 Structure of Relational Databases  39
2.2 Database Schema  42
2.3 Keys  45
2.4 Schema Diagrams  46
2.5 Relational Query Languages  47
2.6 Relational Operations  48
2.7 Summary  52
Exercises  53
Bibliographical Notes  55

Chapter 3  Introduction to SQL
3.1 Overview of the SQL Query Language  57
3.2 SQL Data Definition  58
3.3 Basic Structure of SQL Queries  63
3.4 Additional Basic Operations  74
3.5 Set Operations  79
3.6 Null Values  83
3.7 Aggregate Functions  84
3.8 Nested Subqueries  90
3.9 Modification of the Database  98
3.10 Summary  104
Exercises  105
Bibliographical Notes  112
Chapter 4  Intermediate SQL
   4.1 Join Expressions  113
   4.2 Views  120
   4.3 Transactions  127
   4.4 Integrity Constraints  128
   4.5 SQL Data Types and Schemas  136
   4.6 Authorization  143
   4.7 Summary  150
       Exercises  152
       Bibliographical Notes  156

Chapter 5  Advanced SQL
   5.1 Accessing SQL From a Programming Language  157
   5.2 Functions and Procedures  173
   5.3 Triggers  180
   5.4 Recursive Queries**  187
   5.5 Advanced Aggregation Features**  192
   5.6 OLAP**  197
   5.7 Summary  209
       Exercises  211
       Bibliographical Notes  216

Chapter 6  Formal Relational Query Languages
   6.1 The Relational Algebra  217
   6.2 The Tuple Relational Calculus  239
   6.3 The Domain Relational Calculus  245
   6.4 Summary  248
       Exercises  249
       Bibliographical Notes  254

PART TWO  DATABASE DESIGN

Chapter 7  Database Design and the E-R Model
   7.1 Overview of the Design Process  259
   7.2 The Entity-Relationship Model  262
   7.3 Constraints  269
   7.4 Removing Redundant Attributes in Entity Sets  272
   7.5 Entity-Relationship Diagrams  274
   7.6 Reduction to Relational Schemas  283
   7.7 Entity-Relationship Design Issues  290
   7.8 Extended E-R Features  295
   7.9 Alternative Notations for Modeling Data  304
   7.10 Other Aspects of Database Design  310
   7.11 Summary  313
       Exercises  315
       Bibliographical Notes  321
## Chapter 12  Query Processing

12.1 Overview 537
12.2 Measures of Query Cost 540
12.3 Selection Operation 541
12.4 Sorting 546
12.5 Join Operation 549
12.6 Other Operations 563
12.7 Evaluation of Expressions 567
12.8 Summary 572

Exercises 574

Bibliographical Notes 577

## Chapter 13  Query Optimization

13.1 Overview 579
13.2 Transformation of Relational Expressions 582
13.3 Estimating Statistics of Expression Results 590
13.4 Choice of Evaluation Plans 598
13.5 Materialized Views** 607
13.6 Advanced Topics in Query Optimization** 612
13.7 Summary 615

Exercises 617

Bibliographical Notes 622

## PART FOUR   TRANSACTION MANAGEMENT

### Chapter 14  Transactions

14.1 Transaction Concept 627
14.2 A Simple Transaction Model 629
14.3 Storage Structure 632
14.4 Transaction Atomicity and Durability 633
14.5 Transaction Isolation 635
14.6 Serializability 641
14.7 Transaction Isolation and Atomicity 646
14.8 Transaction Isolation Levels 648
14.9 Implementation of Isolation Levels 650
14.10 Transactions as SQL Statements 653
14.11 Summary 655

Exercises 657

Bibliographical Notes 660

### Chapter 15  Concurrency Control

15.1 Lock-Based Protocols 661
15.2 Deadlock Handling 674
15.3 Multiple Granularity 679
15.4 Timestamp-Based Protocols 682
15.5 Validation-Based Protocols 686
15.6 Multiversion Schemes 689
15.7 Snapshot Isolation 692
15.8 Insert Operations, Delete Operations, and Predicate Reads 697
15.9 Weak Levels of Consistency in Practice 701
15.10 Concurrency in Index Structures** 704
15.11 Summary 708

Exercises 712

Bibliographical Notes 718
Chapter 16  Recovery System
16.1 Failure Classification  721
16.2 Storage  722
16.3 Recovery and Atomicity  726
16.4 Recovery Algorithm  735
16.5 Buffer Management  738
16.6 Failure with Loss of Nonvolatile Storage  743
16.7 Early Lock Release and Logical Undo Operations  744
16.8 ARIES**  750
16.9 Remote Backup Systems  756
16.10 Summary  759
Exercises  762
Bibliographical Notes  766

PART FIVE  ■  SYSTEM ARCHITECTURE

Chapter 17  Database-System Architectures
17.1 Centralized and Client–Server Architectures  769
17.2 Server System Architectures  772
17.3 Parallel Systems  777
17.4 Distributed Systems  784
17.5 Network Types  788
17.6 Summary  791
Exercises  793
Bibliographical Notes  794

Chapter 18  Parallel Databases
18.1 Introduction  797
18.2 I/O Parallelism  798
18.3 Interquery Parallelism  802
18.4 Intraquery Parallelism  803
18.5 Intraoperation Parallelism  804
18.6 Interoperation Parallelism  813
18.7 Query Optimization  814
18.8 Design of Parallel Systems  815
18.9 Parallelism on Multicore Processors  817
18.10 Summary  819
Exercises  821
Bibliographical Notes  824

Chapter 19  Distributed Databases
19.1 Homogeneous and Heterogeneous Databases  825
19.2 Distributed Data Storage  826
19.3 Distributed Transactions  830
19.4 Commit Protocols  832
19.5 Concurrency Control in Distributed Databases  839
19.6 Availability  847
19.7 Distributed Query Processing  854
19.8 Heterogeneous Distributed Databases  857
19.9 Cloud-Based Databases  861
19.10 Directory Systems  870
19.11 Summary  875
Exercises  879
Bibliographical Notes  883
### PART SIX  ■  DATA WAREHOUSING, DATA MINING, AND INFORMATION RETRIEVAL

#### Chapter 20  Data Warehousing and Mining

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>20.1 Decision-Support Systems</td>
<td>887</td>
</tr>
<tr>
<td>20.2 Data Warehousing</td>
<td>889</td>
</tr>
<tr>
<td>20.3 Data Mining</td>
<td>893</td>
</tr>
<tr>
<td>20.4 Classification</td>
<td>894</td>
</tr>
<tr>
<td>20.5 Association Rules</td>
<td>904</td>
</tr>
<tr>
<td>20.6 Other Types of Associations</td>
<td>906</td>
</tr>
<tr>
<td>20.7 Clustering</td>
<td>907</td>
</tr>
<tr>
<td>20.8 Other Forms of Data Mining</td>
<td>908</td>
</tr>
<tr>
<td>20.9 Summary</td>
<td>909</td>
</tr>
<tr>
<td>Exercises</td>
<td>911</td>
</tr>
<tr>
<td>Bibliographical Notes</td>
<td>914</td>
</tr>
</tbody>
</table>

#### Chapter 21  Information Retrieval

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>21.1 Overview</td>
<td>915</td>
</tr>
<tr>
<td>21.2 Relevance Ranking Using Terms</td>
<td>917</td>
</tr>
<tr>
<td>21.3 Relevance Using Hyperlinks</td>
<td>920</td>
</tr>
<tr>
<td>21.4 Synonyms, Homonyms, and Ontologies</td>
<td>925</td>
</tr>
<tr>
<td>21.5 Indexing of Documents</td>
<td>927</td>
</tr>
<tr>
<td>21.6 Measuring Retrieval Effectiveness</td>
<td>929</td>
</tr>
<tr>
<td>21.7 Crawling and Indexing the Web</td>
<td>930</td>
</tr>
<tr>
<td>21.8 Information Retrieval: Beyond Ranking of Pages</td>
<td>931</td>
</tr>
<tr>
<td>21.9 Directories and Categories</td>
<td>935</td>
</tr>
<tr>
<td>21.10 Summary</td>
<td>937</td>
</tr>
<tr>
<td>Exercises</td>
<td>939</td>
</tr>
<tr>
<td>Bibliographical Notes</td>
<td>941</td>
</tr>
</tbody>
</table>

### PART SEVEN  ■  SPECIALTY DATABASES

#### Chapter 22  Object-Based Databases

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>22.1 Overview</td>
<td>945</td>
</tr>
<tr>
<td>22.2 Complex Data Types</td>
<td>946</td>
</tr>
<tr>
<td>22.3 Structured Types and Inheritance in SQL</td>
<td>949</td>
</tr>
<tr>
<td>22.4 Table Inheritance</td>
<td>954</td>
</tr>
<tr>
<td>22.5 Array and Multiset Types in SQL</td>
<td>956</td>
</tr>
<tr>
<td>22.6 Object-Identity and Reference Types in SQL</td>
<td>961</td>
</tr>
<tr>
<td>22.7 Implementing O-R Features</td>
<td>963</td>
</tr>
<tr>
<td>22.8 Persistent Programming Languages</td>
<td>964</td>
</tr>
<tr>
<td>22.9 Object-Relational Mapping</td>
<td>973</td>
</tr>
<tr>
<td>22.10 Object-Oriented versus Object-Relational</td>
<td>973</td>
</tr>
<tr>
<td>22.11 Summary</td>
<td>975</td>
</tr>
<tr>
<td>Exercises</td>
<td>976</td>
</tr>
<tr>
<td>Bibliographical Notes</td>
<td>980</td>
</tr>
</tbody>
</table>

#### Chapter 23  XML

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>23.1 Motivation</td>
<td>981</td>
</tr>
<tr>
<td>23.2 Structure of XML Data</td>
<td>986</td>
</tr>
<tr>
<td>23.3 XML Document Schema</td>
<td>990</td>
</tr>
<tr>
<td>23.4 Querying and Transformation</td>
<td>998</td>
</tr>
<tr>
<td>23.5 Application Program Interfaces to XML</td>
<td>1008</td>
</tr>
<tr>
<td>23.6 Storage of XML Data</td>
<td>1009</td>
</tr>
<tr>
<td>23.7 XML Applications</td>
<td>1016</td>
</tr>
<tr>
<td>23.8 Summary</td>
<td>1019</td>
</tr>
<tr>
<td>Exercises</td>
<td>1021</td>
</tr>
<tr>
<td>Bibliographical Notes</td>
<td>1024</td>
</tr>
</tbody>
</table>
PART EIGHT  ■  ADVANCED TOPICS

Chapter 24  Advanced Application Development
24.1 Performance Tuning  1029
24.2 Performance Benchmarks  1045
24.3 Other Issues in Application Development  1048
24.4 Standardization  1051
24.5 Summary  1056
Exercises  1057
Bibliographical Notes  1059

Chapter 25  Spatial and Temporal Data and Mobility
25.1 Motivation  1061
25.2 Time in Databases  1062
25.3 Spatial and Geographic Data  1064
25.4 Multimedia Databases  1076
25.5 Mobility and Personal Databases  1079
25.6 Summary  1085
Exercises  1087
Bibliographical Notes  1089

Chapter 26  Advanced Transaction Processing
26.1 Transaction-Processing Monitors  1091
26.2 Transactional Workflows  1096
26.3 E-Commerce  1102
26.4 Main-Memory Databases  1105
26.5 Real-Time Transaction Systems  1108
26.6 Long-Duration Transactions  1109
26.7 Summary  1115
Exercises  1117
Bibliographical Notes  1119

PART NINE  ■  CASE STUDIES

Chapter 27  PostgreSQL
27.1 Introduction  1123
27.2 User Interfaces  1124
27.3 SQL Variations and Extensions  1126
27.4 Transaction Management in PostgreSQL  1137
27.5 Storage and Indexing  1146
27.6 Query Processing and Optimization  1151
27.7 System Architecture  1154
Bibliographical Notes  1155

Chapter 28  Oracle
28.1 Database Design and Querying Tools  1157
28.2 SQL Variations and Extensions  1158
28.3 Storage and Indexing  1162
28.4 Query Processing and Optimization  1172
28.5 Concurrency Control and Recovery  1180
28.6 System Architecture  1183
28.7 Replication, Distribution, and External Data  1188
28.8 Database Administration Tools  1189
28.9 Data Mining  1191
Bibliographical Notes  1191
Appendix D  Network Model (contents online)
D.1 Basic Concepts  D1
D.2 Data-Structure Diagrams  D2
D.3 The DBTG CODASYL Model  D7
D.4 DBTG Data-Retrieval Facility  D13
D.5 DBTG Update Facility  D20
D.6 DBTG Set-Processing Facility  D22
D.7 Mapping of Networks to Files  D27
D.8 Summary  D31
Exercises  D32
Bibliographical Notes  D35

Appendix E  Hierarchical Model (contents online)
E.1 Basic Concepts  E1
E.2 Tree-Structure Diagrams  E2
E.3 Data-Retrieval Facility  E13
E.4 Update Facility  E17
E.5 Virtual Records  E20
E.6 Mapping of Hierarchies to Files  E22
E.7 The IMS Database System  E24
E.8 Summary  E25
Exercises  E26
Bibliographical Notes  E29

Bibliography  1283
Index  1315
This page intentionally left blank
Preface

Database management has evolved from a specialized computer application to a central component of a modern computing environment, 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 undergraduate, 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, or Pascal. 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. Details of particular database systems are discussed in Part 9, “Case Studies.”

In this, the sixth edition of Database System Concepts, we have retained the overall 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. We have also taken into account trends in the teaching of database concepts and made adaptations to facilitate these trends where appropriate.
Organization

The text is organized in nine major parts, plus five appendices.

- **Overview** (Chapter 1). Chapter 1 provides a general overview of the nature and purpose 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 organization 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.

- **Part 1: Relational Databases** (Chapters 2 through 6). Chapter 2 introduces the relational model of data, covering basic concepts such as the structure of relational databases, database schemas, keys, schema diagrams, relational query languages, and relational operations. Chapters 3, 4, and 5 focus on the most influential of the user-oriented relational languages: SQL. Chapter 6 covers the formal relational query languages: relational algebra, tuple relational calculus, and domain relational calculus.

  The chapters in this part describe data manipulation: queries, updates, insertions, and deletions, assuming a schema design has been provided. Schema design issues are deferred to Part 2.

- **Part 2: Database Design** (Chapters 7 through 9). Chapter 7 provides an overview of the database-design process, with major emphasis on database design using 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 that we encounter 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 8 introduces the theory of relational 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 in Sections 8.1 through 8.3 without loss of continuity. Instructors covering the entire chapter will benefit from students having a good understanding of normalization concepts to motivate some of the challenging concepts of functional-dependency theory.

  Chapter 9 covers application design and development. This chapter emphasizes the construction of database applications with Web-based interfaces. In addition, the chapter covers application security.
• **Part 3: Data Storage and Querying** (Chapters 10 through 13). Chapter 10 deals with storage devices, files, and data-storage structures. A variety of data-access techniques are presented in Chapter 11, including B+-tree indices and hashing. Chapters 12 and 13 address query-evaluation algorithms and query optimization. These chapters provide an understanding of the internals of the storage and retrieval components of a database.

• **Part 4: Transaction Management** (Chapters 14 through 16). Chapter 14 focuses on the fundamentals of a transaction-processing system: atomicity, consistency, isolation, and durability. It provides an overview of the methods used to ensure these properties, including locking and snapshot isolation.

  Chapter 15 focuses on concurrency control and presents several techniques for ensuring serializability, including locking, timestamping, and optimistic (validation) techniques. The chapter also covers deadlock issues. Alternatives to serializability are covered, most notably the widely-used snapshot isolation, which is discussed in detail.

  Chapter 16 covers the primary techniques for ensuring correct transaction execution despite system crashes and storage failures. These techniques include logs, checkpoints, and database dumps. The widely-used ARIES algorithm is presented.

• **Part 5: System Architecture** (Chapters 17 through 19). Chapter 17 covers computer-system architecture, and describes the influence of the underlying computer system on the database system. We discuss centralized systems, client–server systems, and parallel and distributed architectures in this chapter.

  Chapter 18, on parallel databases, explores a variety of parallelization techniques, including I/O parallelism, interquery and intraquery parallelism, and interoperation and intraoperation parallelism. The chapter also describes parallel-system design.

  Chapter 19 covers distributed database systems, revisiting the issues of database design, transaction management, and query evaluation and optimization, in the context of distributed databases. The chapter also covers issues of system availability during failures, heterogeneous distributed databases, cloud-based databases, and distributed directory systems.

• **Part 6: Data Warehousing, Data Mining, and Information Retrieval** (Chapters 20 and 21). Chapter 20 introduces the concepts of data warehousing and data mining. Chapter 21 describes information-retrieval techniques for querying textual data, including hyperlink-based techniques used in Web search engines.

  Part 6 uses the modeling and language concepts from Parts 1 and 2, but does not depend on Parts 3, 4, or 5. It can therefore be incorporated easily into a course that focuses on SQL and on database design.
• **Part 7: Specialty Databases** (Chapters 22 and 23). Chapter 22 covers object-based databases. The chapter describes the object-relational data model, which extends the relational data model to support complex data types, type inheritance, and object identity. The chapter also describes database access from object-oriented programming languages.

Chapter 23 covers the XML standard for data representation, which is seeing increasing use in the exchange and storage of complex data. The chapter also describes query languages for XML.

• **Part 8: Advanced Topics** (Chapters 24 through 26). Chapter 24 covers advanced issues in application development, including performance tuning, performance benchmarks, database-application testing, and standardization.

Chapter 25 covers spatial and geographic data, temporal data, multimedia data, and issues in the management of mobile and personal databases.

Finally, Chapter 26 deals with advanced transaction processing. Topics covered in the chapter include transaction-processing monitors, transactional workflows, electronic commerce, high-performance transaction systems, real-time transaction systems, and long-duration transactions.

• **Part 9: Case Studies** (Chapters 27 through 30). In this part, we present case studies of four of the leading database systems, PostgreSQL, Oracle, IBM DB2, and Microsoft SQL Server. These chapters outline unique features of each of these systems, and describe their internal structure. They provide a wealth of interesting information about the respective products, and help you see how the various implementation techniques described in earlier parts are used in real systems. They also cover several interesting practical aspects in the design of real systems.

• **Appendices.** We provide five appendices that cover material that is of historical nature or is advanced; these appendices are available only online on the Web site of the book (http://www.db-book.com). An exception is Appendix A, which presents details of our university schema including the full schema, DDL, and all the tables. This appendix appears in the actual text.

Appendix B describes other relational query languages, including QBE Microsoft Access, and Datalog.

Appendix C describes advanced relational database design, including the theory of multivalued dependencies, join dependencies, and the project-join and domain-key normal forms. This appendix is for the benefit of individuals who wish to study the theory of relational database design in more detail, and instructors who wish to do so in their courses. This appendix, too, is available only online, on the Web site of the book.

Although most new database applications use either the relational model or the object-relational model, the network and hierarchical data models are still in use in some legacy applications. For the benefit of readers who wish to learn about these data models, we provide appendices describing the network and hierarchical data models, in Appendices D and E respectively.
The Sixth Edition

The production of this sixth 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 have replaced the earlier running example of bank enterprise with a university example. This example has an immediate intuitive connection to students that assists not only in remembering the example, but, more importantly, in gaining deeper insight into the various design decisions that need to be made.

We have reorganized the book so as to collect all of our SQL coverage together and place it early in the book. Chapters 3, 4, and 5 present complete SQL coverage. Chapter 3 presents the basics of the language, with more advanced features in Chapter 4. In Chapter 5, we present JDBC along with other means of accessing SQL from a general-purpose programming language. We present triggers and recursion, and then conclude with coverage of online analytic processing (OLAP). Introductory courses may choose to cover only certain sections of Chapter 5 or defer sections until after the coverage of database design without loss of continuity.

Beyond these two major changes, we revised the material in each chapter, bringing the older material up-to-date, adding discussions on recent developments in database technology, and improving descriptions of topics that students found difficult to understand. We have also added new exercises and updated references. The list of specific changes includes the following:

- **Earlier coverage of SQL.** Many instructors use SQL as a key component of term projects (see our Web site, [www.db-book.com](http://www.db-book.com), for sample projects). In order to give students ample time for the projects, particularly for universities and colleges on the quarter system, it is essential to teach SQL as early as possible. With this in mind, we have undertaken several changes in organization:
  - A new chapter on the relational model (Chapter 2) precedes SQL, laying the conceptual foundation, without getting lost in details of relational algebra.
  - Chapters 3, 4, and 5 provide detailed coverage of SQL. These chapters also discuss variants supported by different database systems, to minimize problems that students face when they execute queries on actual database systems. These chapters cover all aspects of SQL, including queries, data definition, constraint specification, OLAP, and the use of SQL from within a variety of languages, including Java/JDBC.
  - Formal languages (Chapter 6) have been postponed to after SQL, and can be omitted without affecting the sequencing of other chapters. Only our discussion of query optimization in Chapter 13 depends on the relational algebra coverage of Chapter 6.
• **New database schema.** We adopted a new schema, which is based on university data, as a running example throughout the book. This schema is more intuitive and motivating for students than the earlier bank schema, and illustrates more complex design trade-offs in the database-design chapters.

• **More support for a hands-on student experience.** To facilitate following our running example, we list the database schema and the sample relation instances for our university database together in Appendix A as well as where they are used in the various regular chapters. In addition, we provide, on our Web site http://www.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.

• **Revised coverage of E-R model.** The E-R diagram notation in Chapter 7 has been modified to make it more compatible with UML. The chapter also makes good use of the new university database schema to illustrate more complex design trade-offs.

• **Revised coverage of relational design.** Chapter 8 now has a more readable style, providing an intuitive understanding of functional dependencies and normalization, before covering functional dependency theory; the theory is motivated much better as a result.

• **Expanded material on application development and security.** Chapter 9 has new material on application development, mirroring rapid changes in the field. In particular, coverage of security has been expanded, considering its criticality in today’s interconnected world, with an emphasis on practical issues over abstract concepts.

• **Revised and updated coverage of data storage, indexing and query optimization.** Chapter 10 has been updated with new technology, including expanded coverage of flash memory.

  Coverage of B⁺-trees in Chapter 11 has been revised to reflect practical implementations, including coverage of bulk loading, and the presentation has been improved. The B⁺-tree examples in Chapter 11 have now been revised with \( n = 4 \), to avoid the special case of empty nodes that arises with the (unrealistic) value of \( n = 3 \).

  Chapter 13 has new material on advanced query-optimization techniques.

• **Revised coverage of transaction management.** Chapter 14 provides full coverage of the basics for an introductory course, with advanced details following in Chapters 15 and 16. Chapter 14 has been expanded to cover the practical issues in transaction management faced by database users and database-application developers. The chapter also includes an expanded overview of topics covered in Chapters 15 and 16, ensuring that even if Chapters 15 and 16 are omitted, students have a basic knowledge of the concepts of concurrency control and recovery.
Chapters 14 and 15 now include detailed coverage of snapshot isolation, which is widely supported and used today, including coverage of potential hazards when using it.

Chapter 16 now has a simplified description of basic log-based recovery leading up to coverage of the ARIES algorithm.

- **Revised and expanded coverage of distributed databases.** We now cover cloud data storage, which is gaining significant interest for business applications. Cloud storage offers enterprises opportunities for improved cost-management and increased storage scalability, particularly for Web-based applications. We examine those advantages along with the potential drawbacks and risks.

  Multidatabases, which were earlier in the advanced transaction processing chapter, are now covered earlier as part of the distributed database chapter.

- **Postponed coverage of object databases and XML.** Although object-oriented languages and XML are widely used outside of databases, their use in databases is still limited, making them appropriate for more advanced courses, or as supplementary material for an introductory course. These topics have therefore been moved to later in the book, in Chapters 22 and 23.

- **QBE, Microsoft Access, and Datalog in an online appendix.** These topics, which were earlier part of a chapter on “other relational languages,” are now covered in online Appendix C.

All topics not listed above are updated from the fifth edition, though their overall organization is relatively unchanged.

**Review Material and Exercises**

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

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 information 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

The book contains both basic and advanced material, which might not be covered in a single semester. We have marked several sections as advanced, using the symbol “**”. These sections may be omitted if so desired, without a loss of continuity. Exercises that are difficult (and can be omitted) are also marked using the symbol “**”.

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:

- **Chapter 5 (Advanced SQL)** can be skipped or deferred to later without 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 (Formal Relational Query Languages)** can be covered immediately after Chapter 2, ahead of SQL. Alternatively, this chapter may be omitted from an introductory course.
  
  We recommend covering Section 6.1 (relational algebra) if the course also covers query processing. However, Sections 6.2 and 6.3 can be omitted if students will not be using relational calculus as part of the course.

- **Chapter 7 (E-R Model)** can be covered ahead of Chapters 3, 4 and 5 if you so desire, since Chapter 7 does not have any dependency on SQL.

- **Chapter 13 (Query Optimization)** can be omitted from an introductory course without affecting coverage of any other chapter.

- Both our coverage of transaction processing (Chapters 14 through 16) and our coverage of system architecture (Chapters 17 through 19) consist of an overview chapter (Chapters 14 and 17, respectively), followed by chapters with details. You might choose to use Chapters 14 and 17, while omitting Chapters 15, 16, 18 and 19, if you defer these latter chapters to an advanced course.

- Chapters 20 and 21, covering data warehousing, data mining, and information retrieval, can be used as self-study material or omitted from an introductory course.

- Chapters 22 (Object-Based Databases), and 23 (XML) can be omitted from an introductory course.

- Chapters 24 through 26, covering advanced application development, spatial, temporal and mobile data, and advanced transaction processing, are suitable for an advanced course or for self-study by students.

- **The case-study Chapters 27 through 30** are suitable for self-study by students. Alternatively, they can be used as an illustration of concepts when the earlier chapters are presented in class.

Model course syllabi, based on the text, can be found on the Web site of the book.
Web Site and Teaching Supplements


- Slides covering all the chapters of the book.
- Answers to the practice exercises.
- The five appendices.
- An up-to-date errata list.
- 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.

The following additional material is available only to faculty:

- An instructor 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 manual and the question bank, please send electronic mail to customer.service@mcgraw-hill.com. In the United States, you may call 800-338-3987. The McGraw-Hill Web site for this book is http://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 readers, 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 correspondence 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 sixth edition, as well as with the previous five editions from which it is derived.
Sixth Edition

- Anastassia Ailamaki, Sailesh Krishnamurthy, Spiros Papadimitriou, and Bianca Schroeder (Carnegie Mellon University) for writing Chapter 27 describing the PostgreSQL database system.
- Hakan Jakobsson (Oracle), for writing Chapter 28 on the Oracle database system.
- Sriram Padmanabhan (IBM), for writing Chapter 29 describing the IBM DB2 database system.
- Sameet Agarwal, José A. Blakeley, Thierry D’Hers, Gerald Hinson, Dirk Myers, Vaqar Pirzada, Bill Ramos, Balaji Rathakrishnan, Michael Rys, Florian Waas, and Michael Zwilling (all of Microsoft) for writing Chapter 30 describing the Microsoft SQL Server database system, and in particular José Blakeley for coordinating and editing the chapter; César Galindo-Legaria, Goetz Graefe, Kalen Delaney, and Thomas Casey (all of Microsoft) for their contributions to the previous edition of the Microsoft SQL Server chapter.
- Daniel Abadi for reviewing the table of contents of the fifth edition and helping with the new organization.
- Steve Dolins, University of Florida; Rolando Fernandez, George Washington University; Frantisek Franek, McMaster University; Latifur Khan, University of Texas - Dallas; Sanjay Madria, University of Missouri - Rolla; 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 formulating this sixth edition.
- 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, in particular Chapter 11; 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.

Previous Editions

- 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 Dupré 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 Bevviner 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.


Book Production

The publisher was Raghu Srinivasan. The developmental editor was Melinda D. Bilecki. The project manager was Melissa Leick. The marketing manager was
Curt Reynolds. The production supervisor was Laura Fuller. The book designer was Brenda Rolwes. The cover designer was Studio Montage, St. Louis, Missouri. The copyeditor was George Watson. The proofreader was Kevin Campbell. The freelance indexer was Tobiah Waldron. The Aptara team consisted of Raman Arora and Sudeshna Nandy

Personal Notes

Sudarshan would like to acknowledge his wife, Sita, for her love 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 understanding. 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.
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. Management of data involves both defining structures for storage of information and providing 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

Databases are widely used. Here are some representative applications:

- **Enterprise Information**
  - Sales: For customer, product, and purchase information.
  - Accounting: For payments, receipts, account balances, assets and other accounting 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.
Chapter 1  Introduction

- **Online retailers**: For sales data noted above plus online order tracking, generation of recommendation lists, and maintenance of online product evaluations.

- **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 financial 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 made, generating monthly bills, maintaining balances on prepaid calling cards, and storing information about the communication networks.

As the list illustrates, databases form an essential part of every enterprise today, storing not only types of information that are common to most enterprises, but also information that is specific to the category of the enterprise.

Over the course of the last four decades of the twentieth century, use of databases grew in all enterprises. In the early days, very few people interacted directly with database systems, although without realizing it, they interacted with databases indirectly—through printed reports such as credit card statements, or through agents such as bank tellers and airline reservation agents. Then automated teller machines came along and let users interact directly with databases. Phone interfaces to computers (interactive voice-response systems) also allowed users to deal directly with databases—a caller could dial a number, and press phone keys to enter information or to select alternative options, to find flight arrival/departure times, for example, or to register for courses in a university.

The Internet revolution of the late 1990s sharply increased direct user access to databases. Organizations converted many of their phone interfaces to databases into Web interfaces, and made a variety of services and information available online. For instance, when you 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, informa-
tion about you may be retrieved from a database to select which advertisements you should see. 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.

The importance of database systems can be judged in another way—today, database system vendors like Oracle are among the largest software companies in the world, and database systems form an important part of the product line of Microsoft and IBM.

1.2 Purpose of Database Systems

Database systems arose in response to early methods of computerized management of commercial data. As an example of such methods, typical of the 1960s, consider part of a university organization that, among other data, keeps information about all instructors, students, departments, 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 transcripts

System programmers wrote these application programs to meet the needs of the university.

New application programs are added to the system as the need arises. For example, suppose that a university decides to create a new major (say, computer science). 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, students in that major, course offerings, degree requirements, etc. The university may have to write new application programs to deal with rules specific to the new major. 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. Before database management systems (DBMSs) were introduced, organizations usually stored information in such systems.

Keeping organizational information in a file-processing system has a number of major disadvantages:
• **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 languages. 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 has now 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 different 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 appropriate 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 consistent state that existed prior to the failure. Consider a program to transfer $500 from the account balance of department A to the account balance of department B. If a system failure occurs during the execution of the program, it is possible that the $500 was removed from the balance of department A but was not credited to the balance of department 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 simultaneously. Indeed, today, the largest Internet retailers may have millions of accesses per day to their data by shoppers. In such an environment, interaction of concurrent updates is possible and may result in inconsistent data. Consider department A, with an account balance of $10,000. If two department clerks debit the account balance (by say $500 and $100, respectively) of department A at almost exactly the same time, the result of the concurrent executions may leave the budget in an incorrect (or inconsistent) state. Suppose that the programs executing on behalf of each withdrawal 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 account balance of department 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 (say) 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.
These difficulties, among others, prompted the development of database systems. 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 Abstraction

For the system to be usable, it must retrieve data efficiently. The need for efficiency has led designers 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 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 independence. 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 interaction with the system. The system may provide many views for the same database.

Figure 1.1 shows the relationship among the three levels of abstraction.

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. For example, we may describe
a record as follows:\footnote{The actual type declaration depends on the language being used. C and C++ use \texttt{struct} declarations. Java does not have such a declaration, but a simple class can be defined to the same effect.}

\begin{verbatim}
  type instructor = record 
    ID : char (5);
    name : char (20);
    dept \_ name : char (20);
    salary : numeric (8,2);
  end;
\end{verbatim}

This code defines a new record type called \texttt{instructor} with four fields. Each field
has a name and a type associated with it. A university organization may have
several such record types, including

- \texttt{department}, with fields \texttt{dept\_name}, \texttt{building}, and \texttt{budget}
- \texttt{course}, with fields \texttt{course\_id}, \texttt{title}, \texttt{dept\_name}, and \texttt{credits}
- \texttt{student}, with fields \texttt{ID}, \texttt{name}, \texttt{dept\_name}, and \texttt{tot\_cred}

At the physical level, an \texttt{instructor}, \texttt{department}, or \texttt{student} record can be de-
scribed as a block of consecutive storage locations. The compiler hides this level
of detail from programmers. Similarly, the database system hides many of the
lowest-level storage details from database programmers. Database administra-
tors, on the other hand, may be aware of certain details of the physical organiza-
tion of the data.
Chapter 1  Introduction

At the logical level, each such record is described by a type definition, as in the previous code segment, and the interrelationship of these record types is defined as well. 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.

1.3.2 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. Schemas are changed infrequently, if at all.

The concept 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 abstraction. 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 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 study languages for describing schemas after introducing the notion of data models in the next section.

1.3.3 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. A data model provides a way to describe the design of a database at the physical, logical, and view levels.
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. Chapters 2 through 8 cover the relational model in detail.

- **Entity-Relationship Model.** The entity-relationship (E-R) data model uses a collection of basic objects, called **entities**, and **relationships** among these objects. An entity 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, and Chapter 7 explores it in detail.

- **Object-Based Data Model.** Object-oriented programming (especially in Java, C++, or C#) has become the dominant software-development methodology. This led to the development of an object-oriented data model that can be seen as extending the E-R model with notions of encapsulation, methods (functions), and object identity. The object-relational data model combines features of the object-oriented data model and relational data model. Chapter 22 examines the object-relational data model.

- **Semistructured Data Model.** The semistructured data model permits the specification 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. The **Extensible Markup Language (XML)** is widely used to represent semistructured data. Chapter 23 covers it.

Historically, the **network data model** and the **hierarchical data model** preceded the relational data model. These models were tied closely to the underlying implementation, and complicated the task of modeling data. As a result they are used little now, except in old database code that is still in service in some places. They are outlined online in Appendices D and E for interested readers.

### 1.4 Database Languages

A database system provides a **data-definition language** to specify the database schema and a **data-manipulation language** to express database queries and up-
dates. In practice, the data-definition and data-manipulation languages are not two separate languages; instead they simply form parts of a single database language, such as the widely used SQL language.

### 1.4.1 Data-Manipulation Language

A **data-manipulation language (DML)** is a language that enables users to access or manipulate 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:

- **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 specify *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 Chapters 3, 4, and 5. We also study some other query languages in Chapter 6.

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 emphasize 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 Chapters 12 and 13) translates DML queries into sequences of actions at the physical level of the database system.

### 1.4.2 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 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 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 department listed for each course must be one that actually exists. 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.

- **Assertions.** An assertion is any condition that the database must always satisfy. Domain constraints and referential-integrity constraints are special forms of assertions. However, there are many constraints that we cannot express by using only these special forms. For example, “Every department must have at least five courses offered every semester” must be expressed as an assertion. When an assertion is created, the system tests it for validity. If the assertion is valid, then any future modification to the database is allowed only if it does not cause that assertion to be violated.

- **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 differentiations are expressed in terms of authorization, the most common being: read authorization, 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 authorization, which allows deletion of data. We may assign the user all, none, or a combination of these types of authorization.
The DDL, just like any other programming language, gets as input some instructions (statements) and 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 only be accessed and updated by the database system itself (not a regular user). The database system consults the data dictionary before reading or modifying actual data.

1.5 Relational Databases

A relational database is based on the relational model and uses a collection of tables to represent both data and the relationships among those data. It also includes a DML and DDL. In Chapter 2 we present a gentle introduction to the fundamentals of the relational model. Most commercial relational database systems employ the SQL language, which we cover in great detail in Chapters 3, 4, and 5. In Chapter 6 we discuss other influential languages.

1.5.1 Tables

Each table has multiple columns and each column has a unique name. Figure 1.2 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.

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.

It is not hard to see how tables may be stored in files. For instance, a special character (such as a comma) may be used to delimit the different attributes of a record, and another special character (such as a new-line character) may be used to delimit records. The relational model hides such low-level implementation details from database developers and users.

We also note that it is possible to create schemas in the relational model 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 a particular budget (say that one for the Physics department) changes, that change must to be reflected in the records of all instructors.
1.5 Relational Databases

![Table: Instructor Table](image1)

(a) The *instructor* table

![Table: Department Table](image2)

(b) The *department* table

*Figure 1.2* A sample relational database.

associated with the Physics department. In Chapter 8, we shall study how to distinguish good schema designs from bad schema designs.

1.5.2 Data-Manipulation Language

The SQL query language is nonprocedural. A query takes as input several tables (possibly 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:

```sql
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. More specifically, 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 \textit{dept\textunderscore name}, is History. If the query is run on the table in Figure 1.2, the result will consist 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 following query finds the instructor ID and department name of all instructors associated with a department with budget of greater than $95,000.

\begin{verbatim}
select instructor.ID, department.dept_name
from instructor, department
where instructor.dept_name = department.dept_name and
department.budget > 95000;
\end{verbatim}

If the above query were run on the tables in Figure 1.2, the system would find that there are two departments with budget of greater than $95,000—Computer Science and Finance; there are five instructors in these departments. Thus, the result will consist of a table with two columns (ID, dept\_name) and five rows: (12121, Finance), (45565, Computer Science), (10101, Computer Science), (83821, Computer Science), and (76543, Finance).

1.5.3 Data-Definition Language

SQL provides a rich DDL that allows one to define tables, integrity constraints, assertions, etc.

For instance, the following SQL DDL statement defines the \textit{department} table:

\begin{verbatim}
create table department
  (dept_name      char (20),
   building      char (15),
   budget        numeric (12,2));
\end{verbatim}

Execution of the above DDL statement creates the \textit{department} table with three columns: \textit{dept\_name}, \textit{building}, and \textit{budget}, each of which has a specific data type associated with it. We discuss data types in more detail in Chapter 3. In addition, the DDL statement updates the data dictionary, which contains metadata (see Section 1.4.2). The schema of a table is an example of metadata.

1.5.4 Database Access from Application Programs

SQL is 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 actions 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++, or Java, 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, etc. To access the database, DML statements need to be executed from the host language. There are two ways to do this:

- By providing 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 Connectivity (ODBC) standard for use with the C language is a commonly used application program interface standard. The Java Database Connectivity (JDBC) standard provides corresponding features to the Java language.

- By extending the host language syntax to embed DML calls within the host language program. Usually, a special character prefaces DML calls, and a preprocessor, called the DML precompiler, converts the DML statements to normal procedure calls in the host language.

### 1.6 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 initially on the writing of database queries and the design of database schemas. Chapter 9 discusses the overall process of application design.

#### 1.6.1 Design Process

A high-level data model provides the database designer with a conceptual framework 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 chosen 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 (Section 1.6.3); 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 (Section 1.6.4).

A fully developed conceptual schema indicates the functional requirements of the enterprise. In a specification of functional requirements, users describe the kinds of operations (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 designer 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 10.

1.6.2 Database Design for a University Organization

To illustrate the design process, let us examine how a database for a university could be designed. The initial specification of user requirements may be based on interviews with the database users, and on the designer’s own analysis of the organization. The description that arises from this design phase serves as the basis for specifying the conceptual structure of the database. Here are the major characteristics of the university.

- The university is organized into departments. Each department is identified by a unique name (dept\_name), is located in a particular building, and has a budget.
- Each department has a list of courses it offers. Each course has associated with it a course\_id, title, dept\_name, and credits, and may also have have associated prerequisites.
- Instructors are identified by their unique ID. Each instructor has name, associated department (dept\_name), and salary.
- Students are identified by their unique ID. Each student has a name, an associated major department (dept\_name), and tot\_cred (total credit hours the student earned thus far).
The university maintains a list of classrooms, specifying the name of the building, room number, and room capacity.

The university maintains a list of all classes (sections) taught. Each section is identified by a course id, sec id, year, and semester, and has associated with it a semester, year, building, room number, and time slot id (the time slot when the class meets).

The department has a list of teaching assignments specifying, for each instructor, the sections the instructor is teaching.

The university has a list of all student course registrations, specifying, for each student, the courses and the associated sections that the student has taken (registered for).

A real university database would be much more complex than the preceding design. However we use this simplified model to help you understand conceptual ideas without getting lost in details of a complex design.

1.6.3 The Entity-Relationship Model

The entity-relationship (E-R) data model uses a collection of basic objects, called entities, and relationships among these objects. An entity is a “thing” or “object” in the real world that is distinguishable from other objects. For example, each person is an entity, and bank accounts can be considered as entities.

Entities are described in a database by a set of attributes. For example, the attributes dept name, building, and budget may describe one particular department in a university, and they form attributes of the department entity set. Similarly, attributes ID, name, and salary may describe an instructor entity.\(^2\)

The extra attribute ID is used to identify an instructor uniquely (since it may be possible to have two instructors with the same name and the same salary). A unique instructor identifier must be assigned to each instructor. In the United States, many organizations use the social-security number of a person (a unique number the U.S. government assigns to every person in the United States) as a unique identifier.

A relationship is an association among several entities. For example, a member relationship associates an instructor with her department. The set of all entities of the same type and the set of all relationships of the same type are termed an entity set and relationship set, respectively.

The overall logical structure (schema) of a database can be expressed graphically by an entity-relationship (E-R) diagram. There are several ways in which to draw these diagrams. One of the most popular is to use the Unified Modeling Language (UML). In the notation we use, which is based on UML, an E-R diagram is represented as follows:

\(^2\)The astute reader will notice that we dropped the attribute dept name from the set of attributes describing the instructor entity set; this is not an error. In Chapter 7 we shall provide a detailed explanation of why this is the case.
• Entity sets are represented by a rectangular box with the entity set name in the header and the attributes listed below it.

• Relationship sets are represented by a diamond connecting a pair of related entity sets. The name of the relationship is placed inside the diamond.

As an illustration, consider part of a university database consisting of instructors and the departments with which they are associated. Figure 1.3 shows the corresponding E-R diagram. The E-R diagram indicates that there are two entity sets, *instructor* and *department*, with attributes as outlined earlier. The diagram also shows a relationship *member* between *instructor* and *department*.

In addition to entities and relationships, the E-R model represents certain constraints to which the contents of a database must conform. One important constraint is **mapping cardinalities**, which express the number of entities to which another entity can be associated via a relationship set. For example, if each instructor must be associated with only a single department, the E-R model can express that constraint.

The entity-relationship model is widely used in database design, and Chapter 7 explores it in detail.

### 1.6.4 Normalization

Another method for designing a relational database is to use a process commonly known as normalization. The goal is to generate a set of relation schemas that allows us to store information without unnecessary redundancy, yet also allows us to retrieve information easily. The approach is to design schemas that are in an appropriate **normal form**. To determine whether a relation schema is in one of the desirable normal forms, we need additional information about the real-world enterprise that we are modeling with the database. The most common approach is to use **functional dependencies**, which we cover in Section 8.4.

To understand the need for normalization, let us look at what can go wrong in a bad database design. Among the undesirable properties that a bad design may have are:

• Repetition of information

• Inability to represent certain information
We shall discuss these problems with the help of a modified database design for our university example.

Suppose that instead of having the two separate tables instructor and department, we have a single table, faculty, that combines the information from the two tables (as shown in Figure 1.4). Notice that there are two rows in faculty that contain repeated information about the History department, specifically, that department’s building and budget. The repetition of information in our alternative design is undesirable. Repeating information wastes space. Furthermore, it complicates updating the database. Suppose that we wish to change the budget amount of the History department from $50,000 to $46,800. This change must be reflected in the two rows; contrast this with the original design, where this requires an update to only a single row. Thus, updates are more costly under the alternative design than under the original design. When we perform the update in the alternative database, we must ensure that every tuple pertaining to the History department is updated, or else our database will show two different budget values for the History department.

Now, let us shift our attention to the issue of “inability to represent certain information.” Suppose we are creating a new department in the university. In the alternative design above, we cannot represent directly the information concerning a department (dept_name, building, budget) unless that department has at least one instructor at the university. This is because rows in the faculty table require values for ID, name, and salary. This means that we cannot record information about the newly created department until the first instructor is hired for the new department.

One solution to this problem is to introduce null values. The null value indicates that the value does not exist (or is not known). An unknown value may be either missing (the value does exist, but we do not have that information) or not known (we do not know whether or not the value actually exists). As we
shall see later, null values are difficult to handle, and it is preferable not to resort to them. If we are not willing to deal with null values, then we can create a particular item of department information only when the department has at least one instructor associated with the department. Furthermore, we would have to delete this information when the last instructor in the department departs. Clearly, this situation is undesirable, since, under our original database design, the department information would be available regardless of whether or not there is an instructor associated with the department, and without resorting to null values.

An extensive theory of normalization has been developed that helps formally define what database designs are undesirable, and how to obtain desirable designs. Chapter 8 covers relational-database design, including normalization.

1.7 Data Storage and Querying

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 and the query processor components.

The storage manager is important because databases typically require a large amount of storage space. Corporate databases range in size from hundreds of gigabytes to, for the largest databases, terabytes of data. A gigabyte is approximately 1000 megabytes (actually 1024) (1 billion bytes), and a terabyte is 1 million megabytes (1 trillion bytes). Since the main memory of computers cannot store this much information, 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 processing 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.

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 understanding 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.

1.7.1 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 storing, 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 (correct) state despite system failures, and that concurrent transaction executions proceed without conflicting.
- **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. Hashing is an alternative to indexing that is faster in some but not all cases.

We discuss storage media, file structures, and buffer management in Chapter 10. Methods of accessing data efficiently via indexing or hashing are discussed in Chapter 11.

### 1.7.2 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 evaluation 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 evaluation 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 alternatives.

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

Query evaluation is covered in Chapter 12, while the methods by which the query optimizer chooses from among the possible evaluation strategies are discussed in Chapter 13.

### 1.8 Transaction Management

Often, several operations on the database form a single logical unit of work. An example is a funds transfer, as in Section 1.2, in which one department account (say A) is debited and another department account (say 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 preserve the consistency of the database. That is, the value of the sum of the balances of A and B must be preserved. This correctness requirement 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 execution of a transaction, it may be necessary temporarily to allow inconsistency, since 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 define properly the various transactions, so that each preserves the consistency of the database. For example, the transaction to transfer funds from the account of department A to the account of department 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, because 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 system must therefore perform failure recovery, that is, 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 consistency 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 interaction 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 14. The management of concurrent transactions is covered in Chapter 15. Chapter 16 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 workflows. These broader applications of the transaction concept are discussed in Chapter 26.

1.9 Database Architecture

We are now in a position to provide a single picture (Figure 1.5) of the various components of a database system and the connections among them.

The architecture of a database system is greatly influenced by the underlying computer system on which the database system runs. Database systems can be centralized, or client-server, where one server machine executes work on behalf of multiple client machines. Database systems can also be designed to exploit parallel computer architectures. Distributed databases span multiple geographically separated machines.

In Chapter 17 we cover the general structure of modern computer systems. Chapter 18 describes how various actions of a database, in particular query processing, can be implemented to exploit parallel processing. Chapter 19 presents a number of issues that arise in 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. Distributed query processing and directory systems are also described in this chapter.

Most users of a database system today are not present at the site of the database system, but connect to it through a network. We can therefore differentiate between client machines, on which remote database users work, and server machines, on which the database system runs.