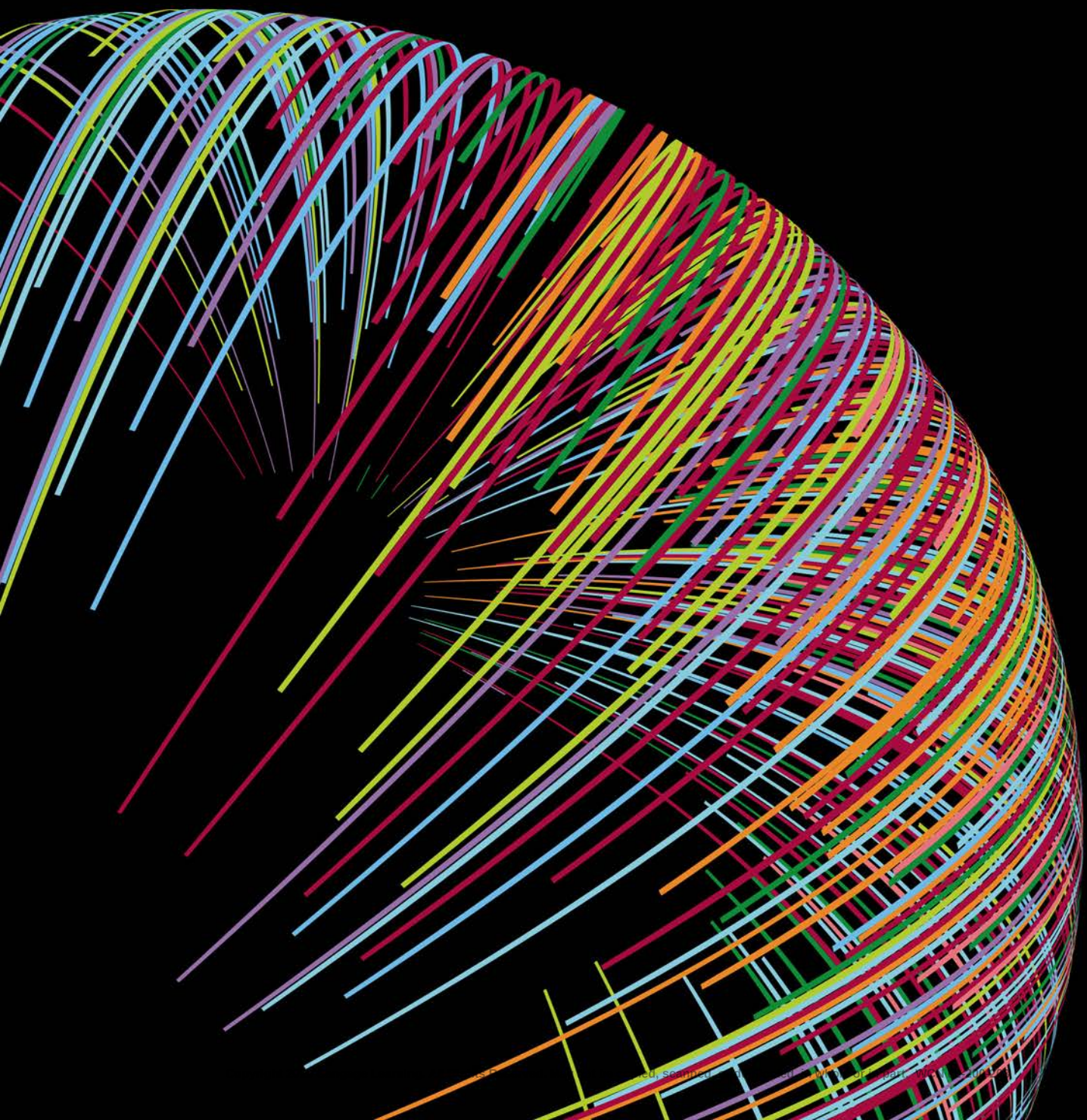
DATABASE SYSTEMS

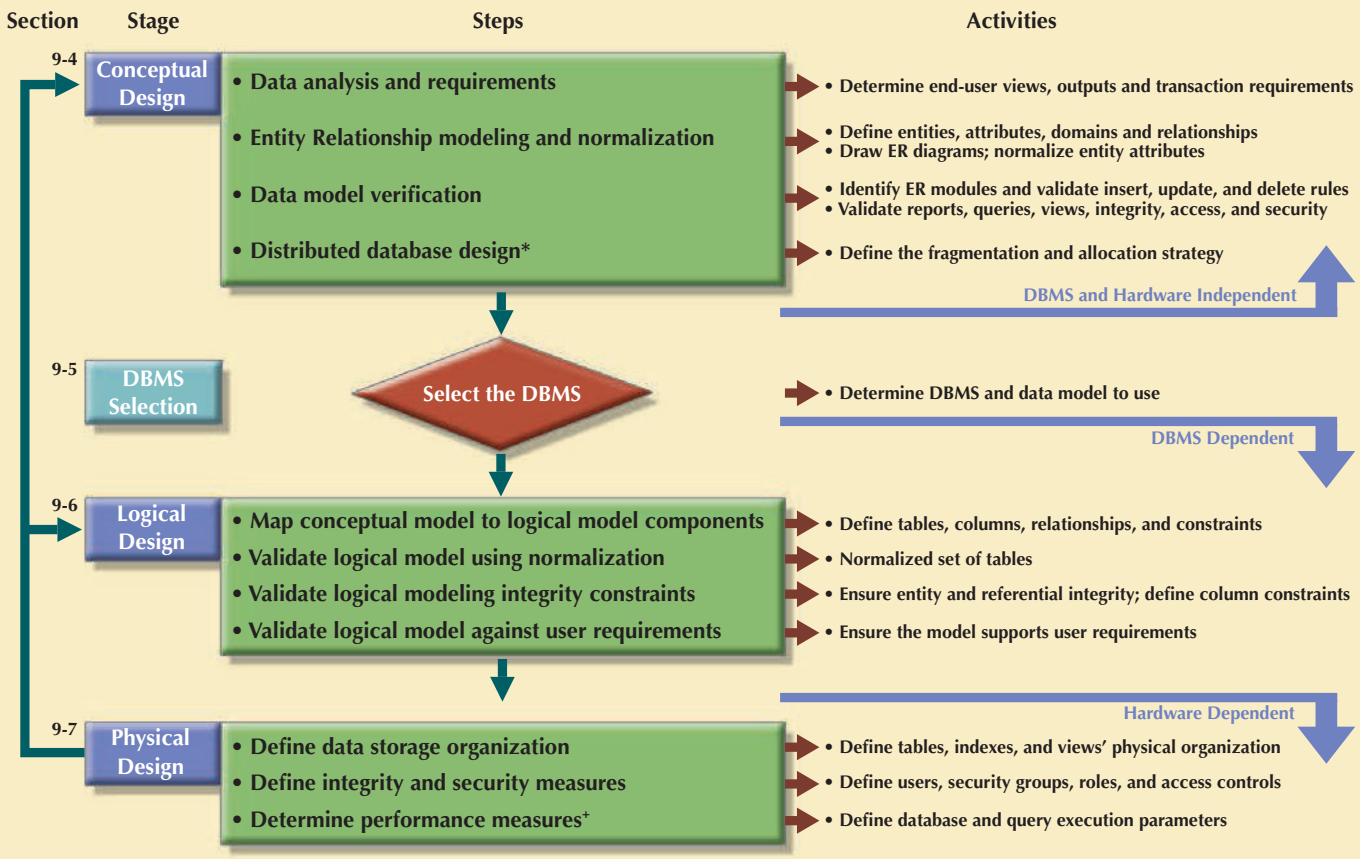
DESIGN, IMPLEMENTATION, & MANAGEMENT

CARLOS CORONEL • STEVEN MORRIS

13TH EDITION



Database Design Process



* See Chapter 12, Distributed Database Management Systems

+ See Chapter 11, Database Performance Tuning and Query Optimization

Data Modeling Checklist

BUSINESS RULES

- ☑ Properly document and verify all business rules with the end users.
- ☑ Ensure that all business rules are written precisely, clearly, and simply. The business rules must help identify entities, attributes, relationships, and constraints.
- ☑ Identify the source of all business rules, and ensure that each business rule is justified, dated, and signed off by an approving authority.

DATA MODELING

Naming Conventions: All names should be limited in length (database-dependent size).

Entity Names:

- ☑ Should be nouns that are familiar to business and should be short and meaningful
- ☑ Should document abbreviations, synonyms, and aliases for each entity
- ☑ Should be unique within the model
- ☑ For composite entities, may include a combination of abbreviated names of the entities linked through the composite entity

Attribute Names:

- ☑ Should be unique within the entity
- ☑ Should use the entity abbreviation as a prefix
- ☑ Should be descriptive of the characteristic
- ☑ Should use suffixes such as _ID, _NUM, or _CODE for the PK attribute
- ☑ Should not be a reserved word
- ☑ Should not contain spaces or special characters such as @, !, or &

Relationship Names:

- ☑ Should be active or passive verbs that clearly indicate the nature of the relationship

Entities:

- ☑ Each entity should represent a single subject.
- ☑ Each entity should represent a set of distinguishable entity instances.
- ☑ All entities should be in 3NF or higher. Any entities below 3NF should be justified.
- ☑ The granularity of the entity instance should be clearly defined.
- ☑ The PK is clearly defined and supports the selected data granularity.

Attributes:

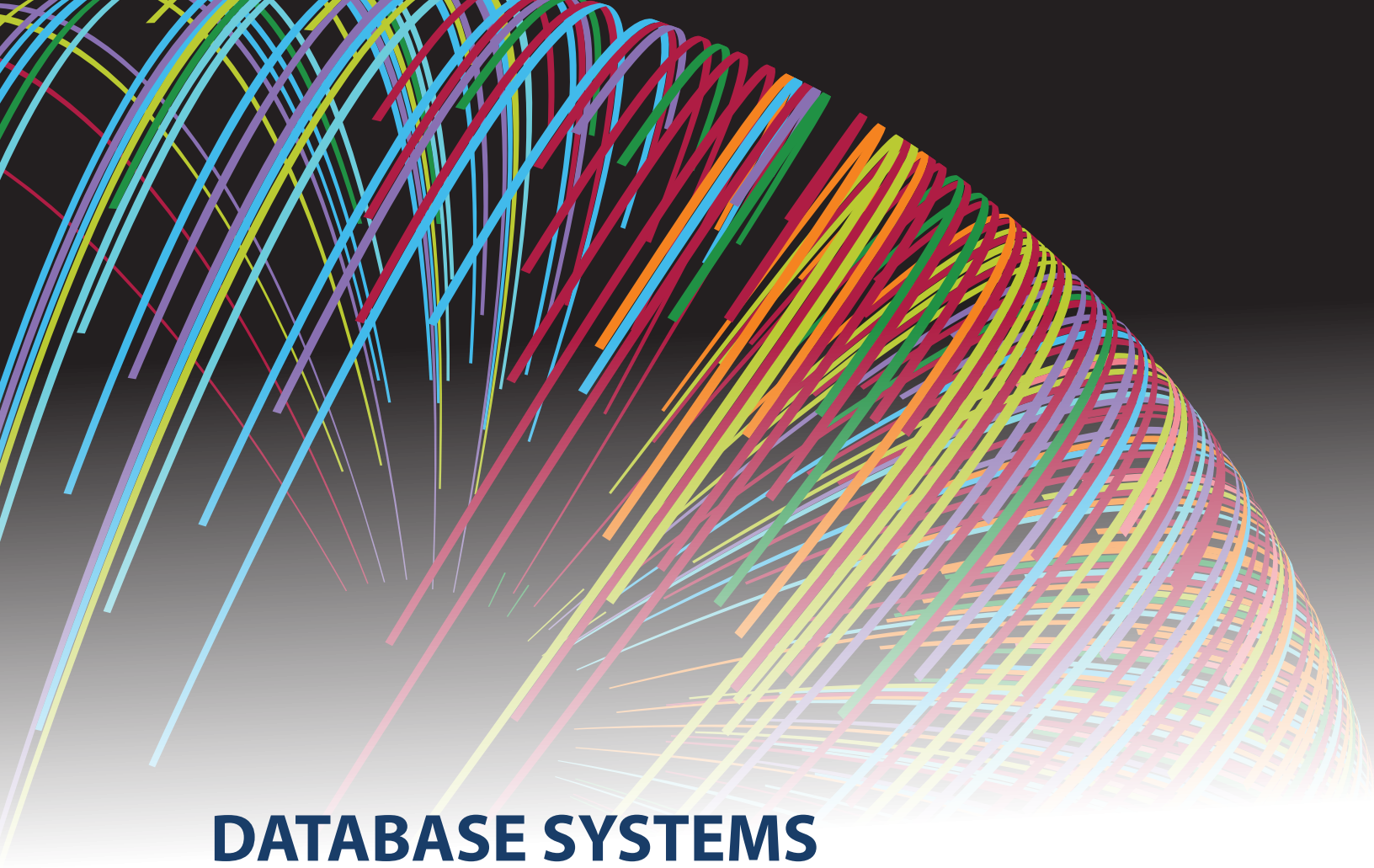
- ☑ Should be simple and single-valued (atomic data)
- ☑ Should document default values, constraints, synonyms, and aliases
- ☑ Derived attributes should be clearly identified and include source(s)
- ☑ Should not be redundant unless they are justified for transaction accuracy, performance, or maintaining a history
- ☑ Nonkey attributes must be fully dependent on the PK attribute

Relationships:

- ☑ Should clearly identify relationship participants
- ☑ Should clearly define participation, connectivity, and document cardinality

ER Model:

- ☑ Should be validated against expected processes: inserts, updates, and deletes
- ☑ Should evaluate where, when, and how to maintain a history
- ☑ Should not contain redundant relationships except as required (see Attributes)
- ☑ Should minimize data redundancy to ensure single-place updates
- ☑ Should conform to the minimal data rule: "All that is needed is there and all that is there is needed."



DATABASE SYSTEMS

Design, Implementation, and Management

13e

Carlos Coronel | Steven Morris

CENGAGE

Australia • Brazil • Mexico • Singapore • United Kingdom • United States

CENGAGE

***Database Systems: Design, Implementation, and Management,* 13th Edition**
Carlos Coronel and Steven Morris

SVP, GM Skills: Jonathan Lau

Product Director: Lauren Murphy

Product Team Manager: Kirstin McNary

Associate Product Manager: Kate Mason

Executive Director of Development: Marah Bellegarde

Senior Content Development Manager: Leigh Hefferon

Content Developer: Maria Garguilo

Product Assistant: Jake Toth

VP, Marketing for Science, Technology, & Math: Jason Sakos

Marketing Director: Michele McTighe

Marketing Manager: Stephanie Albracht

Production Director: Patty Stephan

Content Project Manager: Michele Stulga

Art Director: Diana Graham

Cover Designer: Roycroft Design (roycroftdesign.com)

Cover Image: iStock.com/liuzishan

Library of Congress Control Number: 2015955694

Student Edition ISBN: 978-1-337-62790-0
Loose Leaf Edition ISBN: 978-1-337-68882-6

Cengage
20 Channel Center Street
Boston, MA 02210
USA

Cengage Learning is a leading provider of customized learning solutions with employees residing in nearly 40 different countries and sales in more than 125 countries around the world. Find your local representative at **www.cengage.com.**

Cengage Learning products are represented in Canada by Nelson Education, Ltd.

To learn more about Cengage, visit **www.cengage.com**

Purchase any of our products at your local college store or at our preferred online store **www.cengagebrain.com.**

Printed in the United States of America
Print Number: 01 Print Year: 2017

Dedication

To the treasures in my life: To Victoria, for 28 wonderful years. Thank you for your unending support and for being my angel, my sweetie, and, most importantly, my best friend. To Carlos Anthony, who has become a remarkable man, pride of his father, and husband to our beautiful, sweet, and smart daughter-in-law, Jered. Thank you for your words of wisdom, hard-working attitude, and for giving us reasons to be happy. You are still young; your best times are still to come. To Gabriela Victoria, who is the image of brilliance, beauty, and faithfulness. The way you give your time and talents in the service of others is an inspiration to all of us. Thank you for being my sunshine on cloudy days. Your future is bright and endless. To Christian Javier, who is smarter than of all of us. Thank you for being the youthful reminder of life's simple beauties. Keep challenging yourself to new highs and keep working hard to achieve your dreams. To my parents, Sarah and Carlos, thank you for your sacrifice and example. To all of you, you are all my inspiration. "TQTATA."

Carlos Coronel

To Pamela, from high school sweetheart through nearly 30 years of marriage, you are the beautiful love of my life who has supported, encouraged, and inspired me. More than anyone else, you are responsible for whatever successes I have achieved. To my son, Alexander Logan, your depth of character is without measure. You are my pride and joy. To my daughter, Lauren Elizabeth, your beauty and intensity take my breath away. You are my heart and soul. Thank you all for the sacrifices you have made that enabled me to pursue this dream. I love you so much more than I can express. To my mother, Florence Maryann, and to the memory of my father, Alton Lamar, together they instilled in me the desire to learn and the passion to achieve. To my mother-in-law, Connie Duke, and to the memory of my father-in-law, Wayne Duke, they taught me to find joy in all things. To all of you, with all my love, I dedicate this book.

Steven Morris

For Peter

To longtime colleague and friend, Peter Rob: Your drive and dedication to your students started this book. Your depth of knowledge, attention to detail, and pursuit of excellence made it succeed. Your patience and guidance continue to light our path. It is our sincere hope that, as we move forward, we can continue to live up to your standard. Enjoy your retirement, my friend; you have surely earned it.

Carlos Coronel and Steven Morris

Brief Contents

Preface, xv

Text Features, xx

Additional Features, xxii

Acknowledgments, xxiv

Part 1: Database Concepts 1

1. Database Systems, 2
2. Data Models, 34

Part 2: Design Concepts 67

3. The Relational Database Model, 68
4. Entity Relationship (ER) Modeling, 113
5. Advanced Data Modeling, 167
6. Normalization of Database Tables, 199

Part 3: Advanced Design and Implementation 243

7. Introduction to Structured Query Language (SQL), 244
8. Advanced SQL, 359
9. Database Design, 439

Part 4: Advanced Database Concepts 481

10. Transaction Management and Concurrency Control, 482
11. Database Performance Tuning and Query Optimization, 515
12. Distributed Database Management Systems, 553
13. Business Intelligence and Data Warehouses, 589
14. Big Data and NoSQL, 657

Part 5: Databases and the Internet 691

15. Database Connectivity and Web Technologies, 692

Part 6: Database Administration 733

16. Database Administration and Security, 734

Glossary, 782

Index, 793

The following appendices are included on the Instructor and Student Companion Sites at ***www.cengagebrain.com***.

Appendix A1: Designing Databases with Visio Professional 2010: A Tutorial

Appendix A2: Designing Databases with Visio 2013: A Tutorial

Appendix B: The University Lab: Conceptual Design

Appendix C: The University Lab: Conceptual Design Verification, Logical Design, and Implementation

Appendix D: Converting an ER Model into a Database Structure

Appendix E: Comparison of ER Modeling Notations

Appendix F: Client/Server Systems

Appendix G: Object-Oriented Databases

Appendix H: Unified Modeling Language (UML)

Appendix I: Databases in Electronic Commerce

Appendix J: Web Database Development with ColdFusion

Appendix K: The Hierarchical Database Model

Appendix L: The Network Database Model

Appendix M: MS Access Tutorial

Appendix N: Creating a New Database Using Oracle 12c

Appendix O: Data Warehouse Implementation Factors

Appendix P: Working with MongoDB

Appendix Q: Working with Neo4j

Contents

Preface, xv

Text Features, xx

Additional Features, xxii

Acknowledgments, xxiv

Part 1: Database Concepts 1

Chapter 1: Database Systems 2

1-1 Why Databases? **3**
1-2 Data versus Information **4**
1-3 Introducing the Database **6**
- **1-3a** Role and Advantages of the DBMS 7
- **1-3b** Types of Databases 9

1-4 Why Database Design Is Important **12**
1-5 Evolution of File System Data Processing **15**
- **1-5a** Manual File Systems 15
- **1-5b** Computerized File Systems 15
- **1-5c** File System Redux: Modern End-User Productivity Tools 18

1-6 Problems with File System Data Processing **18**
- **1-6a** Structural and Data Dependence 19
- **1-6b** Data Redundancy 20
- **1-6c** Data Anomalies 21

1-7 Database Systems **21**
- **1-7a** The Database System Environment 22
- **1-7b** DBMS Functions 24
- **1-7c** Managing the Database System: A Shift in Focus 27

1-8 Preparing for Your Database Professional Career **28**

Summary 30 • Key Terms 31 • Review Questions 31 • Problems 32

Chapter 2: Data Models 34

2-1 Data Modeling and Data Models **35**
2-2 The Importance of Data Models **36**
2-3 Data Model Basic Building Blocks **36**
2-4 Business Rules **38**
- **2-4a** Discovering Business Rules 38
- **2-4b** Translating Business Rules into Data Model Components 39
- **2-4c** Naming Conventions 40

2-5 The Evolution of Data Models **40**
- **2-5a** Hierarchical and Network Models 40
- **2-5b** The Relational Model 42
- **2-5c** The Entity Relationship Model 44
- **2-5d** The Object-Oriented Model 47
- **2-5e** Object/Relational and XML 48
- **2-5f** Emerging Data Models: Big Data and NoSQL 49
- **2-5g** Data Models: A Summary 53

2-6 Degrees of Data Abstraction **54**
- **2-6a** The External Model 57
- **2-6b** The Conceptual Model 58
- **2-6c** The Internal Model 59
- **2-6d** The Physical Model 60

Summary 61 • Key Terms 62 • Review Questions 62 • Problems 63

Part 2: Design Concepts 67

Chapter 3: The Relational Database Model 68

3-1 A Logical View of Data **69**
- **3-1a** Tables and Their Characteristics 69

3-2 Keys **72**
- **3-2a** Dependencies 72
- **3-2b** Types of Keys 73

3-3 Integrity Rules **76**

3-4 Relational Algebra **78**
- **3-4a** Formal Definitions and Terminology 78
- **3-4b** Relational Set Operators 79

3-5 The Data Dictionary and the System Catalog **87**

3-6 Relationships within the Relational Database **89**
- **3-6a** The 1:M Relationship 89
- **3-6b** The 1:1 Relationship 91
- **3-6c** The M:N Relationship 93

3-7 Data Redundancy Revisited **97**

3-8 Indexes **99**

3-9 Codd's Relational Database Rules **100**

Summary 102 • Key Terms 103 • Review Questions 103 • Problems 106

Chapter 4: Entity Relationship (ER) Modeling 113

4-1 The Entity Relationship Model **114**
- **4-1a** Entities 114
- **4-1b** Attributes 114
- **4-1c** Relationships 120
- **4-1d** Connectivity and Cardinality 121
- **4-1e** Existence Dependence 122
- **4-1f** Relationship Strength 123
- **4-1g** Weak Entities 125
- **4-1h** Relationship Participation 127
- **4-1i** Relationship Degree 131
- **4-1j** Recursive Relationships 133
- **4-1k** Associative (Composite) Entities 136

4-2 Developing an ER Diagram **138**

4-3 Database Design Challenges: Conflicting Goals **146**

Summary 150 • Key Terms 151 • Review Questions 151 • Problems 154 • Cases 159

Chapter 5: Advanced Data Modeling 167

5-1 The Extended Entity Relationship Model **168**
- **5-1a** Entity Supertypes and Subtypes 168
- **5-1b** Specialization Hierarchy 169
- **5-1c** Inheritance 170
- **5-1d** Subtype Discriminator 172
- **5-1e** Disjoint and Overlapping Constraints 172
- **5-1f** Completeness Constraint 174
- **5-1g** Specialization and Generalization 175

5-2 Entity Clustering **175**

5-3 Entity Integrity: Selecting Primary Keys **176**
- **5-3a** Natural Keys and Primary Keys 177
- **5-3b** Primary Key Guidelines 177
- **5-3c** When to Use Composite Primary Keys 177
- **5-3d** When to Use Surrogate Primary Keys 179

5-4 Design Cases: Learning Flexible Database Design **180**
- **5-4a** Design Case 1: Implementing 1:1 Relationships 181
- **5-4b** Design Case 2: Maintaining History of Time-Variant Data 182
- **5-4c** Design Case 3: Fan Traps 185
- **5-4d** Design Case 4: Redundant Relationships 186

Summary 187 • Key Terms 187 • Review Questions 188 • Problems 189 • Cases 190

Chapter 6: Normalization of Database Tables 199

6-1 Database Tables and Normalization **200**
6-2 The Need for Normalization **200**
6-3 The Normalization Process **203**
6-3a Conversion to First Normal Form (1NF) 205
6-3b Conversion to Second Normal Form (2NF) 209
6-3c Conversion to Third Normal Form (3NF) 211
6-4 Improving the Design **213**
6-5 Surrogate Key Considerations **217**
6-6 Higher-Level Normal Forms **218**
6-6a The Boyce-Codd Normal Form 219
6-6b Fourth Normal Form (4NF) 222
6-7 Normalization and Database Design **224**
6-8 Denormalization **227**
6-9 Data-Modeling Checklist **230**

Summary 232 • Key Terms 233 • Review Questions 233 • Problems 235

Part 3: Advanced Design and Implementation 243

Chapter 7: Introduction to Structured Query Language (SQL) 244

7-1 Introduction to SQL **245**
7-1a Data Types 245
7-1b SQL Queries 247
7-1c The Database Model 248
7-2 Basic SELECT Queries **249**
7-3 SELECT Statement Options **250**
7-3a Using Column Aliases 251
7-3b Using Computed Columns 253
7-3c Arithmetic Operators: The Rule of Precedence 254
7-3d Date Arithmetic 255
7-3e Listing Unique Values 255
7-4 FROM Clause Options **256**
7-4a Natural Join 257
7-4b JOIN USING Syntax 259
7-4c JOIN ON Syntax 260
7-4d Common Attribute Names 261
7-4e Outer Joins 261
7-4f Cross Join 264
7-4g Joining Tables with an Alias 264
7-4h Recursive Joins 265
7-5 ORDER BY Clause Options **266**
7-6 WHERE Clause Options **269**
7-6a Selecting Rows with Conditional Restrictions 269
7-6b Using Comparison Operators on Character Attributes 271
7-6c Using Comparison Operators on Dates 272
7-6d Logical Operators: AND, OR, and NOT 273
7-6e Old-Style Joins 275
7-6f Special Operators 276
7-7 Aggregate Processing **281**
7-7a Aggregate Functions 281
7-7b Grouping Data 285
7-7c HAVING Clause 288
7-8 Subqueries **290**
7-8a WHERE Subqueries 292
7-8b IN Subqueries 293
7-8c HAVING Subqueries 294
7-8d Multirow Subquery Operators: ALL and ANY 294
7-8e FROM Subqueries 295
7-8f Attribute List Subqueries 296
7-8g Correlated Subqueries 298
7-9 SQL Functions **302**
7-9a Date and Time Functions 302
7-9b Numeric Functions 306

7-9c String Functions 307
7-9d Conversion Functions 309
7-10 Relational Set Operators **311**
7-10a UNION 311
7-10b UNION ALL 313
7-10c INTERSECT 314
7-10d EXCEPT (MINUS) 315
7-10e Syntax Alternatives 316
7-11 Crafting SELECT Queries **317**
7-11a Know Your Data 317
7-11b Know the Problem 317
7-11c Build One Clause at a Time 318
Summary 319 • Key Terms 321 • Review Questions 321 • Problems 323

Chapter 8: Advanced SQL **359**

8-1 Data Definition Commands **360**
8-1a Starting Database Model 360
8-1b Creating the Database 361
8-1c The Database Schema 362
8-1d Data Types 362
8-2 Creating Table Structures **366**
8-2a CREATE TABLE command 366
8-2b SQL Constraints 370
8-2c Create a Table with a SELECT Statement 373
8-2d SQL Indexes 374
8-3 Altering Table Structures **375**
8-3a Changing a Column's Data Type 376
8-3b Changing a Column's Data Characteristics 376
8-3c Adding a Column 377
8-3d Adding Primary Key, Foreign Key, and Check Constraints 377
8-3e Dropping a Column 378
8-3f Deleting a Table from the Database 378
8-4 Data Manipulation Commands **379**
8-4a Adding Table Rows 379
8-4b Inserting Table Rows with a SELECT Subquery 381
8-4c Saving Table Changes 382
8-4d Updating Table Rows 383
8-4e Deleting Table Rows 385
8-4f Restoring Table Contents 386
8-5 Virtual Tables: Creating a View **387**
8-5a Updatable Views 388
8-6 Sequences **391**
8-7 Procedural SQL **396**
8-7a Triggers 401
8-7b Stored Procedures 411
8-7c PL/SQL Processing with Cursors 416
8-7d PL/SQL Stored Functions 418
8-8 Embedded SQL **419**
Summary 423 • Key Terms 425 • Review Questions 425 • Problems 426 • Cases 433

Chapter 9: Database Design **439**

9-1 The Information System **440**
9-2 The Systems Development Life Cycle **442**
9-2a Planning 442
9-2b Analysis 443
9-2c Detailed Systems Design 444
9-2d Implementation 444
9-2e Maintenance 445
9-3 The Database Life Cycle **445**
9-3a The Database Initial Study 445
9-3b Database Design 450
9-3c Implementation and Loading 451
9-3d Testing and Evaluation 454

9-3e Operation 456
9-3f Maintenance and Evolution 457
9-4 Conceptual Design **457**
9-4a Data Analysis and Requirements 459
9-4b Entity Relationship Modeling and Normalization 461
9-4c Data Model Verification 464
9-4d Distributed Database Design 467
9-5 DBMS Software Selection **467**
9-6 Logical Design **468**
9-6a Map the Conceptual Model to the Logical Model 468
9-6b Validate the Logical Model Using Normalization 470
9-6c Validate Logical Model Integrity Constraints 470
9-6d Validate the Logical Model against User Requirements 471
9-7 Physical Design **471**
9-7a Define Data Storage Organization 472
9-7b Define Integrity and Security Measures 472
9-7c Determine Performance Measures 473
9-8 Database Design Strategies **473**
9-9 Centralized versus Decentralized Design **474**
Summary 477 • Key Terms 477 • Review Questions 477 • Problems 478

Part 4: Advanced Database Concepts 481

Chapter 10: Transaction Management and Concurrency Control **482**

10-1 What Is a Transaction? **483**
10-1a Evaluating Transaction Results 484
10-1b Transaction Properties 487
10-1c Transaction Management with SQL 488
10-1d The Transaction Log 489
10-2 Concurrency Control **490**
10-2a Lost Updates 490
10-2b Uncommitted Data 491
10-2c Inconsistent Retrievals 492
10-2d The Scheduler 493
10-3 Concurrency Control with Locking Methods **495**
10-3a Lock Granularity 496
10-3b Lock Types 498
10-3c Two-Phase Locking to Ensure Serializability 500
10-3d Deadlocks 500
10-4 Concurrency Control with Time Stamping Methods **502**
10-4a Wait/Die and Wound/Wait Schemes 502
10-5 Concurrency Control with Optimistic Methods **503**
10-6 ANSI Levels of Transaction Isolation **504**
10-7 Database Recovery Management **506**
10-7a Transaction Recovery 506
Summary 510 • Key Terms 511 • Review Questions 511 • Problems 512

Chapter 11: Database Performance Tuning and Query Optimization **515**

11-1 Database Performance-Tuning Concepts **516**
11-1a Performance Tuning: Client and Server 517
11-1b DBMS Architecture 518
11-1c Database Query Optimization Modes 520
11-1d Database Statistics 521
11-2 Query Processing **522**
11-2a SQL Parsing Phase 523
11-2b SQL Execution Phase 524
11-2c SQL Fetching Phase 525
11-2d Query Processing Bottlenecks 525
11-3 Indexes and Query Optimization **526**
11-4 Optimizer Choices **528**
11-4a Using Hints to Affect Optimizer Choices 530

11-5 SQL Performance Tuning **531**
11-5a Index Selectivity 531
11-5b Conditional Expressions 533
11-6 Query Formulation **534**
11-7 DBMS Performance Tuning **536**
11-8 Query Optimization Example **538**
Summary 546 • Key Terms 547 • Review Questions 547 • Problems 548

Chapter 12: Distributed Database Management Systems **553**

12-1 The Evolution of Distributed Database Management Systems **554**
12-2 DDBMS Advantages and Disadvantages **556**
12-3 Distributed Processing and Distributed Databases **556**
12-4 Characteristics of Distributed Database Management Systems **559**
12-5 DDBMS Components **560**
12-6 Levels of Data and Process Distribution **561**
12-6a Single-Site Processing, Single-Site Data 561
12-6b Multiple-Site Processing, Single-Site Data 562
12-6c Multiple-Site Processing, Multiple-Site Data 563
12-7 Distributed Database Transparency Features **564**
12-8 Distribution Transparency **565**
12-9 Transaction Transparency **568**
12-9a Distributed Requests and Distributed Transactions 568
12-9b Distributed Concurrency Control 571
12-9c Two-Phase Commit Protocol 571
12-10 Performance and Failure Transparency **573**
12-11 Distributed Database Design **575**
12-11a Data Fragmentation 575
12-11b Data Replication 578
12-11c Data Allocation 580
12-12 The CAP Theorem **581**
12-13 C. J. Date's 12 Commandments for Distributed Databases **583**
Summary 584 • Key Terms 585 • Review Questions 585 • Problems 586

Chapter 13: Business Intelligence and Data Warehouses **589**

13-1 The Need for Data Analysis **590**
13-2 Business Intelligence **590**
13-2a Business Intelligence Architecture 592
13-2b Business Intelligence Benefits 596
13-2c Business Intelligence Evolution 597
13-2d Business Intelligence Technology Trends 600
13-3 Decision Support Data **601**
13-3a Operational Data versus Decision Support Data 601
13-3b Decision Support Database Requirements 604
13-4 The Data Warehouse **606**
13-4a Data Marts 609
13-4b Twelve Rules That Define a Data Warehouse 609
13-5 Star Schemas **609**
13-5a Facts 610
13-5b Dimensions 610
13-5c Attributes 611
13-5d Attribute Hierarchies 613
13-5e Star Schema Representation 615
13-5f Performance-Improving Techniques for the Star Schema 616
13-6 Online Analytical Processing **620**
13-6a Multidimensional Data Analysis Techniques 620
13-6b Advanced Database Support 622
13-6c Easy-to-Use End-User Interfaces 622
13-6d OLAP Architecture 622
13-6e Relational OLAP 625
13-6f Multidimensional OLAP 627
13-6g Relational versus Multidimensional OLAP 627

13-7 Data Analytics **628**
13-7a Data Mining 629
13-7b Predictive Analytics 631
13-8 SQL Analytic Functions **632**
13-8a The ROLLUP Extension 633
13-8b The CUBE Extension 634
13-8c Materialized Views 636
13-9 Data Visualization **639**
13-9a The Need for Data Visualization 640
13-9b The Science of Data Visualization 642
13-9c Understanding the Data 644
Summary 645 • Key Terms 646 • Review Questions 647 • Problems 648

Chapter 14: Big Data and NoSQL **657**

14-1 Big Data 658
14-1a Volume 660
14-1b Velocity 661
14-1c Variety 662
14-1d Other Characteristics 663
14-2 Hadoop **664**
14-2a HDFS 665
14-2b MapReduce 667
14-2c Hadoop Ecosystem 669
14-3 NoSQL **672**
14-3a Key-Value Databases 673
14-3b Document Databases 674
14-3c Column-Oriented Databases 675
14-3d Graph Databases 677
14-3e Aggregate Awareness 679
14-4 NewSQL Databases **680**
14-5 Working with Document Databases Using MongoDB **680**
14-5a Importing Documents in MongoDB 682
14-5b Example of a MongoDB Query Using find() 683
14-6 Working with Graph Databases Using Neo4j **684**
14-6a Creating Nodes in Neo4j 685
14-6b Retrieving Node Data with MATCH and WHERE 686
14-6c Retrieving Relationship Data with MATCH and WHERE 686
Summary 688 • Key Terms 689 • Review Questions 690

Part 5: Databases and the Internet 691

Chapter 15: Database Connectivity and Web Technologies **692**

15-1 Database Connectivity **693**
15-1a Native SQL Connectivity 694
15-1b ODBC, DAO, and RDO 695
15-1c OLE-DB 697
15-1d ADO.NET 699
15-1e Java Database Connectivity (JDBC) 703
15-2 Database Internet Connectivity **704**
15-2a Web-to-Database Middleware: Server-Side Extensions 705
15-2b Web Server Interfaces 707
15-2c The Web Browser 708
15-2d Client-Side Extensions 709
15-2e Web Application Servers 710
15-2f Web Database Development 711
15-3 Extensible Markup Language (XML) **715**
15-3a Document Type Definitions (DTD) and XML Schemas 717
15-3b XML Presentation 719
15-3c XML Applications 721

15-4 Cloud Computing Services **722**
15-4a Cloud Implementation Types 725
15-4b Characteristics of Cloud Services 725
15-4c Types of Cloud Services 726
15-4d Cloud Services: Advantages and Disadvantages 727
15-4e SQL Data Services 729
Summary 730 • Key Terms 731 • Review Questions 731 • Problems 732

Part 6: Database Administration 733

Chapter 16: Database Administration and Security **734**

16-1 Data as a Corporate Asset **735**
16-2 The Need for a Database and Its Role in an Organization **736**
16-3 Introduction of a Database: Special Considerations **738**
16-4 The Evolution of Database Administration **739**
16-5 The Database Environment's Human Component **743**
16-5a The DBA's Managerial Role 745
16-5b The DBA's Technical Role 750
16-6 Security **757**
16-6a Security Policies 758
16-6b Security Vulnerabilities 758
16-6c Database Security 760
16-7 Database Administration Tools **761**
16-7a The Data Dictionary 762
16-7b Case Tools 764
16-8 Developing a Data Administration Strategy **767**
16-9 The DBA's Role in the Cloud **768**
16-10 The DBA at Work: Using Oracle for Database Administration **769**
16-10a Oracle Database Administration Tools 770
16-10b Ensuring That the RDBMS Starts Automatically 770
16-10c Creating Tablespaces and Datafiles 772
16-10d Managing Users and Establishing Security 774
16-10e Customizing the Database Initialization Parameters 776
Summary 777 • Key Terms 779 • Review Questions 779
Glossary 782
Index 793

The following appendices are included on the Instructor and Student Companion Sites at ***www.cengagebrain.com***.

Appendix A1: Designing Databases with Visio Professional 2010: A Tutorial
Appendix A2: Designing Databases with Visio 2013: A Tutorial
Appendix B: The University Lab: Conceptual Design
Appendix C: The University Lab: Conceptual Design Verification, Logical Design, and Implementation
Appendix D: Converting an ER Model into a Database Structure
Appendix E: Comparison of ER Modeling Notations
Appendix F: Client/Server Systems
Appendix G: Object-Oriented Databases
Appendix H: Unified Modeling Language (UML)
Appendix I: Databases in Electronic Commerce
Appendix J: Web Database Development with ColdFusion
Appendix K: The Hierarchical Database Model
Appendix L: The Network Database Model
Appendix M: MS Access Tutorial
Appendix N: Creating a New Database Using Oracle 12c
Appendix O: Data Warehouse Implementation Factors
Appendix P: Working with MongoDB
Appendix Q: Working with Neo4j

Preface

It is our great pleasure to present the thirteenth edition of *Database Systems*. We are grateful and humbled that so many of our colleagues around the world have chosen this text to support their classes. We wrote the first edition of this book because we wanted to explain the complexity of database systems in a language that was easy for students to understand. Over the years, we have maintained this emphasis on reaching out to students to explain complex concepts in a practical, approachable manner. This book has been successful through twelve editions because the authors, editors, and the publisher paid attention to the impact of technology and to adopters' questions and suggestions. We believe that this thirteenth edition successfully reflects the same attention to such factors.

In many respects, rewriting a book is more difficult than writing it the first time. If the book is successful, as this one is, a major concern is that the updates, inserts, and deletions will adversely affect writing style and continuity of coverage. The combination of superb reviewers and editors, plus a wealth of feedback from adopters and students of the previous editions, helped make this new edition the best yet.

Changes to the Thirteenth Edition

In this thirteenth edition, we have responded to the requests and suggestions of numerous adopters. We have substantially reorganized the SQL coverage to make the presentation easier to follow and easier to reference. We start with simple SQL statements to familiarize students with the basic SQL syntax and environment. This provides students the confidence to transition to the more advanced SQL features and commands. These changes provide a better flow of material. Additionally, more SQL examples and figures have been added to help students better visualize and understand the code that is presented.

Aside from enhancing the already strong coverage of database design, we made other improvements in the topical coverage. In particular, the continued growth of Big Data and NoSQL technologies continue to challenge the status quo in the database industry. Therefore, we have added two new online appendices on MongoDB and Neo4j, two of the most important of the NoSQL offerings. This new material provides coding examples and data files to allow students to gain hands-on experience using two of the most popular NoSQL databases. The thirteenth edition also presents a major step forward in the integration of digital content with the text through online, automatically graded coding labs that allow students to write SQL code in an interactive environment that can grade and provide feedback on problems. Here are a few of the highlights of changes in the thirteenth edition:

- Streamlined and reorganized coverage of normalization for enhanced clarity
- Complete reorganization of SQL and Advanced SQL chapters to improve flow and make references to keywords and techniques easier to access
- Expanded coverage of MongoDB with hands-on exercises for querying MongoDB databases (Appendix P)
- Expanded coverage of Neo4j with hands-on exercises for querying graph databases using Cypher (Appendix Q)
- New and expanded coverage of data visualization tools and techniques

This thirteenth edition continues to provide a solid and practical foundation for the design, implementation, and management of database systems. This foundation is built on the notion that, while databases are very practical, their successful creation depends on understanding the important concepts that define them. It's not easy to come up with the proper mix of theory and practice, but the previously mentioned feedback suggests that we largely succeeded in our quest to maintain the proper balance.

The Approach: A Continued Emphasis on Design

As the title suggests, ***Database Systems: Design, Implementation, and Management*** covers three broad aspects of database systems. However, for several important reasons, special attention is given to database design.

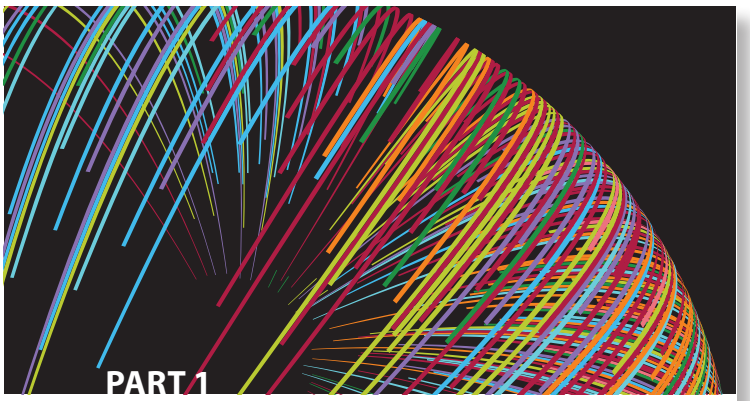
- The availability of excellent database software enables people with little experience to create databases and database applications. Unfortunately, the "create without design" approach usually paves the road to a number of database disasters. In our experience, many database system failures are traceable to poor design and cannot be solved with the help of even the best programmers and managers. Nor is better DBMS software likely to overcome problems created or magnified by poor design. Even the best bricklayers and carpenters can't create a good building from a bad blueprint.
- Most vexing problems of database system management seem to be triggered by poorly designed databases. It hardly seems worthwhile to use scarce resources to develop excellent database management skills merely to use them on crises induced by poorly designed databases.
- Design provides an excellent means of communication. Clients are more likely to get what they need when database system design is approached carefully and thoughtfully. In fact, clients may discover how their organizations really function once a good database design is completed.
- Familiarity with database design techniques promotes understanding of current database technologies. For example, because data warehouses derive much of their data from operational databases, data warehouse concepts, structures, and procedures make more sense when the operational database's structure and implementation are understood.

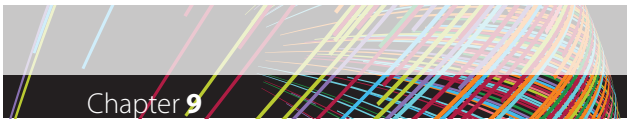
Because the practical aspects of database design are stressed, we have covered design concepts and procedures in detail, making sure that the numerous end-of-chapter problems and cases are sufficiently challenging, so students can develop real and useful design skills. We also make sure that students understand the potential and actual conflicts between database design elegance, information requirements, and transaction processing speed. For example, it makes little sense to design databases that meet design elegance standards while they fail to meet end-user information requirements. Therefore, we explore the use of carefully defined trade-offs to ensure that the databases meet end-user requirements while conforming to high design standards.

Topical Coverage

The Systems View

The book's title begins with ***Database Systems***. Therefore, we examine the database and design concepts covered in Chapters 1–6 as part of a larger whole by placing them within the systems analysis framework of Chapter 9. Database designers who fail to understand that the database is part of a larger system are likely to overlook important design requirements. In fact, Chapter 9, Database Design, provides the map for the advanced database design developed in Appendices B and C. Within the larger systems framework, we can also explore issues such as transaction management and concurrency control (Chapter 10), distributed database management systems (Chapter 12), business intelligence and data warehouses (Chapter 13), new technologies for Big Data (Chapter 14), database connectivity and web technologies (Chapter 15), and database administration and security (Chapter 16).





Database Design

After completing this chapter, you will be able to:

- Describe the role of database design as the foundation of a successful information system
- Describe the five phases in the Systems Development Life Cycle (SDLC)
- Design databases using the six phases in the Database Life Cycle (DBLC) framework
- Conduct evaluation and revision within the SDLC and DBLC frameworks
- Distinguish between top-down and bottom-up approaches in database design
- Distinguish between centralized and decentralized conceptual database design

Preview

Databases are a part of a larger picture called an information system. Database designs that fail to recognize this fact are not likely to be successful. Database designers must recognize that the database is a critical means to an end rather than an end in itself. Managers want the database to serve their management needs, but too many databases seem to force managers to alter their routines to fit the database requirements.

Information systems don't just happen; they are the product of a carefully staged development process. Systems analysis is used to determine the need for an information system and to establish its limits. Within systems analysis, the actual information system is created through a process known as systems development.

The creation and evolution of information systems follows an iterative pattern called the Systems Development Life Cycle (SDLC), which is a continuous process of creation, maintenance, enhancement, and replacement of the information system. A similar cycle applies to databases: the database is created, maintained, enhanced, and eventually replaced. The Database Life Cycle (DBLC) is carefully traced in this chapter, and is shown in the context of the larger Systems Development Life Cycle.

At the end of the chapter, you will be introduced to some classical approaches to database design: top-down versus bottom-up and centralized versus decentralized.

Data Files Available on cengagebrain.com

Note

Because it is purely conceptual, this chapter does not reference any data files.

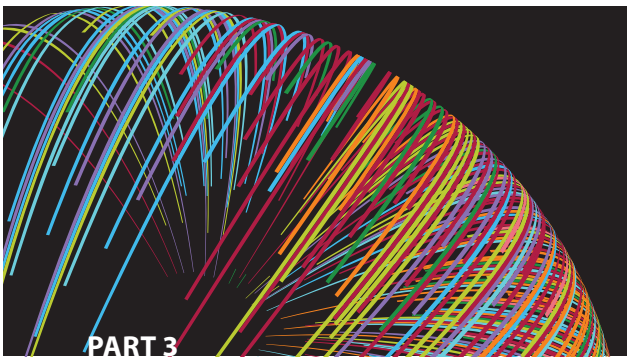
Database Design

The first item in the book's subtitle is ***Design***, and our examination of database design is comprehensive. For example, Chapters 1 and 2 examine the development and future of databases and data models and illustrate the need for design. Chapter 3 examines the details of the relational database model; Chapter 4 provides extensive, in-depth, and practical database design coverage; and Chapter 5 explores advanced database design topics. Chapter 6 is devoted to critical normalization issues that affect database efficiency and effectiveness. Chapter 9 examines database design within the systems framework and maps the activities required to successfully design and implement the complex, real-world database developed in Appendices B and C. Appendices A1 and A2 are good introductory tutorials on designing databases with Visio Professional 2010 and Visio 2013, respectively.

Because database design is affected by real-world transactions, the way data is distributed, and ever-increasing information requirements, we examine major database features that must be supported in current-generation databases and models. For example, Chapter 10, Transaction Management and Concurrency Control, focuses on the characteristics of database transactions and how they affect database integrity and consistency. Chapter 11, Database Performance Tuning and Query Optimization, illustrates the need for query efficiency in a world that routinely generates and uses terabyte-size databases and tables with millions of records. Chapter 12, Distributed Database Management Systems, focuses on data distribution, replication, and allocation. In Chapter 13, Business Intelligence and Data Warehouses, we explore the characteristics of databases that are used in decision support and online analytical processing, including coverage of data visualization and data analytics. Chapter 14, Big Data and NoSQL, explores the challenges of leveraging nonrelational databases to use vast global stores of unstructured data. Chapter 15, Database Connectivity and Web Technologies, covers the basic database connectivity issues in a web-based data world, development of web-based database front ends, and emerging cloud-based services.

Implementation

The second portion of the subtitle is ***Implementation***. We use Structured Query Language (SQL) in Chapters 7 and 8 to show how relational databases are implemented and managed. Appendix M, Microsoft Access Tutorial, provides a quick but comprehensive guide to implementing an MS Access database. Appendices B and C demonstrate the design of a database that was fully implemented; these appendices illustrate a wide range of implementation issues. We had to deal with conflicting design goals: design elegance, information requirements, and operational speed. Therefore, we carefully audited the initial design in Appendix B to check its ability to meet end-user needs and establish appropriate implementation protocols. The result of this audit yielded the final design developed in Appendix C. While relational databases are still the appropriate database technology to use in the vast majority of situations, Big Data issues have created an environment in which special requirements can call for the use of new, nonrelational technologies. Chapter 14, Big Data and NoSQL, describes the types of data that are appropriate for these new technologies and the array of options available in these special cases. Appendix P, Working with MongoDB, and Appendix Q, Working with Neo4j, provide hands-on coverage of using MongoDB and Neo4j, some of the most popular NoSQL options. The



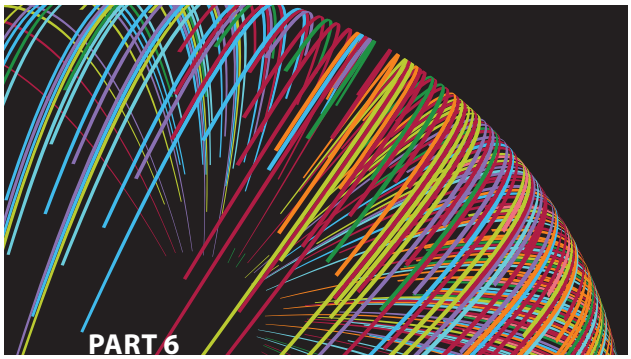
Advanced Design and Implementation

7 Introduction to Structured Query Language (SQL)
8 Advanced SQL
9 Database Design

special issues encountered in an Internet database environment are addressed in Chapter 15, Database Connectivity and Web Technologies, and in Appendix J, Web Database Development with ColdFusion.

Management

The final portion of the subtitle is ***Management***. We deal with database management issues in Chapter 10, Transaction Management and Concurrency Control; Chapter 12, Distributed Database Management Systems; and Chapter 16, Database Administration and Security. Chapter 11, Database Performance Tuning and Query Optimization, is a valuable resource that illustrates how a DBMS manages data retrieval. In addition, Appendix N, Creating a New Database Using Oracle 12c, walks you through the process of setting up a new database.



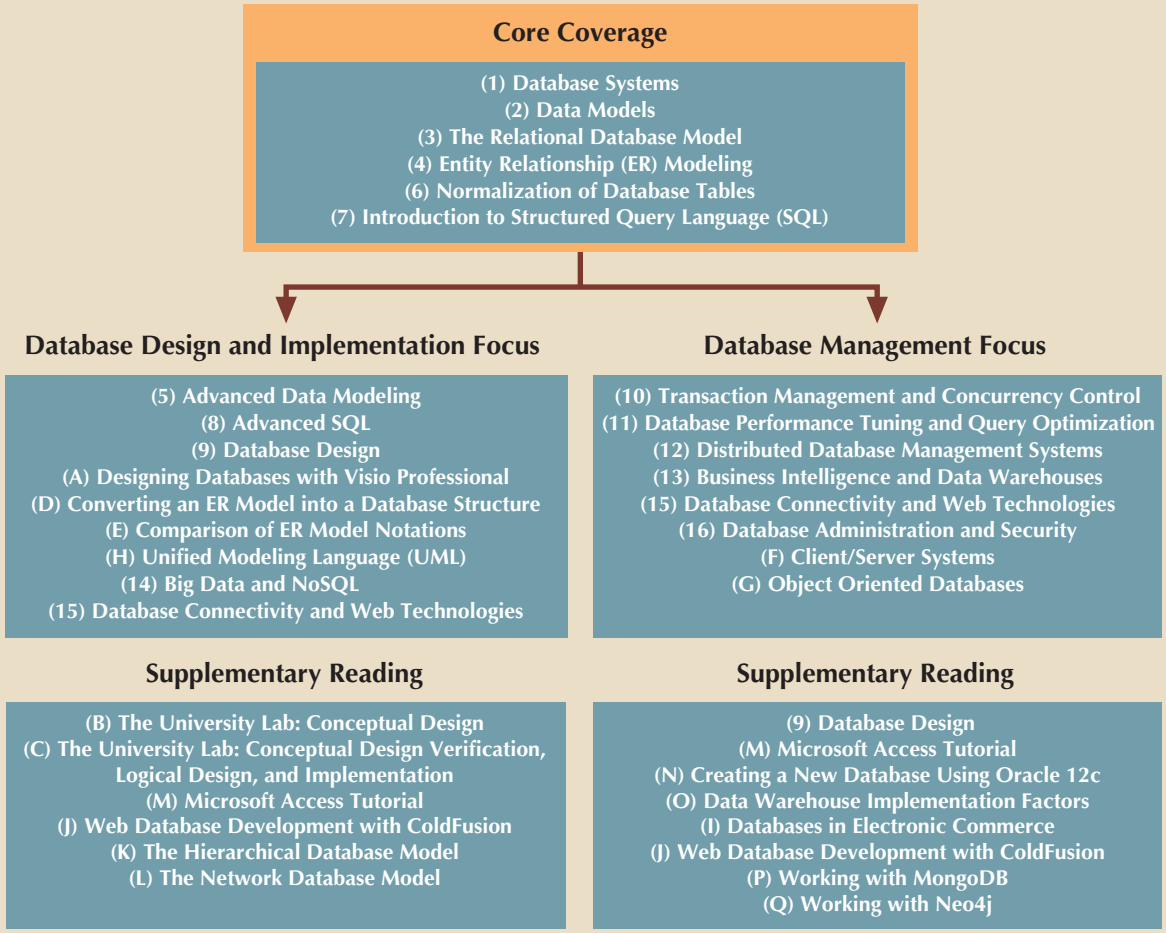
Teaching Database: A Matter of Focus

Given the wealth of detailed coverage, instructors can "mix and match" chapters to produce the desired coverage. Depending on where database courses fit into the curriculum, instructors may choose to emphasize database design or database management. (See Figure 1.)

The hands-on nature of database design lends itself particularly well to class projects in which students use instructor-selected software to prototype a system that they design for the end user. Several end-of-chapter problems are sufficiently complex to serve as projects, or an instructor may work with local businesses to give students hands-on experience. Note that some elements of the database design track are also found in the database management track, because it is difficult to manage database technologies that are not well understood.

The options shown in Figure 1 serve only as a starting point. Naturally, instructors will tailor their coverage based on their specific course requirements. For example, an instructor may decide to make Appendix I an outside reading assignment and make Appendix A a self-taught tutorial, and then use that time to cover client/server systems or object-oriented databases. The latter choice would serve as a gateway to UML coverage.

FIGURE 1



Text Features

Online Content boxes draw attention to material at *www.cengagebrain.com* for this text and provide ideas for incorporating this content into the course.

Online Content

All of the databases used to illustrate the material in this chapter (see the Data Files list at the beginning of the chapter) are available at *www.cengagebrain.com*. The database names match the database names shown in the figures.

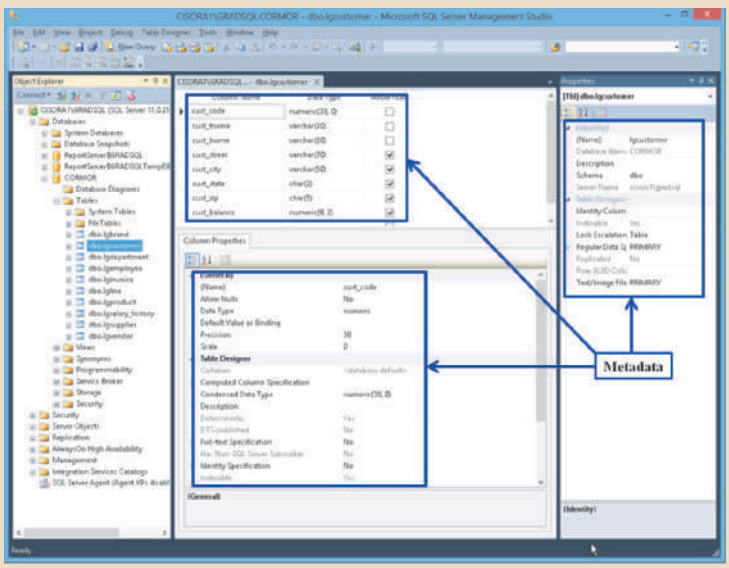
Notes highlights important facts about the concepts introduced in the chapter.

Note

This chapter focuses on SELECT queries to retrieve data from tables. Chapter 8 will explain how those tables are actually created and how the data is loaded into them. This reflects the experience of most entry-level database positions. As a new hire working with databases, you will likely spend quite a bit of time retrieving data from tables that already exist before you begin creating new tables and modifying the data.

A variety of **four-color figures**, including ER models and implementations, tables, and illustrations, clearly illustrate difficult concepts.

FIGURE 1.12 ILLUSTRATING METADATA WITH MICROSOFT SQL SERVER EXPRESS



Summary

- An information system is designed to help transform data into information and to manage both data and information. Thus, the database is a very important part of the information system. Systems analysis is the process that establishes the need for an information system and its extent. Systems development is the process of creating an information system

A robust **Summary** at the end of each chapter ties together the major concepts and serves as a quick review for students.

Key Terms

bottom-up design
boundaries
centralized design
clustered tables
cohesivity
computer-aided software engineering (CASE)
conceptual design
database development
database fragment
Database Life Cycle (DBLC)
database role
decentralized design
description of operations
differential backup
full backup
information system
logical design
minimal data rule
module
module coupling
physical design
scope
systems analysis
systems development
Systems Development Life Cycle (SDLC)
top-down design
transaction log backup
virtualization

An alphabetic list of **Key Terms** summarizes important terms.

Review Questions

1. What is an information system? What is its purpose?
2. How do systems analysis and systems development fit into a discussion about information systems?
3. What does the acronym SDLC mean, and what does an SDLC portray?
4. What does the acronym DBLC mean, and what does a DBLC portray?
5. Discuss the distinction between centralized and decentralized conceptual database design.

Review Questions challenge students to apply the skills learned in each chapter.

Problems

In the following exercises, you will set up database connectivity using MS Excel.

1. Use MS Excel to connect to the Ch02_InsureCo MS Access database using ODBC, and retrieve all of the AGENTs.
2. Use MS Excel to connect to the Ch02_InsureCo MS Access database using ODBC, and retrieve all of the CUSTOMERs.

Problems become progressively more complex as students draw on the lessons learned from the completion of preceding problems.

Additional Features

MindTap® for Database Systems 13e

MindTap® combines learning tools—such as readings, multimedia, activities, and assessments—into a singular learning path that guides students through the course. You'll find a full ebook as well as a robust set of auto-gradable homework problems. Multiple-choice homework questions developed from the end-of-chapter review questions confirm students' understanding of core concepts and key terms. Higher-level assignments enable students to practice database design concepts in an automated environment, and chapter quizzes help prepare students for exams. Students will also benefit from the chapter-opening videos created by the authors, as well as study tools such as crossword puzzles and key-term flashcards.

MindTap® is designed to be fully integrated with any Learning Management System and can be used as a stand-alone product or in conjunction with a print textbook.

Appendices

Eighteen online appendices provide additional material on a variety of important areas, such as using Microsoft® Visio® and Microsoft® Access®, ER model notations, UML, object-oriented databases, databases and electronic commerce, Adobe® ColdFusion®, and working with newer NoSQL databases MongoDB and Neo4j.

Database, SQL Script, JSON Documents, and ColdFusion Files

The online materials for this book include all of the database structures and table contents used in the text. For students using Oracle®, MySQL, and Microsoft SQL Server™, SQL scripts are included to help students create and load all tables used in the SQL chapters (7 and 8). Text documents for importing JSON-formatted documents into MongoDB and a script for creating a graph database in Neo4j (Appendices P and Q) are also included. In addition, all ColdFusion scripts used to develop the web interfaces in Appendix J are included.

Instructor Resources

Database Systems: Design, Implementation, and Management, Thirteenth Edition, includes teaching tools to support instructors in the classroom. The ancillary material that accompanies the textbook is listed below. They are available on the web at *www.cengagebrain.com.*

Instructor's Manual

The authors have created this manual to help instructors make their classes informative and interesting. Because the authors tackle so many problems in depth, instructors will find the *Instructor's Manual* especially useful. The details of the design solution process are shown in the *Instructor's Manual*, as well as notes about alternative approaches that may be used to solve a particular problem.

SQL Script Files for Instructors

The authors have provided teacher's SQL script files to allow instructors to cut and paste the SQL code into the SQL windows. (Scripts are provided for Oracle, MySQL, and MS SQL Server.) The SQL scripts, which have all been tested by Cengage Learning, are a major convenience for instructors. You won't have to type in the SQL commands, and the use of the scripts eliminates typographical errors that are sometimes difficult to trace.

ColdFusion Files for Instructors

The ColdFusion web development solutions are provided. Instructors have access to a menu-driven system that allows teachers to show the code as well as its execution.

Databases

For many chapters, Microsoft® Access® instructor databases are available that include features not found in the student databases. For example, the databases that accompany Chapters 7 and 8 include many of the queries that produce the problem solutions. Other Access databases, such as the ones that accompany Chapters 3, 4, 5, and 6, include implementations of the design problem solutions to allow instructors to illustrate the effect of design decisions. In addition, instructors have access to all the script files for Oracle, MySQL, and MS SQL Server so that all the databases and their tables can be converted easily and precisely.

Cengage Learning Testing Powered by Cognero

A flexible, online system that allows you to:

- Author, edit, and manage test bank content from multiple Cengage Learning solutions
- Create multiple test versions in an instant
- Deliver tests from your LMS, your classroom, or wherever you want

Start right away!

Cengage Learning Testing Powered by Cognero works on any operating system or browser.

- No special installs or downloads needed
- Create tests from school, home, the coffee shop—anywhere with Internet access

What will you find?

- Simplicity at every step. A desktop-inspired interface features drop-down menus and familiar, intuitive tools that take you through content creation and management with ease.
- Full-featured test generator. Create ideal assessments with your choice of 15 question types (including true/false, multiple-choice, opinion scale/Likert, and essay). Multi-language support, an equation editor, and unlimited metadata help ensure your tests are complete and compliant.
- Cross-compatible capability. Import and export content into other systems.

PowerPoint® Presentations

Microsoft PowerPoint slides are included for each chapter. Instructors can use the slides in a variety of ways—for example, as teaching aids during classroom presentations or as printed handouts for classroom distribution. Instructors can modify these slides or include slides of their own for additional topics introduced to the class.

Figure Files

Figure files for solutions are presented in the *Instructor's Manual* to allow instructors to create their own presentations. Instructors can also manipulate these files to meet their particular needs.

Acknowledgments

Regardless of how many editions of this book are published, they will always rest on the solid foundation created by the first edition. We remain convinced that our work has become successful because that first edition was guided by Frank Ruggirello, a former Wadsworth senior editor and publisher. Aside from guiding the book's development, Frank also managed to solicit the great Peter Keen's evaluation (thankfully favorable) and subsequently convinced Peter Keen to write the foreword for the first edition. Although we sometimes found Frank to be an especially demanding taskmaster, we also found him to be a superb professional and a fine friend. We suspect Frank will still see his fingerprints all over our current work. Many thanks.

A difficult task in rewriting a book is deciding what new approaches, topical coverage, and changes to depth of coverage are appropriate for a product that has successfully weathered the test of the marketplace. The comments and suggestions made by the book's adopters, students, and reviewers play a major role in deciding what coverage is desirable and how that coverage is to be treated.

Some adopters became extraordinary reviewers, providing incredibly detailed and well-reasoned critiques even as they praised the book's coverage and style. Dr. David Hatherly, a superb database professional who is a senior lecturer in the School of Information Technology, Charles Sturt University–Mitchell, Bathhurst, Australia, made sure that we knew precisely what issues led to his critiques. Even better for us, he provided the suggestions that made it much easier for us to improve the topical coverage in earlier editions. All of his help was given freely and without prompting on our part. His efforts are much appreciated, and our thanks are heartfelt.

We also owe a debt of gratitude to Professor Emil T. Cipolla, who teaches at St. Mary College. Professor Cipolla's wealth of IBM experience turned out to be a valuable resource when we tackled the embedded SQL coverage in Chapter 8.

Every technical book receives careful scrutiny by several groups of reviewers selected by the publisher. We were fortunate to face the scrutiny of reviewers who were superbly qualified to offer their critiques, comments, and suggestions—many of which strengthened this edition. While holding them blameless for any remaining shortcomings, we owe these reviewers many thanks for their contributions:

Laurie Crawford,
Franklin University

John E. MacDonald IV,
Binghamton University

Mava Wilson,
Lee University

In some respects, writing books resembles building construction: When 90 percent of the work seems done, 90 percent of the work remains to be done. Fortunately for us, we had a great team on our side.

- We are deeply indebted to Deb Kaufmann for her help and guidance. Deb has been everything we could have hoped for in a development editor and more. Deb has been our editor for almost all the editions of this book, and the quality of her work shows in the attention to detail and the cohesiveness and writing style of the material in this book.
- After writing so many books and thirteen editions of *this* book, we know just how difficult it can be to transform the authors' work into an attractive product. The production team, both at Cengage (Michele Stulga) and Lumina Datamatics (Kiruthiga Sowndararajan), have done an excellent job.

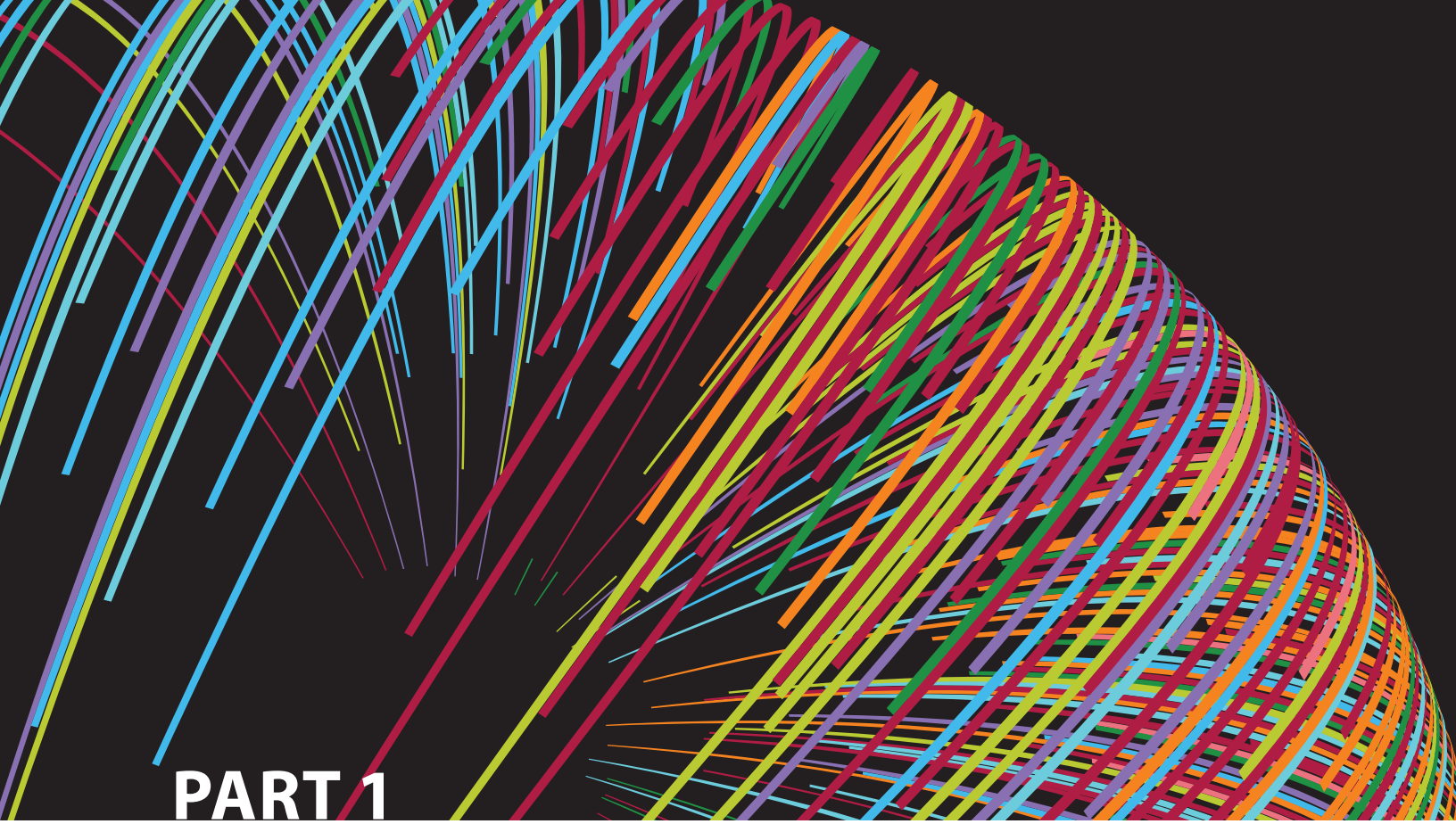
- We also owe Maria Garguilo, our Content Developer, special thanks for her ability to guide this book to a successful conclusion, and John Freitas, our technical editor, deserves many thanks for making sure all code and technical references were accurate.

We also thank our students for their comments and suggestions. They are the reason for writing this book in the first place. One comment stands out in particular: "I majored in systems for four years, and I finally discovered why when I took your course." And one of our favorite comments by a former student was triggered by a question about the challenges created by a real-world information systems job: "Doc, it's just like class, only easier. You really prepared me well. Thanks!"

Special thanks go to a very unique and charismatic gentleman. For over 20 years, Peter Rob has been the driving force behind the creation and evolution of this book. This book originated as a product of his drive and dedication to excellence. For over 22 years, he was the voice of *Database Systems* and the driving force behind its advancement. We wish him peace in his retirement, time with his loved ones, and luck on his many projects.

Last, and certainly not least, we thank our families for their solid support at home. They graciously accepted the fact that during more than a year's worth of rewriting, there would be no free weekends, rare free nights, and even rarer free days. We owe you much, and the dedications we wrote are but a small reflection of the important space you occupy in our hearts.

Carlos Coronel and Steven Morris



PART 1
Database Concepts

1 Database Systems

2 Data Models

Chapter 1

Database Systems

After completing this chapter, you will be able to:

- Define the difference between data and information
- Describe what a database is, the various types of databases, and why they are valuable assets for decision making
- Explain the importance of database design
- See how modern databases evolved from file systems
- Understand flaws in file system data management
- Outline the main components of the database system
- Describe the main functions of a database management system (DBMS)

Preview

Organizations use data to keep track of their day-to-day operations. Such data is used to generate information, which in turn is the basis for good decisions. Data is likely to be managed most efficiently when it is stored in a database. Databases are involved in almost all facets and activities of our daily lives: from school to work, medical care, government, nonprofit organizations, and houses of worship. In this chapter, you will learn what a database is, what it does, and why it yields better results than other data management methods. You will also learn about various types of databases and why database design is so important.

Databases evolved from the need to manage large amounts of data in an organized and efficient manner. In the early days, computer file systems were used to organize such data. Although file system data management is now largely outmoded, understanding the characteristics of file systems is important because file systems are the source of serious data management limitations. In this chapter, you will also learn how the database system approach helps eliminate most of the shortcomings of file system data management.

Data Files and Available Formats

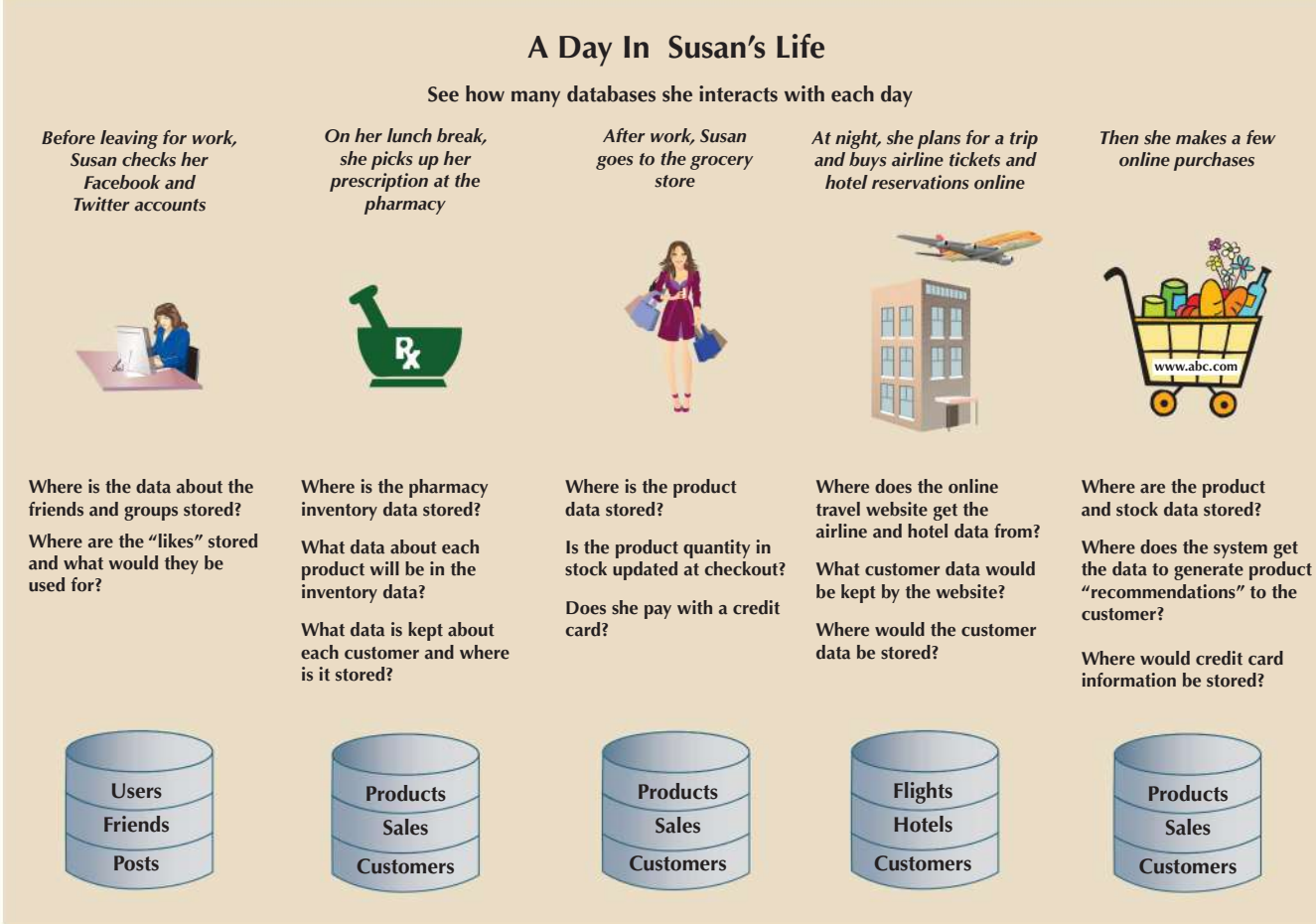
	MS Access	Oracle	MS SQL	My SQL		MS Access	Oracle	MS SQL	My SQL
CH01_Text	✓	✓	✓	✓	CH01_Problems	✓	✓	✓	✓

Data Files Available on cengagebrain.com

1-1 Why Databases?

So, why do we need databases? In today's world, data is ubiquitous (abundant, global, everywhere) and pervasive (unescapable, prevalent, persistent). From birth to death, we generate and consume data. The trail of data starts with the birth certificate and continues all the way to a death certificate (and beyond!). In between, each individual produces and consumes enormous amounts of data. As you will see in this book, databases are the best way to store and manage data. Databases make data persistent and shareable in a secure way. As you look at Figure 1.1, can you identify some of the data generated by your own daily activities?

FIGURE 1.1 THE PERVASIVE NATURE OF DATABASES



Data is not only ubiquitous and pervasive; it is also essential for organizations to survive and prosper. Imagine trying to operate a business without knowing who your customers are, what products you are selling, who is working for you, who owes you money, and to whom you owe money. All businesses have to keep this type of data and much more. Just as important, they must have that data available to decision makers when necessary. It can be argued that the ultimate purpose of all business information systems is to help businesses use information as an organizational resource. At the heart of all of these systems are the collection, storage, aggregation, manipulation, dissemination, and management of data.

Depending on the type of information system and the characteristics of the business, this data could vary from a few megabytes on just one or two topics to terabytes covering hundreds of topics within the business's internal and external environment.

Telecommunications companies, such as Sprint and AT&T, are known to have systems that keep data on trillions of phone calls, with new data being added to the system at speeds up to 70,000 calls per second! Not only do these companies have to store and manage immense collections of data but also they have to be able to find any given fact in that data quickly. Consider the case of Internet search staple Google. While Google is reluctant to disclose many details about its data storage specifications, it is estimated that the company responds to over 91 million searches per day across a collection of data that is several terabytes in size. Impressively, the results of these searches are available almost instantly.

How can these businesses process this much data? How can they store it all, and then quickly retrieve just the facts that decision makers want to know, just when they want to know it? The answer is that they use databases. Databases, as explained in detail throughout this book, are specialized structures that allow computer-based systems to store, manage, and retrieve data very quickly. Virtually all modern business systems rely on databases. Therefore, a good understanding of how these structures are created and their proper use is vital for any information systems professional. Even if your career does not take you down the amazing path of database design and development, databases will be a key component of the systems that you use. In any case, you will probably make decisions in your career based on information generated from data. Thus, it is important that you know the difference between data and information.

1-2 Data versus Information

To understand what drives database design, you must understand the difference between data and information. **Data** consists of raw facts. The word *raw* indicates that the facts have not yet been processed to reveal their meaning. For example, suppose that a university tracks data on faculty members for reporting to accrediting bodies. To get the data for each faculty member into the database, you would provide a screen to allow for convenient data entry, complete with drop-down lists, combo boxes, option buttons, and other data-entry validation controls. Figure 1.2(a) shows a simple data-entry form from a software package named Sedona. When the data is entered into the form and saved, it is placed in the underlying database as raw data, as shown in Figure 1.2(b). Although you now have the facts in hand, they are not particularly useful in this format. Reading through hundreds of rows of data for faculty members does not provide much insight into the overall makeup of the faculty. Therefore, you transform the raw data into a data summary like the one shown in Figure 1.2(c). Now you can get quick answers to questions such as "What percentage of the faculty in the Information Systems (INFS) department are adjuncts?" In this case, you can quickly determine that 20 percent of the INFS faculty members are adjunct faculty. Because graphics can enhance your ability to quickly extract meaning from data, you show the data summary pie chart in Figure 1.2(d).

Information is the result of processing raw data to reveal its meaning. Data processing can be as simple as organizing data to reveal patterns or as complex as making forecasts or drawing inferences using statistical modeling. To reveal meaning, information requires *context*. For example, an average temperature reading of 105 degrees does not mean much unless you also know its context: Is this reading in degrees Fahrenheit or Celsius? Is this a machine temperature, a body temperature, or an outside air temperature? Information can be used as the foundation for decision making. For example, the data summary for the faculty can provide accrediting bodies with insights that are useful in determining whether to renew accreditation for the university.

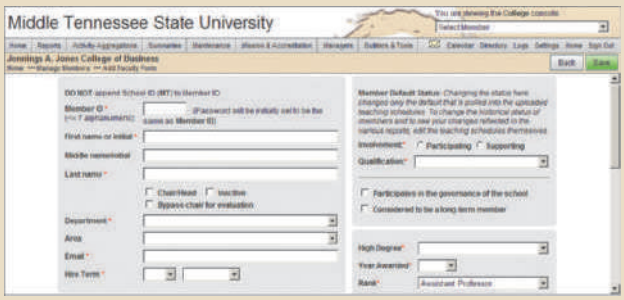
Keep in mind that raw data must be properly *formatted* for storage, processing, and presentation. For example, dates might be stored in Julian calendar formats within the database, but displayed in a variety of formats, such as day-month-year or month/day/year, for different purposes. Respondents' yes/no responses might need to be converted

data
Raw facts, or facts that have not yet been processed to reveal their meaning to the end user.

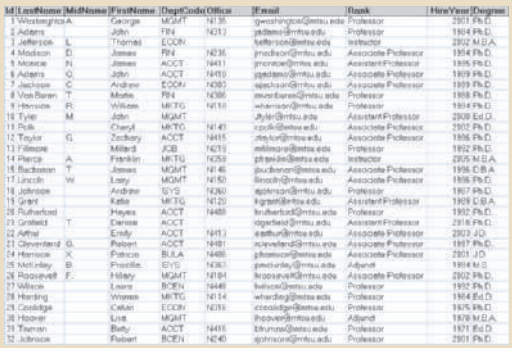
information
The result of processing raw data to reveal its meaning. Information consists of transformed data and facilitates decision making.

FIGURE 1.2 TRANSFORMING RAW DATA INTO INFORMATION

a) Data entry screen



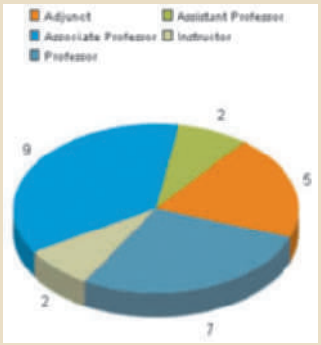
b) Raw data



c) Information in summary format

Rank	COUNT	%INFS	TOTCOL	%COL. TOT.	%COL. FAC.
Adjunct	5	20.00%	23	21.74%	3.27%
Assistant Professor	2	8.00%	28	7.14%	1.31%
Associate Professor	9	36.00%	37	24.32%	5.88%
Instructor	2	8.00%	18	11.11%	1.31%
Professor	7	28.00%	47	14.89%	4.58%

d) Information in graphical format



to a Y/N or 0/1 format for data storage. More complex formatting is required when working with complex data types, such as sounds, videos, or images.

In this "information age," production of accurate, relevant, and timely information is the key to good decision making. In turn, good decision making is the key to business survival in a global market. We are now said to be entering the "knowledge age."[1]

Data is the foundation of information, which is the bedrock of **knowledge**—that is, the body of information and facts about a specific subject. Knowledge implies familiarity, awareness, and understanding of information as it applies to an environment. A key characteristic of knowledge is that "new" knowledge can be derived from "old" knowledge.

Let's summarize some key points:

- Data constitutes the building blocks of information.
- Information is produced by processing data.
- Information is used to reveal the meaning of data.
- Accurate, relevant, and timely information is the key to good decision making.
- Good decision making is the key to organizational survival in a global environment.

The previous paragraphs have explained the importance of data, and how the processing of data is used to reveal information that in turn generates "actionable" knowledge. Let's explore a simple example of how this works in the real world.

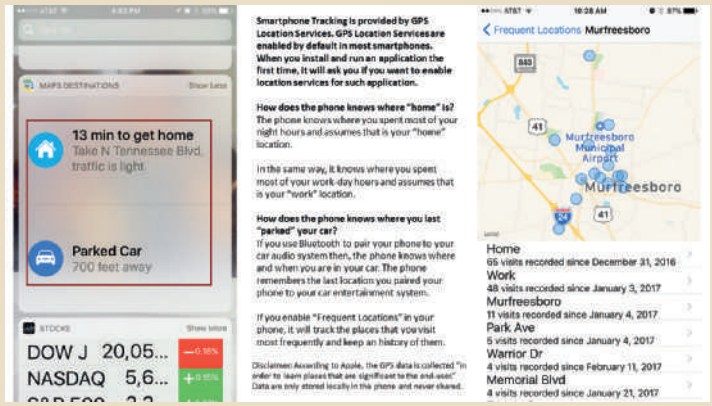
In today's information-centric society, we use smartphones on a daily basis. These devices have advanced GPS functionality that constantly tracks your whereabouts. This data is stored and shared with various applications. When you get a new smartphone,

knowledge
The body of information and facts about a specific subject. Knowledge implies familiarity, awareness, and understanding of information as it applies to an environment. A key characteristic is that new knowledge can be derived from old knowledge.

[1] Peter Drucker coined the phrase "knowledge worker" in 1959 in his book *Landmarks of Tomorrow.* In 1994, Esther Dyson, George Keyworth, and Dr. Alvin Toffler introduced the concept of the "knowledge age."

you can use the map application to go places and to set up your home address (now the phone knows where you live!). The GPS feature in your phone tracks your daily locations. In some cases, the information generated is very helpful: it can help you navigate to various locations and even to find where you parked your car. Figure 1.3 shows screenshots from one of the authors' smartphone. The phone "knows" that this is about the time he goes home and tells him how long it is going to take to get there. It also tells him where he parked his car; if he clicks the Parked Car icon, it will open a map so he can locate the car.

FIGURE 1.3 SMARTPHONE TRACKING



Furthermore, and maybe even scarier in terms of privacy issues, your smartphone may know more about your activities than you imagine. For example, suppose that every Wednesday night you go to the gym and play indoor soccer with your friends. Next Wednesday night, 20 minutes before you leave home, your phone pops up a message saying "19 minutes to [gym address]. Traffic is light." The phone has been storing GPS data on your movements to develop patterns based on days, times, and locations to generate this knowledge. It can then associate such knowledge as your daily activities provide more data points. Imagine that on Wednesday when you go to the Magic Box gym to play soccer, when you arrive you use Facebook on your phone to check in to the gym. Now, your phone also knows the name of the place where you go every Wednesday night.

As you can see from this example, knowledge and information require timely and accurate data. Such data must be properly generated and stored in a format that is easy to access and process. In addition, like any basic resource, the data environment must be managed carefully. **Data management** is a discipline that focuses on the proper generation, storage, and retrieval of data. Given the crucial role that data plays, it should not surprise you that data management is a core activity for any business, government agency, service organization, or charity.

data management
A process that focuses on data collection, storage, and retrieval. Common data management functions include addition, deletion, modification, and listing.

database
A shared, integrated computer structure that houses a collection of related data. A database contains two types of data: end-user data (raw facts) and metadata.

metadata
Data about data; that is, data about data characteristics and relationships. See also *data dictionary*.

1-3 Introducing the Database

Efficient data management typically requires the use of a computer database. A **database** is a shared, integrated computer structure that stores a collection of the following:

- End-user data—that is, raw facts of interest to the end user
- **Metadata**, or data about data, through which the end-user data is integrated and managed

The metadata describes the data characteristics and the set of relationships that links the data found within the database. For example, the metadata component stores information such as the name of each data element, the type of values (numeric, dates, or text) stored on each data element, and whether the data element can be left empty. The metadata provides information that complements and expands the value and use of the data. In short, metadata presents a more complete picture of the data in the database. Given the characteristics of metadata, you might hear a database described as a "collection of *self-describing* data."

A **database management system (DBMS)** is a collection of programs that manages the database structure and controls access to the data stored in the database. In a sense, a database resembles a very well-organized electronic filing cabinet in which powerful software (the DBMS) helps manage the cabinet's contents.

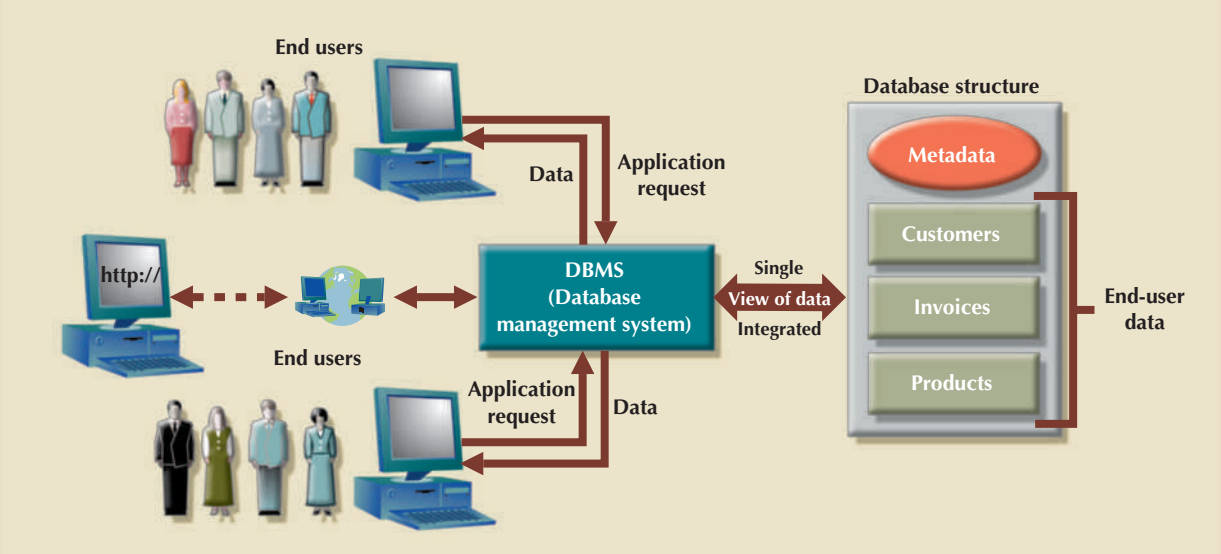
1-3a Role and Advantages of the DBMS

The DBMS serves as the intermediary between the user and the database. The database structure itself is stored as a collection of files, and the only way to access the data in those files is through the DBMS. Figure 1.4 emphasizes the point that the DBMS presents the end user (or application program) with a single, integrated view of the data in the database. The DBMS receives all application requests and translates them into the complex operations required to fulfill those requests. The DBMS hides much of the database's internal complexity from the application programs and users. The application program might be written by a programmer using a programming language, such as Visual Basic.NET, Java, or C#, or it might be created through a DBMS utility program.

database management system (DBMS)
The collection of programs that manages the database structure and controls access to the data stored in the database.

Having a DBMS between the end user's applications and the database offers some important advantages. First, the DBMS enables the data in the database *to be shared* among multiple applications or users. Second, the DBMS *integrates* the many different users' views of the data into a single all-encompassing data repository.

FIGURE 1.4 THE DBMS MANAGES THE INTERACTION BETWEEN THE END USER AND THE DATABASE



Because data is the crucial raw material from which information is derived, you must have a good method to manage such data. As you will discover in this book, the DBMS helps make data management more efficient and effective. In particular, a DBMS provides these advantages:

- *Improved data sharing*. The DBMS helps create an environment in which end users have better access to more and better-managed data. Such access makes it possible for end users to respond quickly to changes in their environment.
- *Improved data security*. The more users access the data, the greater the risks of data security breaches. Corporations invest considerable amounts of time, effort, and money to ensure that corporate data is used properly. A DBMS provides a framework for better enforcement of data privacy and security policies.
- *Better data integration*. Wider access to well-managed data promotes an integrated view of the organization's operations and a clearer view of the big picture. It becomes much easier to see how actions in one segment of the company affect other segments.
- *Minimized data inconsistency*. **Data inconsistency** exists when different versions of the same data appear in different places. For example, data inconsistency exists when a company's sales department stores a sales representative's name as Bill Brown and the company's personnel department stores that same person's name as William G. Brown, or when the company's regional sales office shows the price of a product as $45.95 and its national sales office shows the same product's price as $43.95. The probability of data inconsistency is greatly reduced in a properly designed database.
- *Improved data access*. The DBMS makes it possible to produce quick answers to ad hoc queries. From a database perspective, a **query** is a specific request issued to the DBMS for data manipulation—for example, to read or update the data. Simply put, a query is a question, and an **ad hoc query** is a spur-of-the-moment question. The DBMS sends back an answer (called the **query result set**) to the application. For example, when dealing with large amounts of sales data, end users might want quick answers to questions (ad hoc queries). Some examples are the following:
  - What was the dollar volume of sales by product during the past six months?
  - What is the sales bonus figure for each of our salespeople during the past three months?
  - How many of our customers have credit balances of $3,000 or more?
- *Improved decision making*. Better-managed data and improved data access make it possible to generate better-quality information, on which better decisions are based. The quality of the information generated depends on the quality of the underlying data. **Data quality** is a comprehensive approach to promoting the accuracy, validity, and timeliness of the data. While the DBMS does not guarantee data quality, it provides a framework to facilitate data quality initiatives. Data quality concepts will be covered in more detail in Chapter 16, Database Administration and Security.
- *Increased end-user productivity*. The availability of data, combined with the tools that transform data into usable information, empowers end users to make quick, informed decisions that can make the difference between success and failure in the global economy.

The advantages of using a DBMS are not limited to the few just listed. In fact, you will discover many more advantages as you learn more about the technical details of databases and their proper design.

data inconsistency
A condition in which different versions of the same data yield different (inconsistent) results.

query
A question or task asked by an end user of a database in the form of SQL code. A specific request for data manipulation issued by the end user or the application to the DBMS.

ad hoc query
A "spur-of-the-moment" question.

query result set
The collection of data rows returned by a query.

data quality
A comprehensive approach to ensuring the accuracy, validity, and timeliness of data.

1-3b Types of Databases

A DBMS can be used to build many different types of databases. Each database stores a particular collection of data and is used for a specific purpose. Over the years, as technology and innovative uses of databases have evolved, different methods have been used to classify databases. For example, databases can be classified by the number of users supported, where the data is located, the type of data stored, the intended data usage, and the degree to which the data is structured.

The number of users determines whether the database is classified as single user or multiuser. A **single-user database** supports only one user at a time. In other words, if user A is using the database, users B and C must wait until user A is done. A single-user database that runs on a personal computer is called a **desktop database**. In contrast, a **multiuser database** supports multiple users at the same time. When the multiuser database supports a relatively small number of users (usually fewer than 50) or a specific department within an organization, it is called a **workgroup database**. When the database is used by the entire organization and supports many users (more than 50, usually hundreds) across many departments, the database is known as an **enterprise database**.

Location might also be used to classify the database. For example, a database that supports data located at a single site is called a **centralized database**. A database that supports data distributed across several different sites is called a **distributed database**. The extent to which a database can be distributed and the way in which such distribution is managed are addressed in detail in Chapter 12, Distributed Database Management Systems.

Both centralized and decentralized (distributed) databases require a well-defined infrastructure (hardware, operating systems, network technologies, etc.) to implement and operate the database. Typically, the infrastructure is owned and maintained by the organization that creates and operates the database. But in recent years, the use of cloud databases has been growing in popularity. A **cloud database** is a database that is created and maintained using cloud data services, such as Microsoft Azure or Amazon AWS. These services, provided by third-party vendors, provide defined performance measures (data storage capacity, required throughput, and availability) for the database, but do not necessarily specify the underlying infrastructure to implement it. The data owners do not have to know, or be concerned about, what hardware and software are being used to support their databases. The performance capabilities can be renegotiated with the cloud provider as the business demands on the database change. For example, 3M Health Information Systems, the world's largest provider of health care analytics software in hospitals, used Amazon's AWS cloud database services to consolidate its multiple IT centers. 3M did not have to buy, install, configure, or maintain any hardware, operating systems, or network devices. It simply purchased storage and processing capacity for its data and applications. As the demands on the databases increased, additional processing and storage capabilities could be purchased as needed. As a result, server provisioning processes that previously took 10 weeks to complete could be done in mere minutes. This allows the company to be more responsive to the needs of customers and innovate faster.

In some contexts, such as research environments, a popular way of classifying databases is according to the type of data stored in them. Using this criterion, databases are grouped into two categories: general-purpose and discipline-specific databases. **General-purpose databases** contain a wide variety of data used in multiple disciplines—for example, a census database that contains general demographic data and the LexisNexis and ProQuest databases that contain newspaper, magazine, and journal articles for a variety of topics. **Discipline-specific databases** contain data focused on specific subject areas. The data in this type of database is used mainly for academic or research purposes

single-user database
A database that supports only one user at a time.

desktop database
A single-user database that runs on a personal computer.

multiuser database
A database that supports multiple concurrent users.

workgroup database
A multiuser database that usually supports fewer than 50 users or is used for a specific department in an organization.

enterprise database
The overall company data representation, which provides support for present and expected future needs.

centralized database
A database located at a single site.

distributed database
A logically related database that is stored in two or more physically independent sites.

cloud database
A database that is created and maintained using cloud services, such as Microsoft Azure or Amazon AWS.

general-purpose database
A database that contains a wide variety of data used in multiple disciplines.

discipline-specific database
A database that contains data focused on specific subject areas.

within a small set of disciplines. Examples of discipline-specific databases are financial data stored in databases such as CompuStat or CRSP (Center for Research in Security Prices), geographic information system (GIS) databases that store geospatial and other related data, and medical databases that store confidential medical history data.

The most popular way of classifying databases today, however, is based on how they will be used and on the time sensitivity of the information gathered from them. For example, transactions such as product or service sales, payments, and supply purchases reflect critical day-to-day operations. Such transactions must be recorded accurately and immediately. A database that is designed primarily to support a company's day-to-day operations is classified as an **operational database**, also known as an **online transaction processing (OLTP) database**, **transactional database**, or **production database**. In contrast, an **analytical database** focuses primarily on storing historical data and business metrics used exclusively for tactical or strategic decision making. Such analysis typically requires extensive "data massaging" (data manipulation) to produce information on which to base pricing decisions, sales forecasts, market strategies, and so on. Analytical databases allow the end user to perform advanced analysis of business data using sophisticated tools.

Typically, analytical databases comprise two main components: a data warehouse and an online analytical processing front end. The **data warehouse** is a specialized database that stores data in a format optimized for decision support. The data warehouse contains historical data obtained from the operational databases as well as data from other external sources. **Online analytical processing (OLAP)** is a set of tools that work together to provide an advanced data analysis environment for retrieving, processing, and modeling data from the data warehouse. In recent times, this area of database application has grown in importance and usage, to the point that it has evolved into its own discipline: business intelligence. The term **business intelligence** describes a comprehensive approach to capture and process business data with the purpose of generating information to support business decision making. Chapter 13, Business Intelligence and Data Warehouses, covers this topic in detail.

Databases can also be classified to reflect the degree to which the data is structured. **Unstructured data** is data that exists in its original (raw) state—that is, in the format in which it was collected. Therefore, unstructured data exists in a format that does not lend itself to the processing that yields information. **Structured data** is the result of formatting unstructured data to facilitate storage, use, and generation of information. You apply structure (format) based on the type of processing that you intend to perform on the data. Some data might not be ready (unstructured) for some types of processing, but they might be ready (structured) for other types of processing. For example, the data value 37890 might refer to a zip code, a sales value, or a product code. If this value represents a zip code or a product code and is stored as text, you cannot perform mathematical computations with it. On the other hand, if this value represents a sales transaction, it must be formatted as numeric.

To further illustrate the concept of structure, imagine a stack of printed paper invoices. If you want to merely store these invoices as images for future retrieval and display, you can scan them and save them in a graphic format. On the other hand, if you want to derive information such as monthly totals and average sales, such graphic storage would not be useful. Instead, you could store the invoice data in a (structured) spreadsheet format so that you can perform the requisite computations. Actually, most data you encounter is best classified as semistructured. **Semistructured data** has already been processed to some extent. For example, if you look at a typical webpage, the data is presented in a prearranged format to convey some information. The database types mentioned thus far focus on the storage and management of highly structured data. However, corporations are not limited to the use of structured data.

operational database
A database designed primarily to support a company's day-to-day operations. Also known as a *transactional database*, *OLTP database*, or *production database*.

online transaction processing (OLTP) database
See operational database.

transactional database
See operational database.

production database
See operational database.

analytical database
A database focused primarily on storing historical data and business metrics used for tactical or strategic decision making.

data warehouse
A specialized database that stores historical and aggregated data in a format optimized for decision support.

online analytical processing (OLAP)
A set of tools that provide advanced data analysis for retrieving, processing, and modeling data from the data warehouse.

business intelligence
A set of tools and processes used to capture, collect, integrate, store, and analyze data to support business decision making.

unstructured data
Data that exists in its original, raw state; that is, in the format in which it was collected.

structured data
Data that has been formatted to facilitate storage, use, and information generation.

semistructured data
Data that has already been processed to some extent.

They also use semistructured and unstructured data. Just think of the valuable information that can be found on company emails, memos, and documents such as procedures, rules, and webpages. Unstructured and semistructured data storage and management needs are being addressed through a new generation of databases known as XML databases. **Extensible Markup Language (XML)** is a special language used to represent and manipulate data elements in a textual format. An **XML database** supports the storage and management of semistructured XML data.

Table 1.1 compares the features of several well-known database management systems.

Extensible Markup Language (XML)
A metalanguage used to represent and manipulate data elements. Unlike other markup languages, XML permits the manipulation of a document's data elements.

TABLE 1.1

TYPES OF DATABASES

PRODUCT	NUMBER OF USERS			DATA LOCATION		DATA USAGE		XML
	SINGLE USER	MULTIUSER		CENTRALIZED	DISTRIBUTED	OPERATIONAL	ANALYTICAL	
		WORKGROUP	ENTERPRISE					
MS Access	X	X		X		X		
MS SQL Server	X*	X	X	X	X	X	X	X
IBM DB2	X*	X	X	X	X	X	X	X
MySQL	X	X	X	X	X	X	X	X
Oracle RDBMS	X*	X	X	X	X	X	X	X

* Vendor offers single-user/personal or Express DBMS versions

With the emergence of the web and Internet-based technologies as the basis for the new "social media" generation, great amounts of data are being stored and analyzed. **Social media** refers to web and mobile technologies that enable "anywhere, anytime, always on" human interactions. Websites such as Google, Facebook, Twitter, and LinkedIn capture vast amounts of data about end users and consumers. This data grows exponentially and requires the use of specialized database systems. For example, as of 2017, over 648 million tweets were posted every day on Twitter, and that number continues to grow. As a result, the MySQL database Twitter was using to store user content was frequently overloaded by demand.[2] Facebook faces similar challenges. With over 500 terabytes of data coming in each day, it stores over 100 petabytes of data in a single data storage file system. From this data, its database scans over 200 terabytes of data each hour to process user actions, including status updates, picture requests, and billions of "Like" actions.[3] Over the past few years, this new breed of specialized database has grown in sophistication and widespread usage. Currently, this new type of database is known as a NoSQL database. The term **NoSQL** (Not only SQL) is generally used to describe a new generation of DBMS that is not based on the traditional relational database model. NoSQL databases are designed to handle the unprecedented volume of data, variety of data types and structures, and velocity of data operations that are characteristic of these new business requirements. You will learn more about this type of system in Chapter 2, Data Models.

This section briefly mentioned the many different types of databases. As you learned earlier, a database is a computer structure that houses and manages end-user data. One of the first tasks of a database professional is to ensure that end-user data is properly structured to derive valid and timely information. For this, good database design is essential.

XML database
A database system that stores and manages semistructured XML data.

social media
Web and mobile technologies that enable "anywhere, anytime, always on" human interactions.

NoSQL
A new generation of DBMS that is not based on the traditional relational database model.

[2] www.internetlivestats.com/twitter-statistics/

[3] Josh Constine, "How big is Facebook's data? 2.5 billion pieces of content and 500+ terabytes of data ingested every day," *Tech Crunch,* August 22, 2012, http://techcrunch.com/2012/08/22/how-big-is-facebooks-data-2-5-billion-pieces-of-content-and-500-terabytes-ingested-every-day/

1-4 Why Database Design Is Important

A problem that has evolved with the use of personal productivity tools such as spreadsheets and desktop database programs is that users typically lack proper data-modeling and database design skills. People naturally have a "narrow" view of the data in their environment. For example, consider a student's class schedule. The schedule probably contains the student's identification number and name, class code, class description, class credit hours, class instructor name, class meeting days and times, and class room number. In the mind of the student, these various data items compose a single unit. If a student organization wanted to keep a record of the schedules of its members, an end user might make a spreadsheet to store the schedule information. Even if the student makes a foray into the realm of desktop databases, he or she is likely to create a structure composed of a single table that mimics his or her view of the schedule data. As you will learn in the coming chapters, translating this type of narrow view of data into a single two-dimensional table structure is a poor database design choice.

Database design refers to the activities that focus on the design of the database structure that will be used to store and manage end-user data. A database that meets all user requirements does not just happen; its structure must be designed carefully. In fact, database design is such a crucial aspect of working with databases that most of this book is dedicated to the development of good database design techniques. Even a good DBMS will perform poorly with a badly designed database.

Data is one of an organization's most valuable assets. Data on customers, employees, orders, and receipts is all vital to the existence of a company. Tracking key growth and performance indicators are also vital to strategic and tactical plans to ensure future success; therefore, an organization's data must not be handled lightly or carelessly. Thorough planning to ensure that data is properly used and leveraged to give the company the most benefit is just as important as proper financial planning to ensure that the company gets the best use from its financial resources.

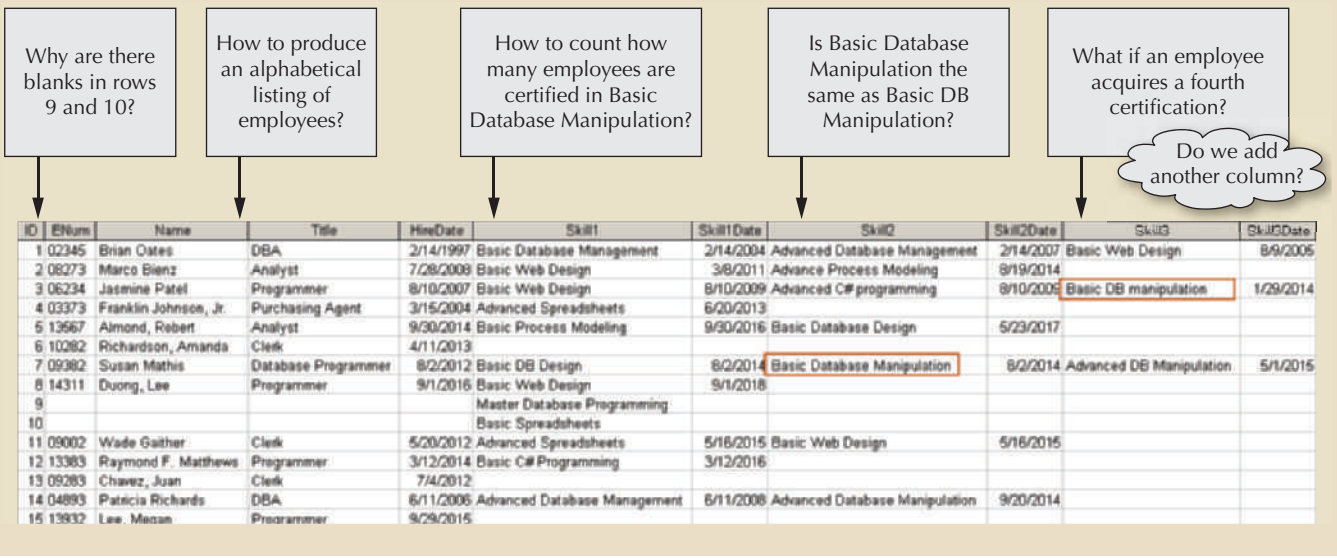
Because current-generation DBMSs are easy to use, an unfortunate side effect is that many computer-savvy business users gain a false sense of confidence in their ability to build a functional database. These users can effectively navigate the creation of database objects, but without the proper understanding of database design, they tend to produce flawed, overly simplified structures that prevent the system from correctly storing data that corresponds to business realities, which produces incomplete or erroneous results when the data is retrieved. Consider the data shown in Figure 1.5, which illustrates the efforts of an organization to keep records about its employees and their skills. Some employees have not passed a certification test in any skill, while others have been certified in several skills. Some certified skills are shared by several employees, while other skills have no employees that hold those certifications.

Based on this storage of the data, notice the following problems:

- It would be difficult, if not impossible, to produce an alphabetical listing of employees based on their last names.
- To determine how many employees are certified in Basic Database Manipulation, you would need a program that counts the number of those certifications recorded in Skill1 and places it in a variable. Then the count of those certifications in Skill2 could be calculated and added to the variable. Finally, the count of those certifications in Skill3 could be calculated and added to the variable to produce the total.
- If you redundantly store the name of a skill with each employee who is certified in that skill, you run the risk of spelling the name differently for different employees. For example, the skill *Basic Database Manipulation* is also entered as *Basic DB Manipulation* for at least one employee in Figure 1.5, which makes it difficult to get an accurate count of employees who have the certification.

database design
The process that yields the description of the database structure and determines the database components. The second phase of the database life cycle.

FIGURE 1.5 EMPLOYEE SKILLS CERTIFICATION IN A POOR DESIGN



ID	ENum	Name	Title	HireDate	Skill1	Skill1Date	Skill2	Skill2Date	Skill3	Skill3Date
1	02345	Brian Oates	DBA	2/14/1997	Basic Database Management	2/14/2004	Advanced Database Management	2/14/2007	Basic Web Design	8/9/2005
2	08273	Marco Bienz	Analyst	7/28/2008	Basic Web Design	3/8/2011	Advance Process Modeling	8/19/2014		
3	06234	Jasmine Patel	Programmer	8/10/2007	Basic Web Design	8/10/2009	Advanced C# programming	8/10/2009	Basic DB manipulation	1/29/2014
4	03373	Franklin Johnson, Jr.	Purchasing Agent	3/15/2004	Advanced Spreadsheets	6/20/2013				
5	13567	Almond, Robert	Analyst	9/30/2014	Basic Process Modeling	9/30/2016	Basic Database Design	5/23/2017		
6	10282	Richardson, Amanda	Clerk	4/11/2013						
7	09382	Susan Mathis	Database Programmer	8/2/2012	Basic DB Design	8/2/2014	Basic Database Manipulation	8/2/2014	Advanced DB Manipulation	5/1/2015
8	14311	Duong, Lee	Programmer	9/1/2016	Basic Web Design	9/1/2018				
9					Master Database Programming					
10					Basic Spreadsheets					
11	09002	Wade Gaither	Clerk	5/20/2012	Advanced Spreadsheets	5/16/2015	Basic Web Design	5/16/2015		
12	13383	Raymond F. Matthews	Programmer	3/12/2014	Basic C# Programming	3/12/2016				
13	09283	Chavez, Juan	Clerk	7/4/2012						
14	04893	Patricia Richards	DBA	6/11/2006	Advanced Database Management	6/11/2008	Advanced Database Manipulation	9/20/2014		
15	13932	Lee, Megan	Programmer	9/29/2015						

- The structure of the database will have to be changed by adding more columns to the table when an employee is certified in a fourth skill. It will have to be modified again if an employee is certified in a fifth skill.

Contrast this poor design with that shown in Figure 1.6 where the design has been improved by decomposing the data into three related tables. These tables contain all of the same data that was represented in Figure 1.5, but the tables are structured so that you can easily manipulate the data to view it in different ways and answer simple questions.

With the improved structure in Figure 1.6, you can use simple commands in a standard data manipulation language to do the following:

- Produce an alphabetical listing of employees by last name:

  SELECT * FROM EMPLOYEE ORDER BY EMPLOYEE_LNAME;

- Determine how many employees are certified in Basic Database Manipulation:

- SELECT COUNT(*)
  FROM SKILL JOIN CERTIFIED ON SKILL.SKILL_ID = CERTIFIED.SKILL_ID
  WHERE SKILL_NAME = 'Basic Database Manipulation';

You will learn more about these commands in Chapter 7, Introduction to Structured Query Language (SQL).

Note that because each skill name is stored only once, the names cannot be spelled or abbreviated differently for different employees. Also, the additional certification of an employee with a fourth or fifth skill does not require changes to the structure of the tables.

Proper database design requires the designer to identify precisely the database's expected use. Designing a transactional database emphasizes accurate and consistent data and operational speed. Designing a data warehouse database emphasizes the use of historical and aggregated data. Designing a database to be used in a centralized, single-user environment requires a different approach from that used in the design of a distributed, multiuser database. This book emphasizes the design of transactional, centralized, single-user, and multiuser databases. Chapters 12 and 13 also examine critical issues confronting the designer of distributed and data warehouse databases.

FIGURE 1.6 EMPLOYEE SKILLS CERTIFICATION IN A GOOD DESIGN

Table name: EMPLOYEE

Employee_ID	Employee_FName	Employee_LName	Employee_HireDate	Employee_Title
02345	Brian	Oates	2/14/1999	DBA
03373	Franklin	Johnson	3/15/2006	Purchasing Agent
04893	Patricia	Richards	6/11/2008	DBA
06234	Jasmine	Patel	8/10/2009	Programmer
08273	Marco	Bienz	7/28/2010	Analyst
09002	Wade	Gaither	5/20/2014	Clerk
09283	Juan	Chavez	7/4/2014	Clerk
09382	Susan	Mathis	8/2/2014	Database Programmer
10282	Amanda	Richardson	4/11/2015	Clerk
13383	Raymond	Matthews	3/12/2016	Programmer
13567	Robert	Almond	9/30/2016	Analyst
13932	Megan	Lee	9/29/2017	Programmer
14311	Lee	Duong	9/1/2018	Programmer

Database name: Ch01_Text

Table name: CERTIFIED

Employee_ID	Skill_ID	Certified_Date
02345	100	2/14/2004
02345	110	8/9/2005
02345	180	2/14/2007
03373	120	6/20/2013
04893	180	6/11/2008
04893	220	9/20/2014
06234	110	8/10/2009
06234	200	8/10/2009
06234	210	1/29/2014
08273	110	3/8/2011
08273	190	8/19/2014
09002	110	5/16/2015
09002	120	5/16/2015
09382	140	8/2/2014
09382	210	8/2/2014
09382	220	5/1/2015
13383	170	3/12/2016
13567	130	9/30/2016
13567	140	5/23/2017
14311	110	9/1/2018

Table name: SKILL

Skill_ID	Skill_Name	Skill_Description
100	Basic Database Management	Create and manage database user accounts.
110	Basic Web Design	Create and maintain HTML and CSS documents.
120	Advanced Spreadsheets	Use of advanced functions, user-defined functions, and macroing.
130	Basic Process Modeling	Create core business process models using standard libraries.
140	Basic Database Design	Create simple data models.
150	Master Database Programming	Create integrated trigger and procedure packages for a distributed environment.
160	Basic Spreadsheets	Create single tab worksheets with basic formulas.
170	Basic C# Programming	Create single-tier data aware modules.
180	Advanced Database Management	Manage Database Server Clusters.
190	Advance Process Modeling	Evaluate and Redesign cross-functional internal and external business processes.
200	Advanced C# Programming	Create multi-tier applications using multi-threading.
210	Basic Database Manipulation	Create simple data retrieval and manipulation statements in SQL.
220	Advanced Database Manipulation	Use of advanced data manipulation methods for multi-table inserts, set operations, and correlated subqueries.

Designing appropriate data repositories of integrated information using the two-dimensional table structures found in most databases is a process of decomposition. The integrated data must be decomposed properly into its constituent parts, with each part stored in its own table. Further, the relationships between these tables must be carefully considered and implemented so the integrated view of the data can be recreated later as information for the end user. A well-designed database facilitates data management and generates accurate and valuable information. A poorly designed database is likely to become a breeding ground for difficult-to-trace errors that may lead to poor decision making—and poor decision making can lead to the failure of an organization. Database design is simply too important to be left to luck. That's why college students study database design, why organizations of all types and sizes send personnel to database design seminars, and why database design consultants often make an excellent living.

1-5 Evolution of File System Data Processing

Understanding what a database is, what it does, and the proper way to use it can be clarified by considering what a database is not. A brief explanation of the evolution of file system data processing can be helpful in understanding the data access limitations that databases attempt to overcome. Understanding these limitations is relevant to database designers and developers because database technologies do not make these problems magically disappear—database technologies simply make it easier to create solutions that avoid these problems. Creating database designs that avoid the pitfalls of earlier systems requires that the designer understand these problems and how to avoid them; otherwise, the database technologies are no better (and are potentially even worse!) than the technologies and techniques they have replaced.

1-5a Manual File Systems

To be successful, an organization must develop systems for handling core business tasks. Historically, such systems were often manual, paper-and-pencil systems. The papers within these systems were organized to facilitate the expected use of the data. Typically, this was accomplished through a system of file folders and filing cabinets. As long as a collection of data was relatively small and an organization's business users had few reporting requirements, the manual system served its role well as a data repository. However, as organizations grew and as reporting requirements became more complex, keeping track of data in a manual file system became more difficult. Therefore, companies looked to computer technology for help.

1-5b Computerized File Systems

Generating reports from manual file systems was slow and cumbersome. In fact, some business managers faced government-imposed reporting requirements that led to weeks of intensive effort each quarter, even when a well-designed manual system was used. Therefore, a **data processing (DP) specialist** was hired to create a computer-based system that would track data and produce required reports.

data processing (DP) specialist
The person responsible for developing and managing a computerized file processing system.

Initially, the computer files within the file system were similar to the manual files. A simple example of a customer data file for a small insurance company is shown in Figure 1.7. (You will discover later that the file structure shown in Figure 1.7, although typically found in early file systems, is unsatisfactory for a database.)

FIGURE 1.7 CONTENTS OF THE CUSTOMER FILE

Database name: Ch01_Text

C_NAME	C_PHONE	C_ADDRESS	C_ZIP	A_NAME	A_PHONE	TP	AMT	REN
Alfred A. Ramas	615-844-2573	218 Fork Rd., Babs, TN	36123	Leah F. Hahn	615-882-1244	T1	100.00	05-Apr-2018
Leona K. Dunne	713-894-1238	Box 12A, Fox, KY	25246	Alex B. Alby	713-228-1249	T1	250.00	16-Jun-2018
Kathy W. Smith	615-894-2285	125 Oak Ln, Babs, TN	36123	Leah F. Hahn	615-882-2144	S2	150.00	29-Jan-2019
Paul F. Olowski	615-894-2180	217 Lee Ln., Babs, TN	36123	Leah F. Hahn	615-882-1244	S1	300.00	14-Oct-2018
Myron Orlando	615-222-1672	Box 111, New, TN	36155	Alex B. Alby	713-228-1249	T1	100.00	28-Dec-2018
Amy B. O'Brian	713-442-3381	387 Troll Dr., Fox, KY	25246	John T. Okon	615-123-5589	T2	850.00	22-Sep-2018
James G. Brown	615-297-1228	21 Tye Rd., Nash, TN	37118	Leah F. Hahn	615-882-1244	S1	120.00	25-Mar-2019
George Williams	615-290-2556	155 Maple, Nash, TN	37119	John T. Okon	615-123-5589	S1	250.00	17-Jul-2018
Anne G. Farriss	713-382-7185	2119 Elm, Crew, KY	25432	Alex B. Alby	713-228-1249	T2	100.00	03-Dec-2018
Olette K. Smith	615-297-3809	2782 Main, Nash, TN	37118	John T. Okon	615-123-5589	S2	500.00	14-Mar-2019

C_NAME = Customer name
C_PHONE = Customer phone
C_ADDRESS = Customer address
C_ZIP = Customer zip code

A_NAME = Agent name
A_PHONE = Agent phone
TP = Insurance type
AMT = Insurance policy amount, in thousands of $
REN = Insurance renewal date

The description of computer files requires a specialized vocabulary. Every discipline develops its own terminology to enable its practitioners to communicate clearly. The basic file vocabulary shown in Table 1.2 will help you to understand subsequent discussions more easily.

TABLE 1.2

BASIC FILE TERMINOLOGY

TERM	DEFINITION
Data	Raw facts, such as a telephone number, a birth date, a customer name, and a year-to-date (YTD) sales value. Data has little meaning unless it has been organized in some logical manner.
Field	A character or group of characters (alphabetic or numeric) that has a specific meaning. A field is used to define and store data.
Record	A logically connected set of one or more fields that describes a person, place, or thing. For example, the fields that constitute a record for a customer might consist of the customer's name, address, phone number, date of birth, credit limit, and unpaid balance.
File	A collection of related records. For example, a file might contain data about the students currently enrolled at Gigantic University.

Online Content

The databases used in each chapter are available at *www.cengagebrain.com*. Throughout the book, Online Content boxes highlight material related to chapter content on the website.

Using the proper file terminology in Table 1.2, you can identify the file components shown in Figure 1.7. The CUSTOMER file contains 10 records. Each record is composed of 9 fields: C_NAME, C_PHONE, C_ADDRESS, C_ZIP, A_NAME, A_PHONE, TP, AMT, and REN. The 10 records are stored in a named file. Because the file in Figure 1.7 contains customer data for the insurance company, its filename is CUSTOMER.

When business users wanted data from the computerized file, they sent requests for the data to the DP specialist. For each request, the DP specialist had to create programs to retrieve the data from the file, manipulate it in whatever manner the user had requested, and present it as a printed report. If a request was for a report that had been run previously, the DP specialist could rerun the existing program and provide the printed results to the user. As other business users saw the new and innovative ways in which customer data was being reported, they wanted to be able to view their data in similar fashions. This generated more requests for the DP specialist to create more computerized files of other business data, which in turn meant that more data management programs had to be created, which led to even more requests for reports. For example, the sales department at the insurance company created a file named SALES, which helped track daily sales efforts. The sales department's success was so obvious that the personnel department manager demanded access to the DP specialist to automate payroll processing and other personnel functions. Consequently, the DP specialist was asked to create the AGENT file shown in Figure 1.8. The data in the AGENT file was used to write checks, keep track of taxes paid, and summarize insurance coverage, among other tasks.

As more and more computerized files were developed, the problems with this type of file system became apparent. While these problems are explored in detail in the next section, the problems basically centered on having many data files that contained related—often overlapping—data with no means of controlling or managing the data consistently across all of the files. As shown in Figure 1.9, each file in the system used its own application program to store, retrieve, and modify data. Also, each file was owned by the individual or the department that commissioned its creation.

The advent of computer files to store company data was significant; it not only established a landmark in the use of computer technologies but also represented a huge step

field
A character or group of characters (alphabetic or numeric) that has a specific meaning. A field is used to define and store data.

record
A logically connected set of one or more fields that describes a person, place, or thing.

file
A collection of related records. For example, a file might contain data about the students currently enrolled at Gigantic University.

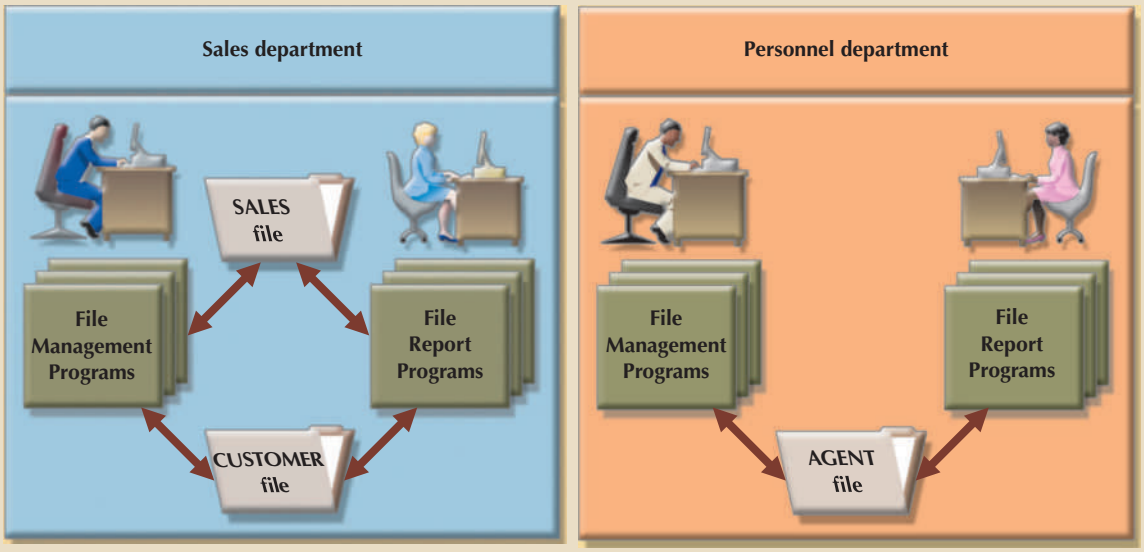
FIGURE 1.8 CONTENTS OF THE AGENT FILE

Database name: Ch01_Text

A_NAME	A_PHONE	A_ADDRESS	ZIP	HIRED	YTD_PAY	YTD_FIT	YTD_FICA	YTD_SLS	DEP
Alex B. Alby	713-228-1249	123 Toll, Nash, TN	37119	01-Nov-2000	26566.24	6641.56	2125.30	132737.75	3
Leah F. Hahn	615-882-1244	334 Main, Fox, KY	25246	23-May-1986	32213.78	8053.44	2577.10	138967.35	0
John T. Okon	615-123-5589	452 Elm, New, TN	36155	15-Jun-2005	23198.29	5799.57	1855.86	127093.45	2

A_NAME = Agent name
A_PHONE = Agent phone
A_ADDRESS = Agent address
ZIP = Agent zip code
HIRED = Agent date of hire
YTD_PAY = Year-to-date pay
YTD_FIT = Year-to-date federal income tax paid
YTD_FICA = Year-to-date Social Security taxes paid
YTD_SLS = Year-to-date sales
DEP = Number of dependents

FIGURE 1.9 A SIMPLE FILE SYSTEM



forward in a business's ability to process data. Previously, users had direct, hands-on access to all of the business data. But they didn't have the tools to convert that data into the information they needed. The creation of computerized file systems gave them improved tools for manipulating the company data that allowed them to create new information. However, it had the additional effect of introducing a schism between the end users and their data. The desire to close the gap between the end users and the data influenced the development of many types of computer technologies, system designs, and uses (and misuses) of many technologies and techniques. However, such developments also created a split between the ways DP specialists and end users viewed the data.

- From the DP specialist's perspective, the computer files within the file system were created to be similar to the manual files. Data management programs were created to add to, update, and delete data from the file.
- From the end user's perspective, the systems separated the users from the data. As the users' competitive environment pushed them to make more and more decisions in less time, users became frustrated by the delay between conceiving of a new way to create information from the data and the point when the DP specialist actually created the programs to generate that information.

1-5c File System Redux: Modern End-User Productivity Tools

The users' desire for direct, hands-on access to data helped to fuel the adoption of personal computers for business use. Although not directly related to file system evolution, the ubiquitous use of personal productivity tools can introduce the same problems as the old file systems.

Personal computer spreadsheet programs such as Microsoft Excel are widely used by business users, and they allow the user to enter data in a series of rows and columns so the data can be manipulated using a wide range of functions. The popularity of spreadsheet applications has enabled users to conduct sophisticated data analysis that has greatly enhanced their ability to understand the data and make better decisions. Unfortunately, as in the old adage "When the only tool you have is a hammer, every problem looks like a nail," users have become so adept at working with spreadsheets that they tend to use them to complete tasks for which spreadsheets are not appropriate.

A common misuse of spreadsheets is as a substitute for a database. Interestingly, end users often take the limited data to which they have direct access and place it in a spreadsheet format similar to that of the traditional, manual data storage systems—which is precisely what the early DP specialists did when creating computerized data files. Due to the large number of users with spreadsheets, each making separate copies of the data, the resulting "file system" of spreadsheets suffers from the same problems as the file systems created by the early DP specialists, which are outlined in the next section.

1-6 Problems with File System Data Processing

The file system method of organizing and managing data was a definite improvement over the manual system, and the file system served a useful purpose in data management for over two decades—a very long time in the computer era. Nonetheless, many problems and limitations became evident in this approach. A critique of the file system method serves two major purposes:

- Understanding the shortcomings of the file system enables you to understand the development of modern databases.
- Failure to understand such problems is likely to lead to their duplication in a database environment, even though database technology makes it easy to avoid them.

The following problems associated with file systems, whether created by DP specialists or through a series of spreadsheets, severely challenge the types of information that can be created from the data as well as the accuracy of the information:

- *Lengthy development times*. The first and most glaring problem with the file system approach is that even the simplest data-retrieval task requires extensive programming. With the older file systems, programmers had to specify what must be done and how to do it. As you will learn in upcoming chapters, modern databases use a nonprocedural data manipulation language that allows the user to specify what must be done without specifying how.
- *Difficulty of getting quick answers*. The need to write programs to produce even the simplest reports makes ad hoc queries impossible. Harried DP specialists who worked with mature file systems often received numerous requests for new reports. They were often forced to say that the report will be ready "next week" or even "next month." If you need the information now, getting it next week or next month will not serve your information needs.
- *Complex system administration*. System administration becomes more difficult as the number of files in the system expands. Even a simple file system with a few files requires creating and maintaining several file management programs. Each file must

have its own file management programs that allow the user to add, modify, and delete records; to list the file contents; and to generate reports. Because ad hoc queries are not possible, the file reporting programs can multiply quickly. The problem is compounded by the fact that each department in the organization "owns" its data by creating its own files.

- *Lack of security and limited data sharing*. Another fault of a file system data repository is a lack of security and limited data sharing. Data sharing and security are closely related. Sharing data among multiple geographically dispersed users introduces a lot of security risks. In terms of spreadsheet data, while many spreadsheet programs provide rudimentary security options, they are not always used, and even when they are, they are insufficient for robust data sharing among users. In terms of creating data management and reporting programs, security and data-sharing features are difficult to program and consequently are often omitted from a file system environment. Such features include effective password protection, the ability to lock out parts of files or parts of the system itself, and other measures designed to safeguard data confidentiality. Even when an attempt is made to improve system and data security, the security devices tend to be limited in scope and effectiveness.
- *Extensive programming*. Making changes to an existing file structure can be difficult in a file system environment. For example, changing just one field in the original CUSTOMER file would require a program that:
  1. reads a record from the original file
  2. transforms the original data to conform to the new structure's storage requirements
  3. writes the transformed data into the new file structure
  4. repeats the preceding steps for each record in the original file.

In fact, any change to a file structure, no matter how minor, forces modifications in all of the programs that use the data in that file. Modifications are likely to produce errors (bugs), and additional time is spent using a debugging process to find those errors. Those limitations, in turn, lead to problems of structural and data dependence.

1-6a Structural and Data Dependence

A file system exhibits **structural dependence**, which means that access to a file is dependent on its structure. For example, adding a customer date-of-birth field to the CUSTOMER file shown in Figure 1.7 would require the four steps described in the previous section. Given this change, none of the previous programs will work with the new CUSTOMER file structure. Therefore, all of the file system programs must be modified to conform to the new file structure. In short, because the file system application programs are affected by changes in the file structure, they exhibit structural dependence. Conversely, **structural independence** exists when you can change the file structure without affecting the application's ability to access the data.

Even changes in the characteristics of data, such as changing a field from integer to decimal, require changes in all the programs that access the file. Because all data access programs are subject to change when any of the file's data storage characteristics change (that is, changing the **data type**), the file system is said to exhibit **data dependence**. Conversely, **data independence** exists when you can change the data storage characteristics without affecting the program's ability to access the data.

The practical significance of data dependence is the difference between the **logical data format** (how the human being views the data) and the **physical data format** (how the computer must work with the data). Any program that accesses a file system's file must tell the computer not only what to do but also how to do it. Consequently, each

structural dependence
A data characteristic in which a change in the database schema affects data access, thus requiring changes in all access programs.

structural independence
A data characteristic in which changes in the database schema do not affect data access.

data type
Defines the kind of values that can be used or stored. Also, used in programming languages and database systems to determine the operations that can be applied to such data.

data dependence
A data condition in which data representation and manipulation are dependent on the physical data storage characteristics.

data independence
A condition in which data access is unaffected by changes in the physical data storage characteristics.

logical data format
The way a person views data within the context of a problem domain.

physical data format
The way a computer "sees" (stores) data.

program must contain lines that specify the opening of a specific file type, its record specification, and its field definitions. Data dependence makes the file system extremely cumbersome from the point of view of a programmer and database manager.

1-6b Data Redundancy

The file system's structure makes it difficult to combine data from multiple sources, and its lack of security renders the file system vulnerable to security breaches. The organizational structure promotes the storage of the same basic data in different locations. (Database professionals use the term **islands of information** for such scattered data locations.) The dispersion of data is exacerbated by the use of spreadsheets to store data. In a file system, the entire sales department would share access to the SALES data file through the data management and reporting programs created by the DP specialist. With the use of spreadsheets, each member of the sales department can create his or her own copy of the sales data. Because data stored in different locations will probably not be updated consistently, the islands of information often contain different versions of the same data. For example, in Figures 1.7 and 1.8, the agent names and phone numbers occur in both the CUSTOMER and the AGENT files. You only need one correct copy of the agent names and phone numbers. Having them occur in more than one place produces data redundancy. **Data redundancy** exists when the same data is stored unnecessarily at different places.

Uncontrolled data redundancy sets the stage for the following:

- *Poor data security*. Having multiple copies of data increases the chances for a copy of the data to be susceptible to unauthorized access. Chapter 16, Database Administration and Security, explores the issues and techniques associated with securing data.
- *Data inconsistency*. Data inconsistency exists when different and conflicting versions of the same data appear in different places. For example, suppose you change an agent's phone number in the AGENT file. If you forget to make the corresponding change in the CUSTOMER file, the files contain different data for the same agent. Reports will yield inconsistent results that depend on which version of the data is used.
- *Data-entry errors*. Data-entry errors are more likely to occur when complex entries (such as 10-digit phone numbers) are made in several different files or recur frequently in one or more files. In fact, the CUSTOMER file shown in Figure 1.7 contains just such an entry error: the third record in the CUSTOMER file has transposed digits in the agent's phone number (615-882-2144 rather than 615-882-1244).
- *Data integrity problems*. It is possible to enter a nonexistent sales agent's name and phone number into the CUSTOMER file, but customers are not likely to be impressed if the insurance agency supplies the name and phone number of an agent who does not exist. Should the personnel manager allow a nonexistent agent to accrue bonuses and benefits? In fact, a data-entry error such as an incorrectly spelled name or an incorrect phone number yields the same kind of data integrity problems.

islands of information
In the old file system environment, pools of independent, often duplicated, and inconsistent data created and managed by different departments.

data redundancy
Exists when the same data is stored unnecessarily at different places.

data integrity
In a relational database, a condition in which the data in the database complies with all entity and referential integrity constraints.

Note

Data that displays data inconsistency is also referred to as data that lacks data integrity. **Data integrity** is defined as the condition in which all of the data in the database is consistent with the real-world events and conditions. In other words, data integrity means the following:

- Data is *accurate*—there are no data inconsistencies.
- Data is *verifiable*—the data will always yield consistent results.

1-6c Data Anomalies

The dictionary defines *anomaly* as "an abnormality." Ideally, a field value change should be made in only a single place. Data redundancy, however, fosters an abnormal condition by forcing field value changes in many different locations. Look at the CUSTOMER file in Figure 1.7. If agent Leah F. Hahn decides to get married and move, the agent's name, address, and phone number are likely to change. Instead of making these changes in a single file (AGENT), you must also make the change each time that agent's name and phone number occur in the CUSTOMER file. You could be faced with the prospect of making hundreds of corrections, one for each of the customers served by that agent! The same problem occurs when an agent decides to quit. Each customer served by that agent must be assigned a new agent. Any change in any field value must be correctly made in many places to maintain data integrity. A **data anomaly** develops when not all of the required changes in the redundant data are made successfully. The data anomalies found in Figure 1.7 are commonly defined as follows:

- *Update anomalies*. If agent Leah F. Hahn has a new phone number, it must be entered in each of the CUSTOMER file records in which Ms. Hahn's phone number is shown. In this case, only four changes must be made. In a large file system, such a change might occur in hundreds or even thousands of records. Clearly, the potential for data inconsistencies is great.
- *Insertion anomalies*. If only the CUSTOMER file existed and you needed to add a new agent, you would also add a dummy customer data entry to reflect the new agent's addition. Again, the potential for creating data inconsistencies would be great.
- *Deletion anomalies*. If you delete the customers Amy B. O'Brian, George Williams, and Olette K. Smith, you will also delete John T. Okon's agent data. Clearly, this is not desirable.

On a positive note, however, this book will help you develop the skills needed to design and model a successful database that avoids the problems listed in this section.

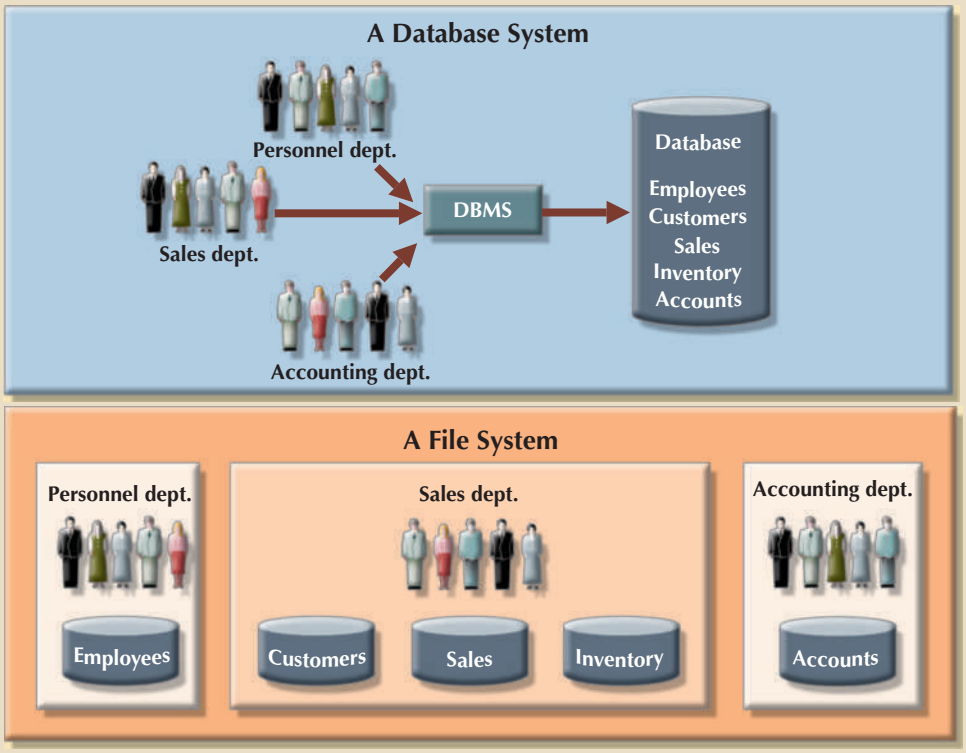
1-7 Database Systems

The problems inherent in file systems make using a database system very desirable. Unlike the file system, with its many separate and unrelated files, the database system consists of logically related data stored in a single logical data repository. (The "logical" label reflects the fact that the data repository appears to be a single unit to the end user, even though data might be physically distributed among multiple storage facilities and locations.) Because the database's data repository is a single logical unit, the database represents a major change in the way end-user data is stored, accessed, and managed. The database's DBMS, shown in Figure 1.10, provides numerous advantages over file system management, shown in Figure 1.9, by making it possible to eliminate most of the file system's data inconsistency, data anomaly, data dependence, and structural dependence problems. Better yet, the current generation of DBMS software stores not only the data structures but also the relationships between those structures and the access paths to those structures—all in a central location. The current generation of DBMS software also takes care of defining, storing, and managing all required access paths to those components.

Remember that the DBMS is just one of several crucial components of a database system. The DBMS may even be referred to as the database system's heart. However, just as it takes more than a heart to make a human being function, it takes more than a

data anomaly
A data abnormality in which inconsistent changes have been made to a database. For example, an employee moves, but the address change is not corrected in all files in the database.

FIGURE 1.10 CONTRASTING DATABASE AND FILE SYSTEMS



DBMS to make a database system function. In the sections that follow, you'll learn what a database system is, what its components are, and how the DBMS fits into the picture.

1-7a The Database System Environment

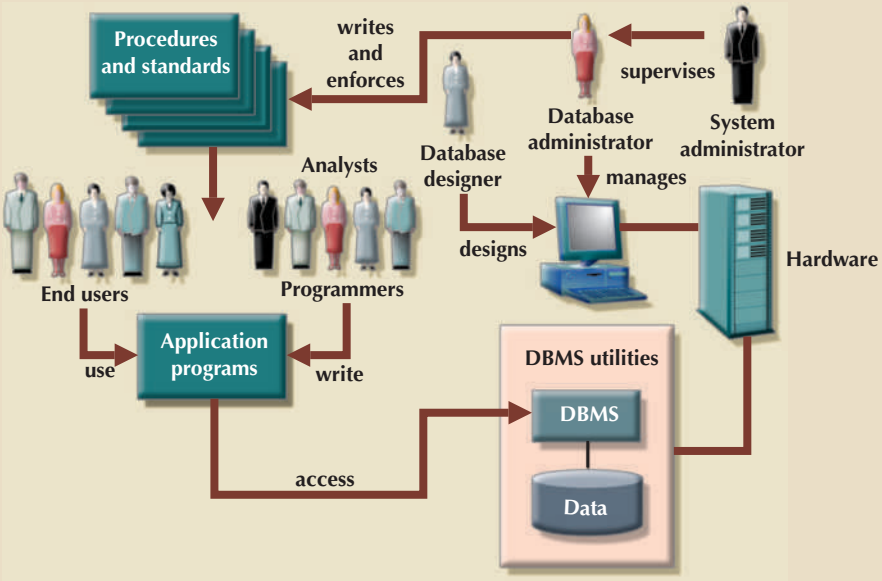
The term **database system** refers to an organization of components that define and regulate the collection, storage, management, and use of data within a database environment. From a general management point of view, the database system is composed of the five major parts shown in Figure 1.11: hardware, software, people, procedures, and data.

Let's take a closer look at the five components shown in Figure 1.11:

- *Hardware*. Hardware refers to all of the system's physical devices, including computers (PCs, tablets, workstations, servers, and supercomputers), storage devices, printers, network devices (hubs, switches, routers, fiber optics), and other devices (automated teller machines, ID readers, and so on).
- *Software*. Although the most readily identified software is the DBMS itself, three types of software are needed to make the database system function fully: operating system software, DBMS software, and application programs and utilities.
  - *Operating system software* manages all hardware components and makes it possible for all other software to run on the computers. Examples of operating system software are Microsoft Windows, Linux, Mac OS, UNIX, and MVS.
  - *DBMS software* manages the database within the database system. Some examples of DBMS software are Microsoft's SQL Server, Oracle Corporation's Oracle, Oracle's MySQL, and IBM's DB2.

database system
An organization of components that defines and regulates the collection, storage, management, and use of data in a database environment.

FIGURE 1.11 THE DATABASE SYSTEM ENVIRONMENT



  - *Application programs and utility software* are used to access and manipulate data in the DBMS and to manage the computer environment in which data access and manipulation take place. Application programs are most commonly used to access data within the database to generate reports, tabulations, and other information to facilitate decision making. Utilities are the software tools used to help manage the database system's computer components. For example, all of the major DBMS vendors now provide graphical user interfaces (GUIs) to help create database structures, control database access, and monitor database operations.

- *People*. This component includes all users of the database system. On the basis of primary job functions, five types of users can be identified in a database system: system administrators, database administrators, database designers, system analysts and programmers, and end users. Each user type, described next, performs both unique and complementary functions.
  - *System administrators* oversee the database system's general operations.
  - *Database administrators*, also known as DBAs, manage the DBMS and ensure that the database is functioning properly. The DBA's role is sufficiently important to warrant a detailed exploration in Chapter 16, Database Administration and Security.
  - *Database designers* design the database structure. They are, in effect, the database architects. If the database design is poor, even the best application programmers and the most dedicated DBAs cannot produce a useful database environment. Because organizations strive to optimize their data resources, the database designer's job description has expanded to cover new dimensions and growing responsibilities.
  - *System analysts and programmers* design and implement the application programs. They design and create the data-entry screens, reports, and procedures through which end users access and manipulate the database's data.
  - *End users* are the people who use the application programs to run the organization's daily operations. For example, sales clerks, supervisors, managers, and directors are all classified as end users. High-level end users employ the information obtained from the database to make tactical and strategic business decisions.

- *Procedures*. Procedures are the instructions and rules that govern the design and use of the database system. Procedures are a critical, although occasionally forgotten, component of the system. Procedures play an important role in a company because they enforce the standards by which business is conducted within the organization and with customers. Procedures also help to ensure that companies have an organized way to monitor and audit the data that enter the database and the information generated from those data.
- *Data*. The word *data* covers the collection of facts stored in the database. Because data is the raw material from which *information* is generated, determining which data to enter into the database and how to organize that data is a vital part of the database designer's job.

A database system adds a new dimension to an organization's management structure. The complexity of this managerial structure depends on the organization's size, its functions, and its corporate culture. Therefore, database systems can be created and managed at different levels of complexity and with varying adherence to precise standards. For example, compare a local convenience store system with a national insurance claims system. The convenience store system may be managed by two people, the hardware used is probably a single computer, the procedures are probably simple, and the data volume tends to be low. The national insurance claims system is likely to have at least one systems administrator, several full-time DBAs, and many designers and programmers; the hardware probably includes several servers at multiple locations throughout the United States; the procedures are likely to be numerous, complex, and rigorous; and the data volume tends to be high.

In addition to the different levels of database system complexity, managers must also take another important fact into account: database solutions must be cost-effective as well as tactically and strategically effective. Producing a million-dollar solution to a thousand-dollar problem is hardly an example of good database system selection or of good database design and management. Finally, the database technology already in use is likely to affect the selection of a database system.

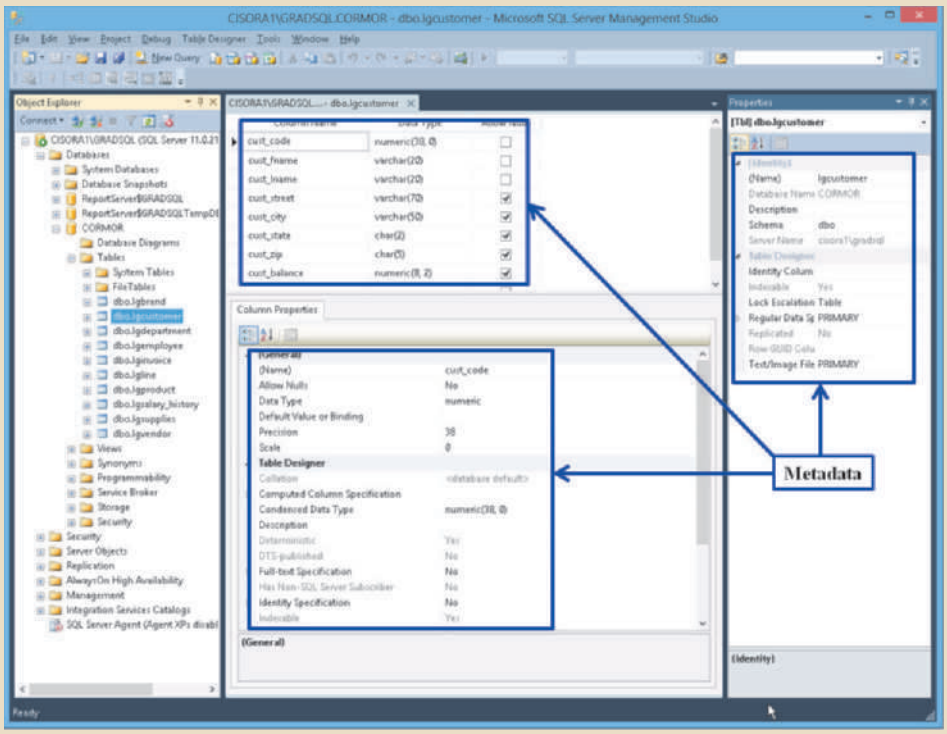
1-7b DBMS Functions

A DBMS performs several important functions that guarantee the integrity and consistency of the data in the database. Most of those functions are transparent to end users, and most can be achieved only through the use of a DBMS. They include data dictionary management, data storage management, data transformation and presentation, security management, multiuser access control, backup and recovery management, data integrity management, database access languages and application programming interfaces, and database communication interfaces. Each of these functions is explained as follows:

- *Data dictionary management*. The DBMS stores definitions of the data elements and their relationships (metadata) in a data dictionary. In turn, all programs that access the data in the database work through the DBMS. The DBMS uses the **data dictionary** to look up the required data component structures and relationships, thus relieving you from having to code such complex relationships in each program. Additionally, any changes made in a database structure are automatically recorded in the data dictionary, thereby freeing you from having to modify all of the programs that access the changed structure. In other words, the DBMS provides data abstraction, and it removes structural and data dependence from the system. For example, Figure 1.12 shows how Microsoft SQL Server Express presents the data definition for the CUSTOMER table.

data dictionary
A DBMS component that stores metadata—data about data. The data dictionary contains data definitions as well as data characteristics and relationships. May also include data that is external to the DBMS.

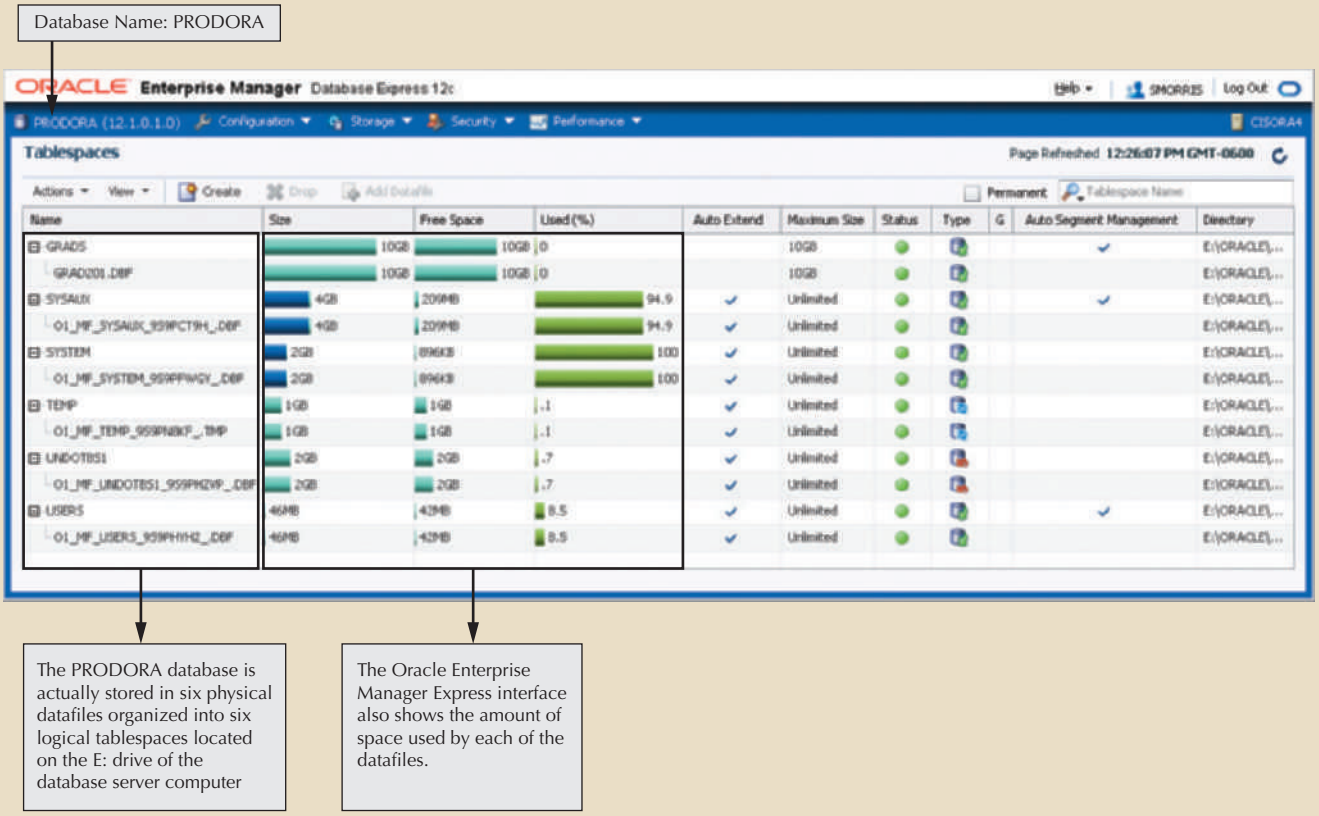
FIGURE 1.12 ILLUSTRATING METADATA WITH MICROSOFT SQL SERVER EXPRESS



- *Data storage management*. The DBMS creates and manages the complex structures required for data storage, thus relieving you from the difficult task of defining and programming the physical data characteristics. A modern DBMS provides storage not only for the data but also for related data-entry forms or screen definitions, report definitions, data validation rules, procedural code, structures to handle video and picture formats, and so on. Data storage management is also important for database performance tuning. **Performance tuning** relates to the activities that make the database perform more efficiently in terms of storage and access speed. Although the user sees the database as a single data storage unit, the DBMS actually stores the database in multiple physical data files (see Figure 1.13). Such data files may even be stored on different storage media. Therefore, the DBMS doesn't have to wait for one disk request to finish before the next one starts. In other words, the DBMS can fulfill database requests concurrently. Data storage management and performance tuning issues are addressed in Chapter 11, Database Performance Tuning and Query Optimization.
- *Data transformation and presentation*. The DBMS transforms entered data to conform to required data structures. The DBMS relieves you of the chore of distinguishing between the logical data format and the physical data format. That is, the DBMS formats the physically retrieved data to make it conform to the user's logical expectations. For example, imagine an enterprise database used by a multinational company. An end user in England would expect to enter the date July 11, 2017, as "11/07/2017." In contrast, the same date would be entered in the United States as "07/11/2017." Regardless of the data presentation format, the DBMS must manage the date in the proper format for each country.
- *Security management*. The DBMS creates a security system that enforces user security and data privacy. Security rules determine which users can access the

performance tuning
Activities that make a database perform more efficiently in terms of storage and access speed.

FIGURE 1.13 ILLUSTRATING DATA STORAGE MANAGEMENT WITH ORACLE



The Oracle Enterprise Manager Express GUI shows the data storage management characteristics for the PRODORA database.

database, which data items each user can access, and which data operations (read, add, delete, or modify) the user can perform. This is especially important in multiuser database systems. Chapter 16, Database Administration and Security, examines data security and privacy issues in greater detail. All database users may be authenticated to the DBMS through a username and password or through biometric authentication such as a fingerprint scan. The DBMS uses this information to assign access privileges to various database components such as queries and reports.

- *Multiuser access control*. To provide data integrity and data consistency, the DBMS uses sophisticated algorithms to ensure that multiple users can access the database concurrently without compromising its integrity. Chapter 10, Transaction Management and Concurrency Control, covers the details of multiuser access control.
- *Backup and recovery management*. The DBMS provides backup and data recovery to ensure data safety and integrity. Current DBMS systems provide special utilities that allow the DBA to perform routine and special backup and restore procedures. Recovery management deals with the recovery of the database after a failure, such as a bad sector in the disk or a power failure. Such capability is critical to preserving the database's integrity. Chapter 16 covers backup and recovery issues.
- *Data integrity management*. The DBMS promotes and enforces integrity rules, thus minimizing data redundancy and maximizing data consistency. The data

relationships stored in the data dictionary are used to enforce data integrity. Ensuring data integrity is especially important in transaction-oriented database systems. Data integrity and transaction management issues are addressed in Chapter 7, Introduction to Structured Query Language (SQL), and Chapter 10.

- *Database access languages and application programming interfaces*. The DBMS provides data access through a query language. A **query language** is a nonprocedural language—one that lets the user specify what must be done without having to specify how. **Structured Query Language (SQL)** is the de facto query language and data access standard supported by the majority of DBMS vendors. Chapter 7, Introduction to Structure Query Language (SQL), and Chapter 8, Advanced SQL, address the use of SQL. The DBMS also provides application programming interfaces to procedural languages such as COBOL, C, Java, Visual Basic.NET, and C#. In addition, the DBMS provides administrative utilities used by the DBA and the database designer to create, implement, monitor, and maintain the database.
- *Database communication interfaces*. A current-generation DBMS accepts end-user requests via multiple, different network environments. For example, the DBMS might provide access to the database via the Internet through the use of web browsers such as Mozilla Firefox, Google Chrome, Microsoft Edge, or Microsoft Internet Explorer. In this environment, communications can be accomplished in several ways:
  - End users can generate answers to queries by filling in screen forms through their preferred web browser.
  - The DBMS can automatically publish predefined reports on a website.
  - The DBMS can connect to third-party systems to distribute information via email or other productivity applications.

query language
A nonprocedural language that is used by a DBMS to manipulate its data. An example of a query language is SQL.

Structured Query Language (SQL)
A powerful and flexible relational database language composed of commands that enable users to create database and table structures, perform various types of data manipulation and data administration, and query the database to extract useful information.

Database communication interfaces are examined in greater detail in Chapter 12, Distributed Database Management Systems; in Chapter 15, Database Connectivity and Web Technologies; and in Appendix I, Databases in Electronic Commerce. (Appendixes are available at *www.cengagebrain.com*.)

Note

Why a Spreadsheet Is Not a Database

While a spreadsheet allows for the manipulation of data in a tabular format, it does not support even the most basic database functionality such as support for self-documentation through metadata, enforcement of data types or domains to ensure consistency of data within a column, defined relationships among tables, or constraints to ensure consistency of data across related tables. Most users lack the necessary training to recognize the limitations of spreadsheets for these types of tasks.

1-7c Managing the Database System: A Shift in Focus

The introduction of a database system over the file system provides a framework in which strict procedures and standards can be enforced. Consequently, the role of the human component changes from an emphasis on programming (in the file system) to a focus on the broader aspects of managing the organization's data resources and on the administration of the complex database software itself.

The database system makes it possible to tackle far more sophisticated uses of the data resources, as long as the database is designed to make use of that power. The

kinds of data structures created within the database and the extent of the relationships among them play a powerful role in determining the effectiveness of the database system.

Although the database system yields considerable advantages over previous data management approaches, database systems do carry significant disadvantages:

- *Increased costs*. Database systems require sophisticated hardware and software and highly skilled personnel. The cost of maintaining the hardware, software, and personnel required to operate and manage a database system can be substantial. Training, licensing, and regulation compliance costs are often overlooked when database systems are implemented.
- *Management complexity*. Database systems interface with many different technologies and have a significant impact on a company's resources and culture. The changes introduced by the adoption of a database system must be properly managed to ensure that they help advance the company's objectives. Because database systems hold crucial company data that are accessed from multiple sources, security issues must be assessed constantly.
- *Maintaining currency*. To maximize the efficiency of the database system, you must keep your system current. Therefore, you must perform frequent updates and apply the latest patches and security measures to all components. Because database technology advances rapidly, personnel training costs tend to be significant.
- *Vendor dependence*. Given the heavy investment in technology and personnel training, companies might be reluctant to change database vendors. As a consequence, vendors are less likely to offer pricing point advantages to existing customers, and those customers might be limited in their choice of database system components.
- *Frequent upgrade/replacement cycles*. DBMS vendors frequently upgrade their products by adding new functionality. Such new features often come bundled in new upgrade versions of the software. Some of these versions require hardware upgrades. Not only do the upgrades themselves cost money but it also costs money to train database users and administrators to properly use and manage the new features.

Now that you know what a database and DBMS are, and why they are necessary, you are ready to begin developing your career as a database professional.

1-8 Preparing for Your Database Professional Career

In this chapter, you were introduced to the concepts of data, information, databases, and DBMSs. You also learned that, regardless of what type of database you use (OLTP, OLAP, or NoSQL), or what type of database environment you are working in (e.g., Oracle, Microsoft, IBM, or Hadoop), the success of a database system greatly depends on how well the database structure is designed.

Throughout this book, you will learn the building blocks that lay the foundation for your career as a database professional. Understanding these building blocks and developing the skills to use them effectively will prepare you to work with databases at many different levels within an organization. A small sample of such career opportunities is shown in Table 1.3.

TABLE 1.3

DATABASE CAREER OPPORTUNITIES

JOB TITLE	DESCRIPTION	SAMPLE SKILLS REQUIRED
Database Developer	Create and maintain database-based applications	Programming, database fundamentals, SQL
Database Designer	Design and maintain databases	Systems design, database design, SQL
Database Administrator	Manage and maintain DBMS and databases	Database fundamentals, SQL, vendor courses
Database Analyst	Develop databases for decision support reporting	SQL, query optimization, data warehouses
Database Architect	Design and implementation of database environments (conceptual, logical, and physical)	DBMS fundamentals, data modeling, SQL, hardware knowledge, etc.
Database Consultant	Help companies leverage database technologies to improve business processes and achieve specific goals	Database fundamentals, data modeling, database design, SQL, DBMS, hardware, vendor-specific technologies, etc.
Database Security Officer	Implement security policies for data administration	DBMS fundamentals, database administration, SQL, data security technologies, etc.
Cloud Computing Data Architect	Design and implement the infrastructure for next-generation cloud database systems	Internet technologies, cloud storage technologies, data security, performance tuning, large databases, etc.
Data Scientist	Analyze large amounts of varied data to generate insights, relationships, and predictable behaviors	Data analysis, statistics, advanced mathematics, SQL, programming, data mining, machine learning, data visualization

As you also learned in this chapter, database technologies are constantly evolving to address new challenges such as large databases, semistructured and unstructured data, increasing processing speed, and lowering costs. While database technologies can change quickly, the fundamental concepts and skills do not. It is our goal that after you learn the database essentials in this book, you will be ready to apply your knowledge and skills to work with traditional OLTP and OLAP systems as well as cutting-edge, complex database technologies such as the following:

- *Very large databases (VLDB)*. Many vendors are addressing the need for databases that support large amounts of data, usually in the petabyte range. (A petabyte is more than 1,000 terabytes.) VLDB vendors include Oracle Exadata, IBM's Netezza, HP's Vertica, and Teradata. VLDBs are now being superceded by Big Data databases.
- *Big Data databases*. Products such as Cassandra (Facebook) and BigTable (Google) are using "columnar-database" technologies to support the needs of database applications that manage large amounts of "nontabular" data. See more about this topic in Chapter 2.
- *In-memory databases*. Most major database vendors also offer some type of in-memory database support to address the need for faster database processing. In-memory databases store most of their data in primary memory (RAM) rather than in slower secondary storage (hard disks). In-memory databases include IBM's solidDB and Oracle's TimesTen.
- *Cloud databases*. Companies can now use cloud database services to quickly add database systems to their environment while simultaneously lowering the total cost of ownership of a new DBMS. A cloud database offers all the advantages of a local DBMS, but instead of residing within your organization's network infrastructure, it resides on the Internet. See more about this topic in Chapter 15.

We address some of these topics in this book, but not all—no single book can cover the entire realm of database technologies. This book's primary focus is to help you learn database fundamentals, develop your database design skills, and master your SQL skills so you will have a head start in becoming a successful database professional. However, you first must learn about the tools at your disposal. In the next chapter, you will learn different approaches to data management and how these approaches influence your designs.

Summary

- Data consists of raw facts. Information is the result of processing data to reveal its meaning. Accurate, relevant, and timely information is the key to good decision making, and good decision making is the key to organizational survival in a global environment.
- Data is usually stored in a database. To implement a database and to manage its contents, you need a database management system (DBMS). The DBMS serves as the intermediary between the user and the database. The database contains the data you have collected and "data about data," known as metadata.
- Database design defines the database structure. A well-designed database facilitates data management and generates accurate and valuable information. A poorly designed database can lead to poor decision making, and poor decision making can lead to the failure of an organization.
- Databases can be classified according to the number of users supported, where the data is located, the type of data stored, the intended data usage, and the degree to which the data is structured.
- Databases evolved from manual and then computerized file systems. In a file system, data is stored in independent files, each requiring its own data management programs. Although this method of data management is largely outmoded, understanding its characteristics makes database design easier to comprehend.
- Some limitations of file system data management are that it requires extensive programming, system administration can be complex and difficult, making changes to existing structures is difficult, and security features are likely to be inadequate. Also, independent files tend to contain redundant data, leading to problems of structural and data dependence.
- DBMSs were developed to address the file system's inherent weaknesses. Rather than depositing data in independent files, a DBMS presents the database to the end user as a single data repository. This arrangement promotes data sharing, thus eliminating the potential problem of islands of information. In addition, the DBMS enforces data integrity, eliminates redundancy, and promotes data security.
- Knowledge of database technologies leads to many career opportunities in the ever-expanding IT industry. There is a variety of specialization within the database arena for a wide range of skills and expertise.

Key Terms

ad hoc query
analytical database
business intelligence
centralized database
cloud database
data
data anomaly
data dependence
data dictionary
data inconsistency
data independence
data integrity
data management
data processing (DP) specialist
data quality
data redundancy
data type
data warehouse
database
database design
database management system (DBMS)
database system
desktop database
discipline-specific database
distributed database
enterprise database
Extensible Markup Language (XML)
field
file
general-purpose database
information
islands of information
knowledge
logical data format
metadata
multiuser database
NoSQL
online analytical processing (OLAP)
online transaction processing (OLTP) database
operational database
performance tuning
physical data format
production database
query
query language
query result set
record
semistructured data
single-user database
social media
structural dependence
structural independence
structured data
Structured Query Language (SQL)
transactional database
unstructured data
workgroup database
XML database

Review Questions

1. Define each of the following terms:
   a. data
   b. field
   c. record
   d. file
2. What is data redundancy, and which characteristics of the file system can lead to it?
3. What is data independence, and why is it lacking in file systems?
4. What is a DBMS, and what are its functions?
5. What is structural independence, and why is it important?
6. Explain the differences among data, information, and a database.
7. What is the role of a DBMS, and what are its advantages? What are its disadvantages?

8. List and describe the different types of databases.
9. What are the main components of a database system?
10. What is metadata?
11. Explain why database design is important.
12. What are the potential costs of implementing a database system?
13. Use examples to compare and contrast unstructured and structured data. Which type is more prevalent in a typical business environment?
14. What are some basic database functions that a spreadsheet cannot perform?
15. What common problems do a collection of spreadsheets created by end users share with the typical file system?
16. Explain the significance of the loss of direct, hands-on access to business data that end users experienced with the advent of computerized data repositories.
17. Explain why the cost of ownership may be lower with a cloud database than with a traditional, company database.

Problems

Online Content

The file structures you see in this problem set are simulated in a Microsoft Access database named Ch01_Problems, which is available at *www.cengagebrain.com*.

Given the file structure shown in Figure P1.1, answer Problems 1–4.

FIGURE P1.1 THE FILE STRUCTURE FOR PROBLEMS 1–4

PROJECT_CODE	PROJECT_MANAGER	MANAGER_PHONE	MANAGER_ADDRESS	PROJECT_BID_PRICE
21-5Z	Holly B. Parker	904-338-3416	3334 Lee Rd., Gainesville, FL 37123	16833460.00
25-2D	Jane D. Grant	615-898-9909	218 Clark Blvd., Nashville, TN 38382	12500000.00
25-5A	George F. Dorts	615-227-1245	124 River Dr., Franklin, TN 29185	32512420.00
25-9T	Holly B. Parker	904-338-3416	3334 Lee Rd., Gainesville, FL 37123	21563234.00
27-4Q	George F. Dorts	615-227-1245	124 River Dr., Franklin, TN 29185	10314545.00
29-2D	Holly B. Parker	904-338-3416	3334 Lee Rd., Gainesville, FL 37123	25559999.00
31-7P	William K. Moor	904-445-2719	216 Morton Rd., Stetson, FL 30155	56850000.00

1. How many records does the file contain? How many fields are there per record?
2. What problem would you encounter if you wanted to produce a listing by city? How would you solve this problem by altering the file structure?
3. If you wanted to produce a listing of the file contents by last name, area code, city, state, or zip code, how would you alter the file structure?
4. What data redundancies do you detect? How could those redundancies lead to anomalies?
5. Identify and discuss the serious data redundancy problems exhibited by the file structure shown in Figure P1.5.

FIGURE P1.5 THE FILE STRUCTURE FOR PROBLEMS 5–8

PROJ_NUM	PROJ_NAME	EMP_NUM	EMP_NAME	JOB_CODE	JOB_CHG_HOUR	PROJ_HOURS	EMP_PHONE
1	Hurricane	101	John D. Newson	EE	65.00	13.3	653-234-3245
1	Hurricane	105	David F. Schwann	CT	60.00	16.2	653-234-1123
1	Hurricane	110	Anne R. Ramoras	CT	60.00	14.3	615-233-5568
2	Coast	101	John D. Newson	EE	65.00	19.8	653-234-3254
2	Coast	108	June H. Sattlemeir	EE	65.00	17.5	905-554-7812
3	Satellite	110	Anne R. Ramoras	CT	62.00	11.6	615-233-5568
3	Satellite	105	David F. Schwann	CT	26.00	23.4	653-234-1123
3	Satellite	123	Mary D. Chen	EE	65.00	19.1	615-233-5432
3	Satellite	112	Allecia R. Smith	BE	65.00	20.7	615-678-6879

6. Looking at the EMP_NAME and EMP_PHONE contents in Figure P1.5, what change(s) would you recommend?
7. Identify the various data sources in the file you examined in Problem 5.
8. Given your answer to Problem 7, what new files should you create to help eliminate the data redundancies found in the file shown in Figure P1.5?
9. Identify and discuss the serious data redundancy problems exhibited by the file structure shown in Figure P1.9. (The file is meant to be used as a teacher class assignment schedule. One of the many problems with data redundancy is the likely occurrence of data inconsistencies—two different initials have been entered for the teacher named Maria Cordoza.)

FIGURE P1.9 THE FILE STRUCTURE FOR PROBLEMS 9–10

BUILDING_CODE	ROOM_CODE	TEACHER_LNAME	TEACHER_FNAME	TEACHER_INITIAL	DAYS_TIME
KOM	204E	v'illiston	Horace	G	MWF 8:00-8:50
KOM	123	Cordoza	Maria	L	MWF 8:00-8:50
LDB	504	Patroski	Donald	J	TTh 1:00-2:15
KOM	34	Hawkins	Anne	vV	MWF 10:00-10:50
JKP	225B	Rioel	James		TTh 9:00-10:15
LDB	301	Robertson	Jeanette	P	TTh 9:00-10:15
KOM	204E	Cordoza	Maria	I	MWF 9:00-9:50
LDB	504	v'illiston	Horace	G	TTh 1:00-2:15
KOM	34	Cordoza	Maria	L	MWF 11:00-11:50
LDB	504	Patroski	Donald	J	MWF 2:00-2:50

10. Given the file structure shown in Figure P1.9, what problem(s) might you encounter if building KOM were deleted?
11. Using your school's student information system, print your class schedule. The schedule probably would contain the student identification number, student name, class code, class name, class credit hours, class instructor name, the class meeting days and times, and the class room number. Use Figure P1.11 as a template to complete the following actions.

FIGURE P1.11 STUDENT SCHEDULE DATA FORMAT

STU_ID	STU_ NAME	CLASS_ CODE	CLASS_ NAME	CLASS_ CREDHRS	INSTR_ NAME	CLASS_ DAYS	CLASS_ TIMES	ROOM

a) Create a spreadsheet using the template shown in Figure P1.11 and enter your current class schedule.

b) Enter the class schedule of two of your classmates into the same spreadsheet.

c) Discuss the redundancies and anomalies caused by this design.

Chapter 2

Data Models

After completing this chapter, you will be able to:

- Discuss data modeling and why data models are important
- Describe the basic data-modeling building blocks
- Define what business rules are and how they influence database design
- Understand how the major data models evolved
- List emerging alternative data models and the needs they fulfill
- Explain how data models can be classified by their level of abstraction

Preview

This chapter examines data modeling. Data modeling is the first step in the database design journey, serving as a bridge between real-world objects and the computer database.

One of the most vexing problems of database design is that designers, programmers, and end users see data in different ways. Consequently, different views of the same data can lead to database designs that do not reflect an organization's actual operation, thus failing to meet end-user needs and data efficiency requirements. To avoid such failures, database designers must obtain a precise description of the data's nature and many uses within the organization. Communication among database designers, programmers, and end users should be frequent and clear. Data modeling clarifies such communication by reducing the complexities of database design to more easily understood abstractions that define entities, relations, and data transformations.

First, you will learn some basic data-modeling concepts and how current data models have developed from earlier models. Tracing the development of those database models will help you understand the database design and implementation issues that are addressed in the rest of this book. In chronological order, you will be introduced to the hierarchical and network models, the relational model, and the entity relationship (ER) model. You will also learn about the use of the entity relationship diagram (ERD) as a data-modeling tool and the different notations used for ER diagrams. Next, you will be introduced to the object-oriented (OO) model and the object/relational model. Then, you will learn about the emerging NoSQL data model and how it is being used to fulfill the current need to manage very large social media data sets efficiently and effectively. Finally, you will learn how various degrees of data abstraction help reconcile varying views of the same data.

Data Files and Available Formats

	MS Access	Oracle	MS SQL	My SQL		MS Access	Oracle	MS SQL	My SQL
CH02_InsureCo	✓	✓	✓	✓	CH02_DealCo	✓	✓	✓	✓
					CH02_TinyCollege	✓	✓	✓	✓

Data Files Available on cengagebrain.com

2-1 Data Modeling and Data Models

data modeling
The process of creating a specific data model for a determined problem domain.

data model
A representation, usually graphic, of a complex "real-world" data structure. Data models are used in the database design phase of the Database Life Cycle.

Database design focuses on how the database structure will be used to store and manage end-user data. **Data modeling**, the first step in designing a database, refers to the process of creating a specific data model for a determined problem domain. (A *problem domain* is a clearly defined area within the real-world environment, with a well-defined scope and boundaries that will be systematically addressed.) A **data model** is a relatively simple representation, usually graphical, of more complex real-world data structures. In general terms, a *model* is an abstraction of a more complex real-world object or event. A model's main function is to help you understand the complexities of the real-world environment. Within the database environment, a data model represents data structures and their characteristics, relations, constraints, transformations, and other constructs with the purpose of supporting a specific problem domain.

Note

The terms *data model* and *database model* are often used interchangeably. In this book, the term *database model* is used to refer to the implementation of a *data model* in a specific database system

Data modeling is an iterative, progressive process. You start with a simple understanding of the problem domain, and as your understanding increases, so does the level of detail of the data model. When done properly, the final data model effectively is a "blueprint" with all the instructions to build a database that will meet all end-user requirements. This blueprint is narrative and graphical in nature, meaning that it contains both text descriptions in plain, unambiguous language and clear, useful diagrams depicting the main data elements.

Note

An implementation-ready data model should contain at least the following components:

- A description of the data structure that will store the end-user data
- A set of enforceable rules to guarantee the integrity of the data
- A data manipulation methodology to support the real-world data transformations

Traditionally, database designers relied on good judgment to help them develop a good data model. Unfortunately, good judgment is often in the eye of the beholder, and it often develops after much trial and error. For example, if each student in this class has to create a data model for a video store, it is very likely that each will come up with a different model. Which one would be correct? The simple answer is "the one that meets all the end-user requirements," and there may be more than one correct solution! Fortunately, database designers make use of existing data-modeling constructs and powerful database design tools that substantially diminish the potential for errors in database modeling. In the following sections, you will learn how existing data models are used to represent real-world data and how the different degrees of data abstraction facilitate data modeling.

2-2 The Importance of Data Models

Data models can facilitate interaction among the designer, the applications programmer, and the end user. A well-developed data model can even foster improved understanding of the organization for which the database design is developed. In short, data models are a communication tool. This important aspect of data modeling was summed up neatly by a client whose reaction was as follows: "I created this business, I worked with this business for years, and this is the first time I've really understood how all the pieces really fit together."

The importance of data modeling cannot be overstated. Data constitutes the most basic information employed by a system. Applications are created to manage data and to help transform data into information, but data is viewed in different ways by different people. For example, contrast the view of a company manager with that of a company clerk. Although both work for the same company, the manager is more likely to have an enterprise-wide view of company data than the clerk.

Even different managers view data differently. For example, a company president is likely to take a universal view of the data because he or she must be able to tie the company's divisions to a common (database) vision. A purchasing manager in the same company is likely to have a more restricted view of the data, as is the company's inventory manager. In effect, each department manager works with a subset of the company's data. The inventory manager is more concerned about inventory levels, while the purchasing manager is more concerned about the cost of items and about relationships with the suppliers of those items.

Applications programmers have yet another view of data, being more concerned with data location, formatting, and specific reporting requirements. Basically, applications programmers translate company policies and procedures from a variety of sources into appropriate interfaces, reports, and query screens.

The different users and producers of data and information often reflect the fable of the blind people and the elephant: the blind person who felt the elephant's trunk had quite a different view from the one who felt the elephant's leg or tail. A view of the whole elephant is needed. Similarly, a house is not a random collection of rooms; to build a house, a person should first have the overall view that is provided by blueprints. Likewise, a sound data environment requires an overall database blueprint based on an appropriate data model.

When a good database blueprint is available, it does not matter that an applications programmer's view of the data is different from that of the manager or the end user. Conversely, when a good database blueprint is not available, problems are likely to ensue. For instance, an inventory management program and an order entry system may use conflicting product-numbering schemes, thereby costing the company thousands or even millions of dollars.

Keep in mind that a house blueprint is an abstraction; you cannot live in the blueprint. Similarly, the data model is an abstraction; you cannot draw the required data out of the data model. Just as you are not likely to build a good house without a blueprint, you are equally unlikely to create a good database without first creating an appropriate data model.

entity
A person, place, thing, concept, or event for which data can be stored. See also *attribute*.

2-3 Data Model Basic Building Blocks

The basic building blocks of all data models are entities, attributes, relationships, and constraints. An **entity** is a person, place, thing, or event about which data will be

collected and stored. An entity represents a particular type of object in the real world, which means an entity is "distinguishable"—that is, each entity occurrence is unique and distinct. For example, a CUSTOMER entity would have many distinguishable customer occurrences, such as John Smith, Pedro Dinamita, and Tom Strickland. Entities may be physical objects, such as customers or products, but entities may also be abstractions, such as flight routes or musical concerts.

An **attribute** is a characteristic of an entity. For example, a CUSTOMER entity would be described by attributes such as customer last name, customer first name, customer phone number, customer address, and customer credit limit. Attributes are the equivalent of fields in file systems.

A **relationship** describes an association among entities. For example, a relationship exists between customers and agents that can be described as follows: an agent can serve many customers, and each customer may be served by one agent. Data models use three types of relationships: one-to-many, many-to-many, and one-to-one. Database designers usually use the shorthand notations 1:M or 1..*, M:N or *..*, and 1:1 or 1..1, respectively. (Although the M:N notation is a standard label for the many-to-many relationship, the label M:M may also be used.) The following examples illustrate the distinctions among the three relationships.

- **One-to-many (1:M or 1..*) relationship.** A painter creates many different paintings, but each is painted by only one painter. Thus, the painter (the "one") is related to the paintings (the "many"). Therefore, database designers label the relationship "PAINTER paints PAINTING" as 1:M. Note that entity names are often capitalized as a convention, so they are easily identified. Similarly, a customer (the "one") may generate many invoices, but each invoice (the "many") is generated by only a single customer. The "CUSTOMER generates INVOICE" relationship would also be labeled 1:M.
- **Many-to-many (M:N or *..*) relationship**. An employee may learn many job skills, and each job skill may be learned by many employees. Database designers label the relationship "EMPLOYEE learns SKILL" as M:N. Similarly, a student can take many classes and each class can be taken by many students, thus yielding the M:N label for the relationship expressed by "STUDENT takes CLASS."
- **One-to-one (1:1 or 1..1) relationship**. A retail company's management structure may require that each of its stores be managed by a single employee. In turn, each store manager, who is an employee, manages only a single store. Therefore, the relationship "EMPLOYEE manages STORE" is labeled 1:1.

The preceding discussion identified each relationship in both directions; that is, relationships are bidirectional:

- *One* CUSTOMER can generate *many* INVOICEs.
- Each of the *many* INVOICEs is generated by only *one* CUSTOMER.

A **constraint** is a restriction placed on the data. Constraints are important because they help to ensure data integrity. Constraints are normally expressed in the form of rules:

- An employee's salary must have values that are between 6,000 and 350,000.
- A student's GPA must be between 0.00 and 4.00.
- Each class must have one and only one teacher.

How do you properly identify entities, attributes, relationships, and constraints? The first step is to clearly identify the business rules for the problem domain you are modeling.

attribute
A characteristic of an entity or object. An attribute has a name and a data type.

relationship
An association between entities.

one-to-many (1:M or 1..*) relationship
Associations among two or more entities that are used by data models. In a 1:M relationship, one entity instance is associated with many instances of the related entity.

many-to-many (M:N or *..*) relationship
Association among two or more entities in which one occurrence of an entity is associated with many occurrences of a related entity and one occurrence of the related entity is associated with many occurrences of the first entity.

one-to-one (1:1 or 1..1) relationship
Associations among two or more entities that are used by data models. In a 1:1 relationship, one entity instance is associated with only one instance of the related entity.

constraint
A restriction placed on data, usually expressed in the form of rules. For example, "A student's GPA must be between 0.00 and 4.00."

2-4 Business Rules

When database designers go about selecting or determining the entities, attributes, and relationships that will be used to build a data model, they might start by gaining a thorough understanding of what types of data exist in an organization, how the data is used, and in what time frames it is used. But such data and information do not, by themselves, yield the required understanding of the total business. From a database point of view, the collection of data becomes meaningful only when it reflects properly defined *business rules*. A **business rule** is a brief, precise, and unambiguous description of a policy, procedure, or principle within a specific organization. In a sense, business rules are misnamed: they apply to *any* organization, large or small—a business, a government unit, a religious group, or a research laboratory—that stores and uses data to generate information.

Business rules derived from a detailed description of an organization's operations help to create and enforce actions within that organization's environment. Business rules must be rendered in writing and updated to reflect any change in the organization's operational environment.

Properly written business rules are used to define entities, attributes, relationships, and constraints. Any time you see relationship statements such as "an agent can serve many customers, and each customer can be served by only one agent," business rules are at work. You will see the application of business rules throughout this book, especially in the chapters devoted to data modeling and database design.

To be effective, business rules must be easy to understand and widely disseminated to ensure that every person in the organization shares a common interpretation of the rules. Business rules describe, in simple language, the main and distinguishing characteristics of the data *as viewed by the company*. Examples of business rules are as follows:

- A customer may generate many invoices.
- An invoice is generated by only one customer.
- A training session cannot be scheduled for fewer than 10 employees or for more than 30 employees.

Note that those business rules establish entities, relationships, and constraints. For example, the first two business rules establish two entities (CUSTOMER and INVOICE) and a 1:M relationship between those two entities. The third business rule establishes a constraint (no fewer than 10 people and no more than 30 people) and two entities (EMPLOYEE and TRAINING), and also implies a relationship between EMPLOYEE and TRAINING.

2-4a Discovering Business Rules

business rule
A description of a policy, procedure, or principle within an organization. For example, a pilot cannot be on duty for more than 10 hours during a 24-hour period, or a professor may teach up to four classes during a semester.

The main sources of business rules are company managers, policy makers, department managers, and written documentation such as a company's procedures, standards, and operations manuals. A faster and more direct source of business rules is direct interviews with end users. Unfortunately, because perceptions differ, end users are sometimes a less reliable source when it comes to specifying business rules. For example, a maintenance department mechanic might believe that any mechanic can initiate a maintenance procedure, when actually only mechanics with inspection authorization can perform such a task. Such a distinction might seem trivial, but it can have major legal consequences. Although end users are crucial contributors to the development of business rules, *it pays to verify end-user perceptions*. Too often, interviews with several people who perform

the same job yield very different perceptions of what the job components are. While such a discovery may point to "management problems," that general diagnosis does not help the database designer. The database designer's job is to reconcile such differences and verify the results of the reconciliation to ensure that the business rules are appropriate and accurate.

The process of identifying and documenting business rules is essential to database design for several reasons:

- It helps to standardize the company's view of data.
- It can be a communication tool between users and designers.
- It allows the designer to understand the nature, role, and scope of the data.
- It allows the designer to understand business processes.
- It allows the designer to develop appropriate relationship participation rules and constraints and to create an accurate data model.

Of course, not all business rules can be modeled. For example, a business rule that specifies "no pilot can fly more than 10 hours within any 24-hour period" cannot be modeled in the database model directly. However, such a business rule can be represented and enforced by application software.

2-4b Translating Business Rules into Data Model Components

Business rules set the stage for the proper identification of entities, attributes, relationships, and constraints. In the real world, names are used to identify objects. If the business environment wants to keep track of the objects, there will be specific business rules for the objects. As a general rule, a noun in a business rule will translate into an entity in the model, and a verb (active or passive) that associates the nouns will translate into a relationship among the entities. For example, the business rule "a customer may generate many invoices" contains two nouns (*customer* and *invoices*) and a verb (*generate*) that associates the nouns. From this business rule, you could deduce the following:

- *Customer* and *invoice* are objects of interest for the environment and should be represented by their respective entities.
- There is a *generate* relationship between customer and invoice.

To properly identify the type of relationship, you should consider that relationships are bidirectional; that is, they go both ways. For example, the business rule "a customer may generate many invoices" is complemented by the business rule "an invoice is generated by only one customer." In that case, the relationship is one-to-many (1:M). Customer is the "1" side, and invoice is the "many" side.

To properly identify the relationship type, you should generally ask two questions:

- How many instances of B are related to one instance of A?
- How many instances of A are related to one instance of B?

For example, you can assess the relationship between student and class by asking two questions:

- In how many classes can one student enroll? Answer: many classes.
- How many students can enroll in one class? Answer: many students.

Online Content

The hierarchical and network models are largely of historical interest, yet they do contain some elements and features that interest current database professionals. The technical details of those two models are discussed in Appendixes K and L, respectively, which are available at *www.cengagebrain.com*. Appendix G is devoted to the object-oriented (OO) model. However, given the dominant market presence of the relational model, most of the book focuses on the relational model.

Therefore, the relationship between student and class is many-to-many (M:N). You will have many opportunities to determine the relationships between entities as you proceed through this book, and soon the process will become second nature.

2-4c Naming Conventions

During the translation of business rules to data model components, you identify entities, attributes, relationships, and constraints. This identification process includes naming the object in a way that makes it unique and distinguishable from other objects in the problem domain. Therefore, it is important to pay special attention to how you name the objects you are discovering.

Entity names should be descriptive of the objects in the business environment and use terminology that is familiar to the users. An attribute name should also be descriptive of the data represented by that attribute. It is also a good practice to prefix the name of an attribute with the name or abbreviation of the entity in which it occurs. For example, in the CUSTOMER entity, the customer's credit limit may be called CUS_CREDIT_LIMIT. The CUS indicates that the attribute is descriptive of the CUSTOMER entity, while CREDIT_LIMIT makes it easy to recognize the data that will be contained in the attribute. This will become increasingly important in later chapters when you learn about the need to use common attributes to specify relationships between entities. The use of a proper naming convention will improve the data model's ability to facilitate communication among the designer, application programmer, and the end user. In fact, a proper naming convention can go a long way toward making your model self-documenting.

2-5 The Evolution of Data Models

The quest for better data management has led to several models that attempt to resolve the previous model's critical shortcomings and to provide solutions to ever-evolving data management needs. These models represent schools of thought as to what a database is, what it should do, the types of structures that it should employ, and the technology that would be used to implement these structures. Perhaps confusingly, these models are called data models, as are the graphical data models discussed earlier in this chapter. This section gives an overview of the major data models in roughly chronological order. You will discover that many of the "new" database concepts and structures bear a remarkable resemblance to some of the "old" data model concepts and structures. Table 2.1 traces the evolution of the major data models.

hierarchical model
An early database model whose basic concepts and characteristics formed the basis for subsequent database development. This model is based on an upside-down tree structure in which each record is called a segment. The top record is the root segment. Each segment has a 1:M relationship to the segment directly below it.

segment
In the hierarchical data model, the equivalent of a file system's record type.

network model
An early data model that represented data as a collection of record types in 1:M relationships.

2-5a Hierarchical and Network Models

The **hierarchical model** was developed in the 1960s to manage large amounts of data for complex manufacturing projects, such as the Apollo rocket that landed on the moon in 1969. The model's basic logical structure is represented by an upside-down tree. The hierarchical structure contains levels, or segments. A **segment** is the equivalent of a file system's record type. Within the hierarchy, a higher layer is perceived as the parent of the segment directly beneath it, which is called the child. The hierarchical model depicts a set of one-to-many (1:M) relationships between a parent and its children segments. (Each parent can have many children, but each child has only one parent.)

The **network model** was created to represent complex data relationships more effectively than the hierarchical model, to improve database performance, and to impose a database standard. In the network model, the user perceives the network database as a collection of records in 1:M relationships. However, unlike the hierarchical model, the

TABLE 2.1

EVOLUTION OF MAJOR DATA MODELS

GENERATION	TIME	DATA MODEL	EXAMPLES	COMMENTS
First	1960s–1970s	File system	VMS/VSAM	Used mainly on IBM mainframe systems Managed records, not relationships
Second	1970s	Hierarchical and network	IMS, ADABAS, IDS-II	Early database systems Navigational access
Third	Mid-1970s	Relational	DB2 Oracle MS SQL Server MySQL	Conceptual simplicity Entity relationship (ER) modeling and support for relational data modeling
Fourth	Mid-1980s	Object-oriented Object/relational (O/R)	Versant Objectivity/DB DB2 UDB Oracle 12c	Object/relational supports object data types Star Schema support for data warehousing Web databases become common
Fifth	Mid-1990s	XML Hybrid DBMS	dbXML Tamino DB2 UDB Oracle 12c MS SQL Server	Unstructured data support O/R model supports XML documents Hybrid DBMS adds object front end to relational databases Support large databases (terabyte size)
Emerging Models: NoSQL	Early 2000s to present	Key-value store Column store	SimpleDB (Amazon) BigTable (Google) Cassandra (Apache) MongoDB Riak	Distributed, highly scalable High performance, fault tolerant Very large storage (petabytes) Suited for sparse data Proprietary application programming interface (API)

network model allows a record to have more than one parent. While the network database model is generally not used today, the definitions of standard database *concepts* that emerged with the network model are still used by modern data models:

- The **schema** is the conceptual organization of the entire database as viewed by the database administrator.
- The **subschema** defines the portion of the database "seen" by the application programs that actually produce the desired information from the data within the database.
- A **data manipulation language (DML)** defines the environment in which data can be managed and is used to work with the data in the database.
- A schema **data definition language (DDL)** enables the database administrator to define the schema components.

schema
A logical grouping of database objects, such as tables, indexes, views, and queries, that are related to each other.

subschema
The portion of the database that interacts with application programs.

data manipulation language (DML)
The set of commands that allows an end user to manipulate the data in the database, such as SELECT, INSERT, UPDATE, DELETE, COMMIT, and ROLLBACK.

As information needs grew and more sophisticated databases and applications were required, the network model became too cumbersome. The lack of ad hoc query capability put heavy pressure on programmers to generate the code required to produce even the simplest reports. Although the existing databases provided limited data independence, any structural change in the database could still produce havoc in all application programs that drew data from the database. Because of the disadvantages of the hierarchical and network models, they were largely replaced by the relational data model in the 1980s.

data definition language (DDL)
The language that allows a database administrator to define the database structure, schema, and subschema.

2-5b The Relational Model

The **relational model** was introduced in 1970 by E. F. Codd of IBM in his landmark paper "A Relational Model of Data for Large Shared Databanks" (*Communications of the ACM*, June 1970, pp. 377–387). The relational model represented a major breakthrough for both users and designers. To use an analogy, the relational model produced an "automatic transmission" database to replace the "standard transmission" databases that preceded it. Its conceptual simplicity set the stage for a genuine database revolution.

Note

The relational database model presented in this chapter is an introduction and an overview. A more detailed discussion is in Chapter 3, The Relational Database Model. In fact, the relational model is so important that it will serve as the basis for discussions in most of the remaining chapters.

relational model
Developed by E. F. Codd of IBM in 1970, the relational model is based on mathematical set theory and represents data as independent relations. Each relation (table) is conceptually represented as a two-dimensional structure of intersecting rows and columns. The relations are related to each other through the sharing of common entity characteristics (values in columns).

table (relation)
A logical construct perceived to be a two-dimensional structure composed of intersecting rows (entities) and columns (attributes) that represents an entity set in the relational model.

tuple
In the relational model, a table row.

relational database management system (RDBMS)
A collection of programs that manages a relational database. The RDBMS software translates a user's logical requests (queries) into commands that physically locate and retrieve the requested data.

The relational model's foundation is a mathematical concept known as a relation. To avoid the complexity of abstract mathematical theory, you can think of a **relation** (sometimes called a **table**) as a two-dimensional structure composed of intersecting rows and columns. Each row in a relation is called a **tuple**. Each column represents an attribute. The relational model also describes a precise set of data manipulation constructs based on advanced mathematical concepts.

In 1970, Codd's work was considered ingenious but impractical. The relational model's conceptual simplicity was bought at the expense of computer overhead; computers at that time lacked the power to implement the relational model. Fortunately, computer power grew exponentially, as did operating system efficiency. Better yet, the cost of computers diminished rapidly as their power grew. Today, even PCs, which cost a fraction of what their mainframe ancestors cost, can run sophisticated relational database software such as Oracle, DB2, Microsoft SQL Server, MySQL, and other mainframe relational software.

The relational data model is implemented through a very sophisticated **relational database management system (RDBMS)**. The RDBMS performs the same basic functions provided by the hierarchical and network DBMS systems, in addition to a host of other functions that make the relational data model easier to understand and implement (as outlined in Chapter 1, in the DBMS Functions section).

Arguably the most important advantage of the RDBMS is its ability to hide the complexities of the relational model from the user. The RDBMS manages all of the physical details, while the user sees the relational database as a collection of tables in which data is stored. The user can manipulate and query the data in a way that seems intuitive and logical.

Tables are related to each other through the sharing of a common attribute (a value in a column). For example, the CUSTOMER table in Figure 2.1 might contain a sales agent's number that is also contained in the AGENT table.

The common link between the CUSTOMER and AGENT tables enables you to match the customer to his or her sales agent, even though the customer data is stored in one table and the sales representative data is stored in another table. For example, you can easily determine that customer Dunne's agent is Alex Alby because for customer Dunne, the CUSTOMER table's AGENT_CODE is 501, which matches the AGENT table's AGENT_CODE for Alex Alby. Although the tables are independent of one another, you

FIGURE 2.1 LINKING RELATIONAL TABLES

Table name: AGENT (first six attributes) **Database name: Ch02_InsureCo**

AGENT_CODE	AGENT_LNAME	AGENT_FNAME	AGENT_INITIAL	AGENT_AREACODE	AGENT_PHONE
501	Alby	Alex	B	713	228-1249
502	Hahn	Leah	F	615	882-1244
503	Okon	John	T	615	123-5589

Link through AGENT_CODE

Table name: CUSTOMER

CUS_CODE	CUS_LNAME	CUS_FNAME	CUS_INITIAL	CUS_AREACODE	CUS_PHONE	CUS_INSURE_TYPE	CUS_INSURE_AMT	CUS_RENEW_DATE	AGENT_CODE
10010	Ramas	Alfred	A	615	844-2573	T1	100.00	05-Apr-2018	502
10011	Dunne	Leona	K	713	894-1238	T1	250.00	16-Jun-2018	501
10012	Smith	Kathy	W	615	894-2285	S2	150.00	29-Jan-2019	502
10013	Olowski	Paul	F	615	894-2180	S1	300.00	14-Oct-2018	502
10014	Orlando	Myron		615	222-1672	T1	100.00	28-Dec-2019	501
10015	O'Brian	Amy	B	713	442-3381	T2	850.00	22-Sep-2018	503
10016	Brown	James	G	615	297-1228	S1	120.00	25-Mar-2019	502
10017	Williams	George		615	290-2556	S1	250.00	17-Jul-2018	503
10018	Farriss	Anne	G	713	382-7185	T2	100.00	03-Dec-2018	501
10019	Smith	Olette	K	615	297-3809	S2	500.00	14-Mar-2019	503

can easily associate the data between tables. The relational model provides a minimum level of controlled redundancy to eliminate most of the redundancies commonly found in file systems.

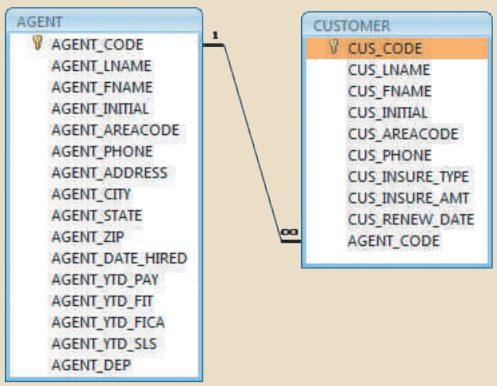
The relationship type (1:1, 1:M, or M:N) is often shown in a relational schema, an example of which is shown in Figure 2.2. A **relational diagram** is a representation of the relational database's entities, the attributes within those entities, and the relationships between those entities.

In Figure 2.2, the relational diagram shows the connecting fields (in this case, AGENT_CODE) and the relationship type (1:M). Microsoft Access, the database software application used to generate Figure 2.2, employs the infinity symbol (∞) to indicate the "many" side. In this example, the CUSTOMER represents the "many" side because an AGENT can have many CUSTOMERs. The AGENT represents the "1" side because each CUSTOMER has only one AGENT.

Online Content

This chapter's databases are available at *www.cengagebrain.com*. For example, the contents of the AGENT and CUSTOMER tables shown in Figure 2.1 are in the database named Ch02_InsureCo.

FIGURE 2.2 A RELATIONAL DIAGRAM



relational diagram
A graphical representation of a relational database's entities, the attributes within those entities, and the relationships among the entities.

A relational table stores a collection of related entities. In this respect, the relational database table resembles a file, but there is a crucial difference between a table and a file: a table yields complete data and structural independence because it is a purely logical structure. How the data is physically stored in the database is of no concern to the user or the designer; the perception is what counts. This property of the relational data model, which is explored in depth in the next chapter, became the source of a real database revolution.

Another reason for the relational data model's rise to dominance is its powerful and flexible query language. Most relational database software uses Structured Query Language (SQL), which allows the user to specify what must be done without specifying how. The RDBMS uses SQL to translate user queries into instructions for retrieving the requested data. SQL makes it possible to retrieve data with far less effort than any other database or file environment.

From an end-user perspective, any SQL-based relational database application involves three parts: a user interface, a set of tables stored in the database, and the SQL "engine." Each of these parts is explained as follows:

- *The end-user interface*. Basically, the interface allows the end user to interact with the data (by automatically generating SQL code). Each interface is a product of the software vendor's idea of meaningful interaction with the data. You can also design your own customized interface with the help of application generators that are now standard fare in the database software arena.
- *A collection of tables stored in the database*. In a relational database, all data is perceived to be stored in tables. The tables simply "present" the data to the end user in a way that is easy to understand. Each table is independent. Rows in different tables are related by common values in common attributes.
- *SQL engine*. Largely hidden from the end user, the SQL engine executes all queries, or data requests. Keep in mind that the SQL engine is part of the DBMS software. The end user uses SQL to create table structures and to perform data access and table maintenance. The SQL engine processes all user requests—largely behind the scenes and without the end user's knowledge. Hence, SQL is said to be a declarative language that tells what must be done but not how. (You will learn more about the SQL engine in Chapter 11, Database Performance Tuning and Query Optimization.)

Because the RDBMS performs some tasks behind the scenes, it is not necessary to focus on the physical aspects of the database. Instead, the following chapters concentrate on the logical portion of the relational database and its design. Furthermore, SQL is covered in detail in Chapter 7, Introduction to Structured Query Language (SQL), and in Chapter 8, Advanced SQL.

2-5c The Entity Relationship Model

The conceptual simplicity of relational database technology triggered the demand for RDBMSs. In turn, the rapidly increasing requirements for transaction and information created the need for more complex database implementation structures, thus creating the need for more effective database design tools. (Building a skyscraper requires more detailed design activities than building a doghouse, for example.)

Complex design activities require conceptual simplicity to yield successful results. Although the relational model was a vast improvement over the hierarchical and network models, it still lacked the features that would make it an effective database *design* tool. Because it is easier to examine structures graphically than to describe them in text,

database designers prefer to use a graphical tool in which entities and their relationships are pictured. Thus, the **entity relationship (ER) model,** or **ERM**, has become a widely accepted standard for data modeling.

Peter Chen first introduced the ER data model in 1976; the graphical representation of entities and their relationships in a database structure quickly became popular because it *complemented* the relational data model concepts. The relational data model and ERM combined to provide the foundation for tightly structured database design. ER models are normally represented in an **entity relationship diagram (ERD)**, which uses graphical representations to model database components. You will learn how to use ERDs to design databases in Chapter 4, Entity Relationship (ER) Modeling.

The ER model is based on the following components:

- *Entity*. Earlier in this chapter, an entity was defined as anything about which data will be collected and stored. An entity is represented in the ERD by a rectangle, also known as an entity box. The name of the entity, a noun, is written in the center of the rectangle. The entity name is generally written in capital letters and in singular form: PAINTER rather than PAINTERS, and EMPLOYEE rather than EMPLOYEES. Usually, when applying the ERD to the relational model, an entity is mapped to a relational table. Each row in the relational table is known as an **entity instance** or **entity occurrence** in the ER model. A collection of like entities is known as an **entity set**. For example, you can think of the AGENT file in Figure 2.1 as a collection of three agents (*entities*) in the AGENT *entity* set. Technically speaking, the ERD depicts entity sets. Unfortunately, ERD designers use the word *entity* as a substitute for *entity set*, and this book will conform to that established practice when discussing any ERD and its components.
- Each entity consists of a set of *attributes* that describes particular characteristics of the entity. For example, the entity EMPLOYEE will have attributes such as a Social Security number, a last name, and a first name. (Chapter 4 explains how attributes are included in the ERD.)
- *Relationships*. Relationships describe associations among data. Most relationships describe associations between two entities. When the basic data model components were introduced, three types of data relationships were illustrated: one-to-many (1:M), many-to-many (M:N), and one-to-one (1:1). The ER model uses the term **connectivity** to label the relationship types. The name of the relationship is usually an active or passive verb. For example, a PAINTER *paints* many PAINTINGs, an EMPLOYEE *learns* many SKILLs, and an EMPLOYEE *manages* a STORE.

Figure 2.3 shows the different types of relationships using three ER notations: the original **Chen notation**, the **Crow's Foot notation**, and the newer **class diagram notation**, which is part of the Unified Modeling Language (UML).

The left side of the ER diagram shows the Chen notation, based on Peter Chen's landmark paper. In this notation, the connectivities are written next to each entity box. Relationships are represented by a diamond connected to the related entities through a relationship line. The relationship name is written inside the diamond.

The middle of Figure 2.3 illustrates the Crow's Foot notation. The name *Crow's Foot* is derived from the three-pronged symbol used to represent the "many" side of the relationship. As you examine the basic Crow's Foot ERD in Figure 2.3, note that the connectivities are represented by symbols. For example, the "1" is represented by a short line segment, and the "M" is represented by the three-pronged "crow's foot." In this example, the relationship name is written above the relationship line.

The right side of Figure 2.3 shows the UML notation (also known as the UML class notation). Note that the connectivities are represented by lines with symbols (1..1, 1..*).

entity relationship (ER) model (ERM)
A data model that describes relationships (1:1, 1:M, and M:N) among entities at the conceptual level with the help of ER diagrams.

entity relationship diagram (ERD)
A diagram that depicts an entity relationship model's entities, attributes, and relations.

entity instance (entity occurrence)
A row in a relational table.

entity set
A collection of like entities.

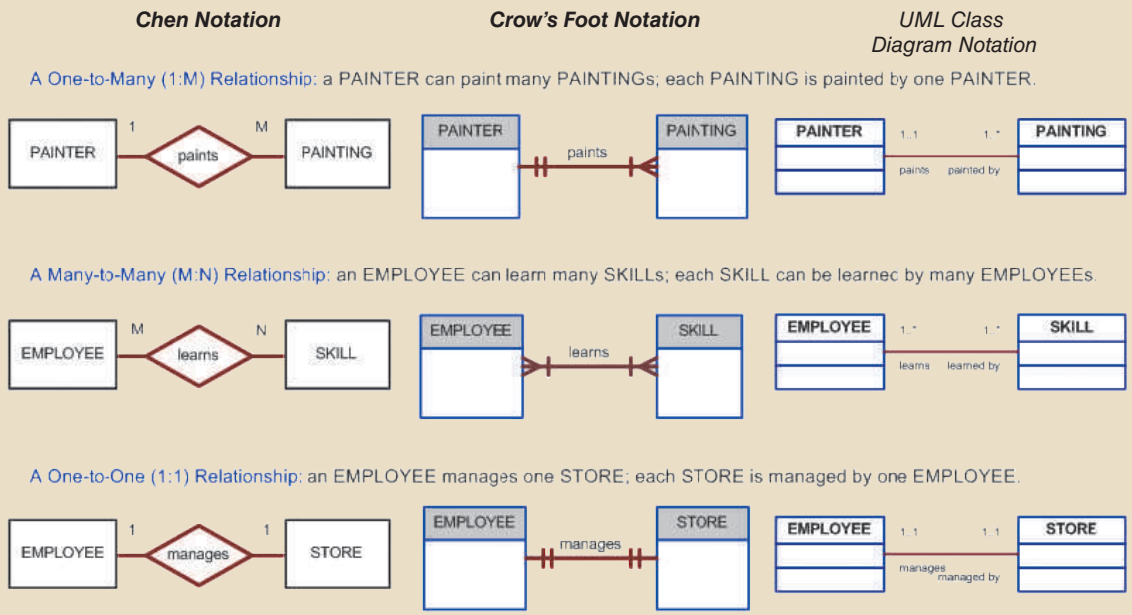
connectivity
The type of relationship between entities. Classifications include 1:1, 1:M, and M:N.

Chen notation
See *entity relationship (ER) model*.

Crow's Foot notation
A representation of the entity relationship diagram that uses a three-pronged symbol to represent the "many" sides of the relationship.

class diagram notation
The set of symbols used in the creation of class diagrams.

FIGURE 2.3 THE ER MODEL NOTATIONS



Also, the UML notation uses names in both sides of the relationship. For example, to read the relationship between PAINTER and PAINTING, note the following:

- A PAINTER "paints" one to many PAINTINGs, as indicated by the 1..* symbol.
- A PAINTING is "painted by" one and only one PAINTER, as indicated by the 1..1 symbol.

Note

Many-to-many (M:N) relationships exist at a conceptual level, and you should know how to recognize them. However, you will learn in Chapter 3 that M:N relationships are not appropriate in a relational model. For that reason, Microsoft Visio does not support the M:N relationship directly. Therefore, to illustrate the existence of an M:N relationship using Visio, you have to change the line style of the connector (see Appendix A, Designing Databases with Visio Professional: A Tutorial, at *www.cengagebrain.com*).

In Figure 2.3, entities and relationships are shown in a horizontal format, but they may also be oriented vertically. The entity location and the order in which the entities are presented are immaterial; just remember to read a 1:M relationship from the "1" side to the "M" side.

The Crow's Foot notation is used as the design standard in this book. However, the Chen notation is used to illustrate some of the ER modeling concepts whenever necessary. Most data modeling tools let you select the Crow's Foot or UML class diagram notation. Microsoft Visio Professional software was used to generate the Crow's Foot designs you will see in subsequent chapters.

The ER model's exceptional visual simplicity makes it the dominant database modeling and design tool. Nevertheless, the search for better data-modeling tools continues as the data environment continues to evolve.

2-5d The Object-Oriented Model

Increasingly complex real-world problems demonstrated a need for a data model that more closely represented the real world. In the **object-oriented data model (OODM)**, both data *and its relationships* are contained in a single structure known as an **object**. In turn, the OODM is the basis for the **object-oriented database management system (OODBMS)**.

An OODM reflects a very different way to define and use entities. Like the relational model's entity, an object is described by its factual content. But, quite *unlike* an entity, an object includes information about relationships between the facts within the object, as well as information about its relationships with other objects. Therefore, the facts within the object are given greater *meaning*. The OODM is said to be a **semantic data model** because *semantic* indicates meaning.

Subsequent OODM development has allowed an object also to contain all *operations* that can be performed on it, such as changing its data values, finding a specific data value, and printing data values. Because objects include data, various types of relationships, and operational procedures, the object becomes self-contained, thus making it—at least potentially—a basic building block for autonomous structures.

The OO data model is based on the following components:

- An object is an abstraction of a real-world entity. In general terms, an object may be considered equivalent to an ER model's entity. More precisely, an object represents only one occurrence of an entity. (The object's semantic content is defined through several of the items in this list.)
- Attributes describe the properties of an object. For example, a PERSON object includes the attributes Name, Social Security Number, and Date of Birth.
- Objects that share similar characteristics are grouped in classes. A **class** is a collection of similar objects with shared structure (attributes) and behavior (methods). In a general sense, a class resembles the ER model's *entity* set. However, a class is different from an entity set in that it contains a set of procedures known as *methods*. A class's **method** represents a real-world action such as *finding* a selected PERSON's name, *changing* a PERSON's name, or *printing* a PERSON's address. In other words, methods are the equivalent of *procedures* in traditional programming languages. In OO terms, methods define an object's *behavior*.
- Classes are organized in a *class hierarchy*. The **class hierarchy** resembles an upside-down tree in which each class has only one parent. For example, the CUSTOMER class and the EMPLOYEE class share a parent PERSON class. (Note the similarity to the hierarchical data model in this respect.)
- **Inheritance** is the ability of an object within the class hierarchy to inherit the attributes and methods of the classes above it. For example, two classes, CUSTOMER and EMPLOYEE, can be created as subclasses from the class PERSON. In this case, CUSTOMER and EMPLOYEE will inherit all attributes and methods from PERSON.
- Object-oriented data models are typically depicted using **Unified Modeling Language (UML)** class diagrams. UML is a language based on OO concepts that describes a set of diagrams and symbols you can use to graphically model a system. UML **class diagrams** are used to represent data and its relationships within the larger UML object-oriented system's modeling language. For a more complete description of UML, see Appendix H, Unified Modeling Language (UML).

Online Content

This chapter introduces only basic OO concepts. You can examine object-orientation concepts and principles in detail in Appendix G, Object-Oriented Databases, at *www.cengagebrain.com.*

object-oriented data model (OODM)
A data model whose basic modeling structure is an object.

object
An abstract representation of a real-world entity that has a unique identity, embedded properties, and the ability to interact with other objects and itself.

object-oriented database management system (OODBMS)
Data management software used to manage data in an object-oriented database model.

semantic data model
The first of a series of data models that models both data and their relationships in a single structure known as an object.

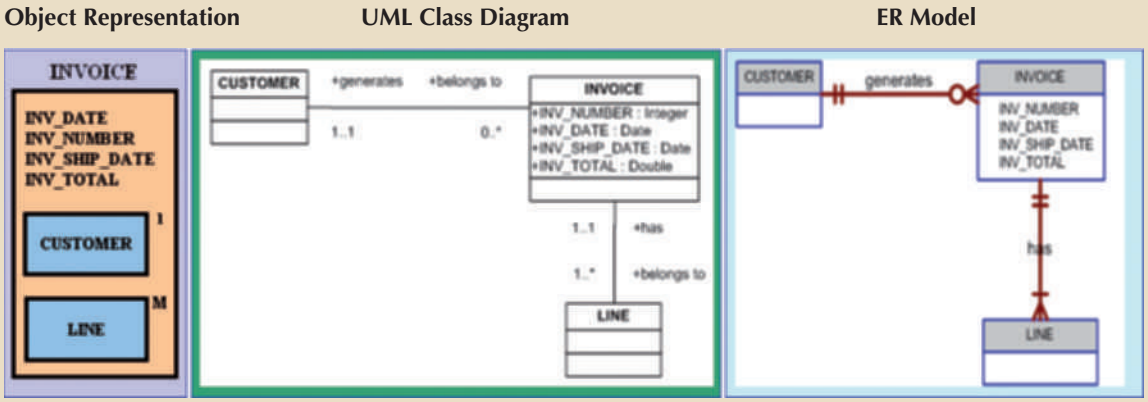
class
A collection of similar objects with shared structure (attributes) and behavior (methods). A class encapsulates an object's data representation and a method's implementation.

method
In the object-oriented data model, a named set of instructions to perform an action. Methods represent real-world actions.

class hierarchy
The organization of classes in a hierarchical tree in which each parent class is a *superclass* and each child class is a *subclass*. See also *inheritance*.

To illustrate the main concepts of the OODM, consider a simple invoicing problem. In this case, invoices are generated by customers, each invoice references one or more lines, and each line represents an item purchased by a customer. Figure 2.4 illustrates the object representation for this simple invoicing problem, as well as the equivalent UML class diagram and ER model. The object representation is a simple way to visualize a single object occurrence.

FIGURE 2.4 A COMPARISON OF THE OO, UML, AND ER MODELS



As you examine Figure 2.4, note the following:

- The object representation of the INVOICE includes all related objects within the *same* object box. Note that the connectivities (1 and M) indicate the relationship of the related objects to the INVOICE. For example, the "1" next to the CUSTOMER object indicates that each INVOICE is related to only one CUSTOMER. The "M" next to the LINE object indicates that each INVOICE contains many LINEs.
- The UML class diagram uses three separate object classes (CUSTOMER, INVOICE, and LINE) and two relationships to represent this simple invoicing problem. Note that the relationship connectivities are represented by the 1..1, 0..*, and 1..* symbols, and that the relationships are named in both ends to represent the different "roles" that the objects play in the relationship.
- The ER model also uses three separate entities and two relationships to represent this simple invoice problem.

inheritance
In the object-oriented data model, the ability of an object to inherit the data structure and methods of the classes above it in the class hierarchy. See also *class hierarchy*.

Unified Modeling Language (UML)
A language based on object-oriented concepts that provides tools such as diagrams and symbols to graphically model a system.

class diagram
A diagram used to represent data and their relationships in UML object notation.

The OODM advances influenced many areas, from system modeling to programming. (Most contemporary programming languages have adopted OO concepts, including Java, Ruby, Perl, C#, and Visual Studio .NET.) The added semantics of the OODM allowed for a richer representation of complex objects. This in turn enabled applications to support increasingly complex objects in innovative ways. As you will see in the next section, such evolutionary advances also affected the relational model.

2-5e Object/Relational and XML

Facing the demand to support more complex data representations, the relational model's main vendors evolved the model further and created the **extended relational data**

model (ERDM). The ERDM adds many of the OO model's features within the inherently simpler relational database structure. The ERDM gave birth to a new generation of relational databases that support OO features such as objects (encapsulated data and methods), extensible data types based on classes, and inheritance. That's why a DBMS based on the ERDM is often described as an **object/relational database management system (O/R DBMS)**.

Today, most relational database products can be classified as object/relational, and they represent the dominant market share of OLTP and OLAP database applications. The success of the O/R DBMSs can be attributed to the model's conceptual simplicity, data integrity, easy-to-use query language, high transaction performance, high availability, security, scalability, and expandability. In contrast, the OO DBMS is popular in niche markets such as computer-aided drawing/computer-aided manufacturing (CAD/CAM), geographic information systems (GIS), telecommunications, and multimedia, which require support for more complex objects.

From the start, the OO and relational data models were developed in response to different problems. The OO data model was created to address very specific engineering needs, not the wide-ranging needs of general data management tasks. The relational model was created with a focus on better data management based on a sound mathematical foundation. Given its focus on a smaller set of problem areas, it is not surprising that the OO market has not grown as rapidly as the relational data model market.

The use of complex objects received a boost with the Internet revolution. When organizations integrated their business models with the Internet, they realized its potential to access, distribute, and exchange critical business information. This resulted in the widespread adoption of the Internet as a business communication tool. Within this environment, **Extensible Markup Language (XML)** emerged as the de facto standard for the efficient and effective exchange of structured, semistructured, and unstructured data. Organizations that used XML data soon realized that they needed to manage large amounts of unstructured data such as word-processing documents, webpages, emails, and diagrams. To address this need, XML databases emerged to manage unstructured data within a native XML format (see Chapter 15, Database Connectivity and Web Technologies, for more information about XML). At the same time, O/R DBMSs added support for XML-based documents within their relational data structure. Due to its robust foundation in broadly applicable principles, the relational model is easily extended to include new classes of capabilities, such as objects and XML.

Although relational and object/relational databases address most current data processing needs, a new generation of databases has emerged to address some very specific challenges found in some Internet-era organizations.

extended relational data model (ERDM)
A model that includes the object-oriented model's best features in an inherently simpler relational database structural environment. See *extended entity relationship model (EERM)*.

object/relational database management system (O/R DBMS)
A DBMS based on the extended relational model (ERDM). The ERDM, championed by many relational database researchers, constitutes the relational model's response to the OODM. This model includes many of the object-oriented model's best features within an inherently simpler relational database structure.

Extensible Markup Language (XML)
A metalanguage used to represent and manipulate data elements. Unlike other markup languages, XML permits the manipulation of a document's data elements. XML facilitates the exchange of structured documents such as orders and invoices over the Internet.

2-5f Emerging Data Models: Big Data and NoSQL

Deriving usable business information from the mountains of web data that organizations have accumulated over the years has become an imperative need. Web data in the form of browsing patterns, purchasing histories, customer preferences, behavior patterns, and social media data from sources such as Facebook, Twitter, and LinkedIn have inundated organizations with combinations of structured and unstructured data. In addition, mobile technologies such as smartphones and tablets, plus sensors of all types—GPS, RFID systems, weather sensors, biomedical devices, space research probes, car and aviation black boxes—as well as other Internet and cellular-connected devices, have created new ways to automatically collect massive amounts of data in multiple formats (text, pictures, sound, video, etc.). The amount of data being collected grows exponentially every day. According to IBM, "Every day we create 2.5 quintillion bytes of data—so much that 90 percent of the

data in the world today has been created in the last two years alone."[1] According to some studies, the rapid pace of data growth is the top challenge for organizations,[2] with system performance and scalability as the next biggest challenges. Today's information technology (IT) managers are constantly balancing the need to manage this rapidly growing data with shrinking budgets. The need to manage and leverage all these converging trends (rapid data growth, performance, scalability, and lower costs) has triggered a phenomenon called "Big Data." **Big Data** refers to a movement to find new and better ways to manage large amounts of web- and sensor-generated data and derive business insight from it, while simultaneously providing high performance and scalability at a reasonable cost.

The term *Big Data* has been used in many different frameworks, from law to statistics to economics to computing. The term seems to have been first used in a computing framework by John Mashey, a Silicon Graphics scientist in the 1990s.[3] However, it seems to be Douglas Laney, a data analyst from the Gartner Group, who first described the basic characteristics of Big Data databases[4]: volume, velocity, and variety, or the **3 Vs**.

- *Volume* refers to the amounts of data being stored. With the adoption and growth of the Internet and social media, companies have multiplied the ways to reach customers. Over the years, and with the benefit of technological advances, data for millions of e-transactions were being stored daily on company databases. Furthermore, organizations are using multiple technologies to interact with end users and those technologies are generating mountains of data. This ever-growing volume of data quickly reached petabytes in size, and it's still growing.
- *Velocity* refers not only to the speed with which data grows but also to the need to process this data quickly in order to generate information and insight. With the advent of the Internet and social media, business response times have shrunk considerably. Organizations need not only to store large volumes of quickly accumulating data but also need to process such data quickly. The velocity of data growth is also due to the increase in the number of different data streams from which data is being piped to the organization (via the web, e-commerce, Tweets, Facebook posts, emails, sensors, GPS, and so on).
- *Variety* refers to the fact that the data being collected comes in multiple different data formats. A great portion of these data comes in formats not suitable to be handled by the typical operational databases based on the relational model.

The 3 Vs framework illustrates what companies now know, that the amount of data being collected in their databases has been growing exponentially in size and complexity. Traditional relational databases are good at managing structured data but are not well suited to managing and processing the amounts and types of data being collected in today's business environment.

Big Data
A movement to find new and better ways to manage large amounts of web-generated data and derive business insight from it, while simultaneously providing high performance and scalability at a reasonable cost.

3 Vs
Three basic characteristics of Big Data databases: volume, velocity, and variety.

The problem is that the relational approach does not always match the needs of organizations with Big Data challenges.

- It is not always possible to fit unstructured, social media and sensor-generated data into the conventional relational structure of rows and columns.
- Adding millions of rows of multiformat (structured and nonstructured) data on a daily basis will inevitably lead to the need for more storage, processing power, and

[1] IBM, "What is big data? Bringing big data to the enterprise," http://www-01.ibm.com/software/data/bigdata/, accessed April 2013.
[2] "Gartner survey shows data growth as the largest data center infrastructure challenge," www.gartner.com/it/page.jsp?id=1460213, accessed March 2015.
[3] Steve Lohr, "The origins of 'Big Data': An etymological detective story," *New York Times*, February 1, 2013.
[4] Douglas Laney, "3D data management controlling data volume, velocity and variety," META Group, February 6, 2011.

sophisticated data analysis tools that may not be available in the relational environment. The type of high-volume implementations required in the RDBMS environment for the Big Data problem comes with a hefty price tag for expanding hardware, storage, and software licenses.

- Data analysis based on OLAP tools has proven to be very successful in relational environments with highly structured data. However, mining for usable data in the vast amounts of unstructured data collected from web sources requires a different approach.

There is no "one-size-fits-all" cure to data management needs (although many established database vendors will probably try to sell you on the idea). For some organizations, creating a highly scalable, fault-tolerant infrastructure for Big Data analysis could prove to be a matter of business survival. The business world has many examples of companies that leverage technology to gain a competitive advantage, and others that miss it. Just ask yourself how the business landscape would be different if:

- Blackberry had responded quickly to the emerging Apple smartphone technology.
- MySpace had responded to Facebook's challenge in time.
- Blockbuster had reacted to the Netflix business model sooner.
- Barnes & Noble had developed a viable Internet strategy before Amazon.

Will broadcast television networks be successful in adapting to streaming services such as Hulu, AppleTV, and Roku? Partnerships and mergers will undoubtedly change the landscape of home entertainment as the industry responds to the changing technological possibilities. Will traditional news outlets be able to adapt to the changing news consumption patterns of the millennial generation?

Big Data analytics are being used to create new types of services by all types of companies. For example, Amazon originally competed with "big box" department stores as a low-cost provider. Amazon eventually leveraged storage and processing technologies to begin competing in streaming movie and music service, and more recently, it has leveraged Big Data to create innovative services like predictive shipping. Predictive shipping uses a customer's purchase patterns to predict when a product will be needed and ship it to the customer before the customer even realizes that she needs it! Amazon has also been successful with the sales of products like Amazon Echo that use the Alexa service to perform natural language processing. These "constantly listening" devices are embedded in homes around the world, providing Amazon with unprecedented levels and types of data that it can analyze to improve existing services and support innovation in future services.

In order to create value from their previously unused Big Data stores, companies are using new Big Data technologies. These emerging technologies allow organizations to process massive data stores of multiple formats in cost-effective ways. Some of the most frequently used Big Data technologies are Hadoop, MapReduce, and NoSQL databases.

- **Hadoop** is a Java-based, open-source, high-speed, fault-tolerant distributed storage and computational framework. Hadoop uses low-cost hardware to create clusters of thousands of computer nodes to store and process data. Hadoop originated from Google's work on distributed file systems and parallel processing and is currently supported by the Apache Software Foundation.[5] Hadoop has several modules, but the two main components are Hadoop Distributed File System (HDFS) and MapReduce.
- **Hadoop Distributed File System (HDFS)** is a highly distributed, fault-tolerant file storage system designed to manage large amounts of data at high speeds. In order to achieve high throughput, HDFS uses the write-once, read many model. This means

Hadoop
A Java-based, open-source, high-speed, fault-tolerant distributed storage and computational framework. Hadoop uses low-cost hardware to create clusters of thousands of computer nodes to store and process data.

Hadoop Distributed File System (HDFS)
A highly distributed, fault-tolerant file storage system designed to manage large amounts of data at high speeds.

[5] For more information about Hadoop, visit hadoop.apache.org.

name node
One of three types of nodes used in the Hadoop Distributed File System (HDFS). The name node stores all the metadata about the file system. See also *client node* and *data node*.

data node
One of three types of nodes used in the Hadoop Distributed File System (HDFS). The data node stores fixed-size data blocks (that could be replicated to other data nodes). See also *client node* and *name node*.

client node
One of three types of nodes used in the Hadoop Distributed File System (HDFS). The client node acts as the interface between the user application and the HDFS. See also *name node* and *data node*.

MapReduce
An open-source application programming interface (API) that provides fast data analytics services; one of the main Big Data technologies that allows organizations to process massive data stores.

that once the data is written, it cannot be modified. HDFS uses three types of nodes: a **name node** that stores all the metadata about the file system, a **data node** that stores fixed-size data blocks (that could be replicated to other data nodes), and a **client node** that acts as the interface between the user application and the HDFS.

- **MapReduce** is an open-source application programming interface (API) that provides fast data analytics services. MapReduce distributes the processing of the data among thousands of nodes in parallel. MapReduce works with structured and nonstructured data. The MapReduce framework provides two main functions: Map and Reduce. In general terms, the Map function takes a job and divides it into smaller units of work, and the Reduce function collects all the output results generated from the nodes and integrates them into a single result set. Although MapReduce itself is viewed as fairly limited today, it defined the paradigm for how Big Data is processed.
- **NoSQL** is a large-scale distributed database system that stores structured and unstructured data in efficient ways. NoSQL databases are discussed in more detail in Chapter 14, Big Data and NoSQL.

Hadoop technologies provide a framework for Big Data analytics in which data (structured or unstructured) is distributed, replicated, and processed in parallel using a network of low-cost commodity hardware. Hadoop introduced new ways to store and manage data and Hadoop-related technologies gave rise to a new generation of database systems. Do not be confused: Hadoop and NoSQL databases are often discussed together since they are both components in addressing Big Data issues. However, Hadoop is neither a database nor a data model. It is a distributed file storing and processing model. There is no Hadoop DBMS. NoSQL databases are databases, and the NoSQL model represents a different way of approaching the storage and processing of data in a nonrelational way. NoSQL databases provide distributed, fault-tolerant databases for processing nonstructured data.

With the potential of big gains derived from Big Data analytics, it is not surprising that some organizations are turning to emerging Big Data technologies, such as NoSQL databases, to mine the wealth of information hidden in mountains of web data and gain a competitive advantage.

Note

Does this mean that relational databases don't have a place in organizations with Big Data challenges? No, relational databases remain the preferred and dominant databases to support most day-to-day transactions and structured data analytics needs. Each DBMS technology has its areas of application, and the best approach is to use the best tool for the job. In perspective, object/relational databases serve 98 percent of operational market needs. For Big Data needs, Hadoop and NoSQL databases are among the options. Chapter 14, Big Data and NoSQL, discusses these options in greater detail.

NoSQL
A new generation of database management systems that is not based on the traditional relational database model.

NoSQL Databases

Every time you search for a product on Amazon, send messages to friends in Facebook, watch a video on YouTube, or search for directions in Google Maps, you are using a NoSQL database. As with any new technology, the term *NoSQL* can be loosely applied to many different types of technologies. However, this chapter uses *NoSQL* to refer to a new generation of databases that address the specific challenges of the Big Data era and have the following general characteristics:

- They are not based on the relational model and SQL; hence the name NoSQL.
- They support highly distributed database architectures.

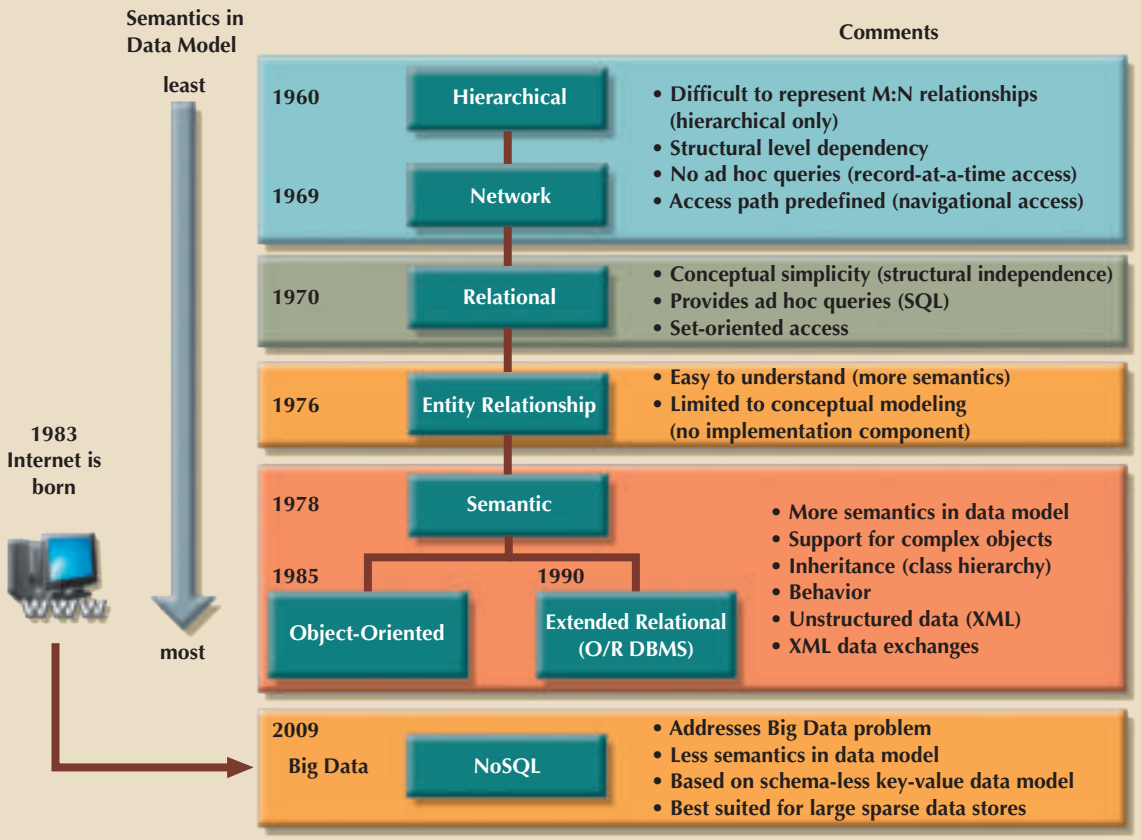
- They provide high scalability, high availability, and fault tolerance.
- They support very large amounts of sparse data (data with a large number of attributes but where the actual number of data instances is low).
- They are geared toward performance rather than transaction consistency.

Unlike the relational model, which provides a very comprehensive and cohesive approach to data storage and manipulation, the NoSQL model is a broad umbrella for a variety of approaches to data storage and manipulation. The most common of those approaches are key-value stores, document databases, columnar databases, and graph databases, as discussed in detail in Chapter 14.

2-5g Data Models: A Summary

The evolution of DBMSs has always been driven by the search for new ways of modeling and managing increasingly complex real-world data. A summary of the most commonly recognized data models is shown in Figure 2.5.

FIGURE 2.5 THE EVOLUTION OF DATA MODELS



In the evolution of data models, some common characteristics have made them widely accepted:

- A data model must show some degree of conceptual simplicity without compromising the semantic completeness of the database. *It does not make sense to have a data model that is more difficult to conceptualize than the real world.* At the same time, the model should show clarity and relevance; that is, the data model should be unambiguous and

applicable to the problem domain. A data model must represent the real world as closely as possible. This goal is more easily realized by adding more semantics to the model's data representation. (Semantics concern dynamic data behavior, while data representation constitutes the static aspect of the real-world scenario.) In other words, the model should be accurate and complete—all the needed data is included and properly described.

- Representation of the real-world transformations (behavior) must be in compliance with the consistency and integrity characteristics required by the intended use of the data model.

Each new data model addresses the shortcomings of previous models. The network model replaced the hierarchical model because the former made it much easier to represent complex (many-to-many) relationships. In turn, the relational model offers several advantages over the hierarchical and network models through its simpler data representation, superior data independence, and easy-to-use query language; these features have made it the preferred data model for business applications. The OO data model introduced support for complex data within a rich semantic framework. The ERDM added many OO features to the relational model and allowed it to maintain strong market share within the business environment. In recent years, the Big Data phenomenon has stimulated the development of alternative ways to model, store, and manage data that represents a break with traditional data management.

It is important to note that not all data models are created equal; some data models are better suited than others for some tasks. For example, *conceptual* models are better suited for high-level data modeling, while *implementation* models are better for managing stored data for implementation purposes. The ER model is an example of a conceptual model, while the hierarchical and network models are examples of implementation models. At the same time, some models, such as the relational model and the OODM, could be used as both conceptual and implementation models. Table 2.2 summarizes the advantages and disadvantages of the various database models.

Note

All databases assume the use of a common data pool within the database. Therefore, *all* database models promote data sharing, thus reducing the potential problem of islands of information.

Thus far, you have been introduced to the basic constructs of the more prominent data models. Each model uses such constructs to capture the meaning of the real-world data environment. Table 2.3 shows the basic terminology used by the various data models.

2-6 Degrees of Data Abstraction

If you ask 10 database designers what a data model is, you will end up with 10 different answers—depending on the degree of data abstraction. To illustrate the meaning of data abstraction, consider the example of automotive design. A car designer begins by drawing the concept of the car to be produced. Next, engineers design the details that help transfer the basic concept into a structure that can be produced. Finally, the engineering drawings are translated into production specifications to be used on the factory floor. As you can see, the process of producing the car begins at a high level of abstraction and proceeds to an ever-increasing level of detail. The factory floor process cannot proceed unless the engineering details are properly specified, and the engineering details cannot exist without the basic conceptual framework created by the designer. Designing a usable database follows the same basic process. That is, a database designer starts with an

TABLE 2.2

ADVANTAGES AND DISADVANTAGES OF VARIOUS DATABASE MODELS

DATA MODEL	DATA INDEPENDENCE	STRUCTURAL INDEPENDENCE	ADVANTAGES	DISADVANTAGES
Hierarchical	Yes	No	1. It promotes data sharing. 2. Parent/child relationship promotes conceptual simplicity. 3. Database security is provided and enforced by DBMS. 4. Parent/child relationship promotes data integrity. 5. It is efficient with 1:M relationships.	1. Complex implementation requires knowledge of physical data storage characteristics. 2. Navigational system yields complex application development, management, and use; requires knowledge of hierarchical path. 3. Changes in structure require changes in all application programs. 4. There are implementation limitations (no multiparent or M:N relationships). 5. There is no data definition or data manipulation language in the DBMS. 6. There is a lack of standards.
Network	Yes	No	1. Conceptual simplicity is at least equal to that of the hierarchical model. 2. It handles more relationship types, such as M:N and multiparent. 3. Data access is more flexible than in hierarchical and file system models. 4. Data owner/member relationship promotes data integrity. 5. There is conformance to standards. 6. It includes data definition language (DDL) and data manipulation language (DML) in DBMS.	1. System complexity limits efficiency—still a navigational system. 2. Navigational system yields complex implementation, application development, and management. 3. Structural changes require changes in all application programs.
Relational	Yes	Yes	1. Structural independence is promoted by the use of independent tables. Changes in a table's structure do not affect data access or application programs. 2. Tabular view substantially improves conceptual simplicity, thereby promoting easier database design, implementation, management, and use. 3. Ad hoc query capability is based on SQL. 4. Powerful RDBMS isolates the end user from physical-level details and improves implementation and management simplicity.	1. The RDBMS requires substantial hardware and system software overhead. 2. Conceptual simplicity gives relatively untrained people the tools to use a good system poorly, and if unchecked, it may produce the same data anomalies found in file systems. 3. It may promote islands of information problems as individuals and departments can easily develop their own applications.
Entity relationship	Yes	Yes	1. Visual modeling yields exceptional conceptual simplicity. 2. Visual representation makes it an effective communication tool. 3. It is integrated with the dominant relational model.	1. There is limited constraint representation. 2. There is limited relationship representation. 3. There is no data manipulation language. 4. Loss of information content occurs when attributes are removed from entities to avoid crowded displays. (This limitation has been addressed in subsequent graphical versions.)
Object-oriented	Yes	Yes	1. Semantic content is added. 2. Visual representation includes semantic content. 3. Inheritance promotes data integrity.	1. Slow development of standards caused vendors to supply their own enhancements, thus eliminating a widely accepted standard. 2. It is a complex navigational system. 3. There is a steep learning curve. 4. High system overhead slows transactions.
NoSQL	Yes	Yes	1. High scalability, availability, and fault tolerance are provided. 2. It uses low-cost commodity hardware. 3. It supports Big Data. 4. Key-value model improves storage efficiency.	1. Complex programming is required. 2. There is no relationship support—only by application code. 3. There is no transaction integrity support. 4. In terms of data consistency, it provides an eventually consistent model.

TABLE 2.3

DATA MODEL BASIC TERMINOLOGY COMPARISON

REAL WORLD	EXAMPLE	FILE PROCESSING	HIERARCHICAL MODEL	NETWORK MODEL	RELATIONAL MODEL	ER MODEL	OO MODEL
A group of vendors	Vendor file cabinet	File	Segment type	Record type	Table	Entity set	Class
A single vendor	Global supplies	Record	Segment occurrence	Current record	Row (tuple)	Entity occurrence	Object instance
The contact name	Johnny Ventura	Field	Segment field	Record field	Table attribute	Entity attribute	Object attribute
The vendor identifier	G12987	Index	Sequence field	Record key	Key	Entity identifier	Object identifier

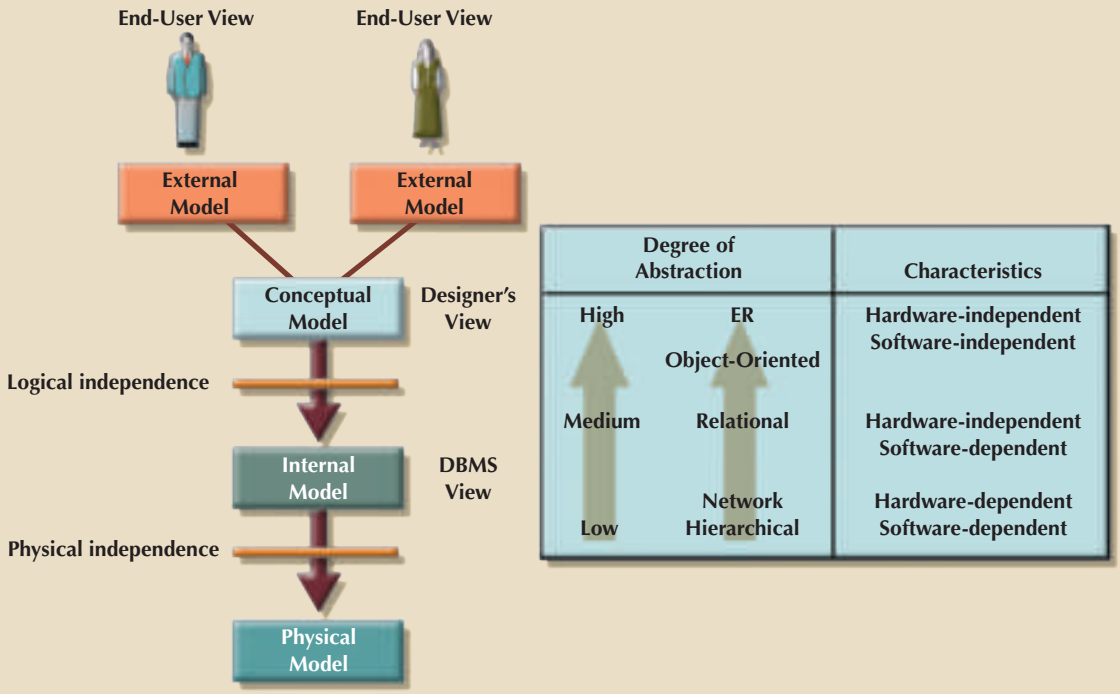
Note: For additional information about the terms used in this table, consult the corresponding chapters and online appendixes that accompany this book. For example, if you want to know more about the OO model, refer to Appendix G, Object-Oriented Databases.

American National Standards Institute (ANSI)
The group that accepted the DBTG recommendations and augmented database standards in 1975 through its SPARC committee.

abstract view of the overall data environment and adds details as the design comes closer to implementation. Using levels of abstraction can also be very helpful in integrating multiple (and sometimes conflicting) views of data at different levels of an organization.

In the early 1970s, the **American National Standards Institute (ANSI)** Standards Planning and Requirements Committee (SPARC) defined a framework for data modeling based on degrees of data abstraction. The resulting ANSI/SPARC architecture defines three levels of data abstraction: external, conceptual, and internal. You can use this framework to better understand database models, as shown in Figure 2.6. In the figure, the ANSI/SPARC framework has been expanded with the addition of a *physical* model to explicitly address physical-level implementation details of the internal model.

FIGURE 2.6 DATA ABSTRACTION LEVELS



2-6a The External Model

The **external model** is the end users' view of the data environment. The term *end users* refers to people who use the application programs to manipulate the data and generate information. End users usually operate in an environment in which an application has a specific business unit focus. Companies are generally divided into several business units, such as sales, finance, and marketing. Each business unit is subject to specific constraints and requirements, and each one uses a subset of the overall data in the organization. Therefore, end users within those business units view their data subsets as separate from or external to other units within the organization.

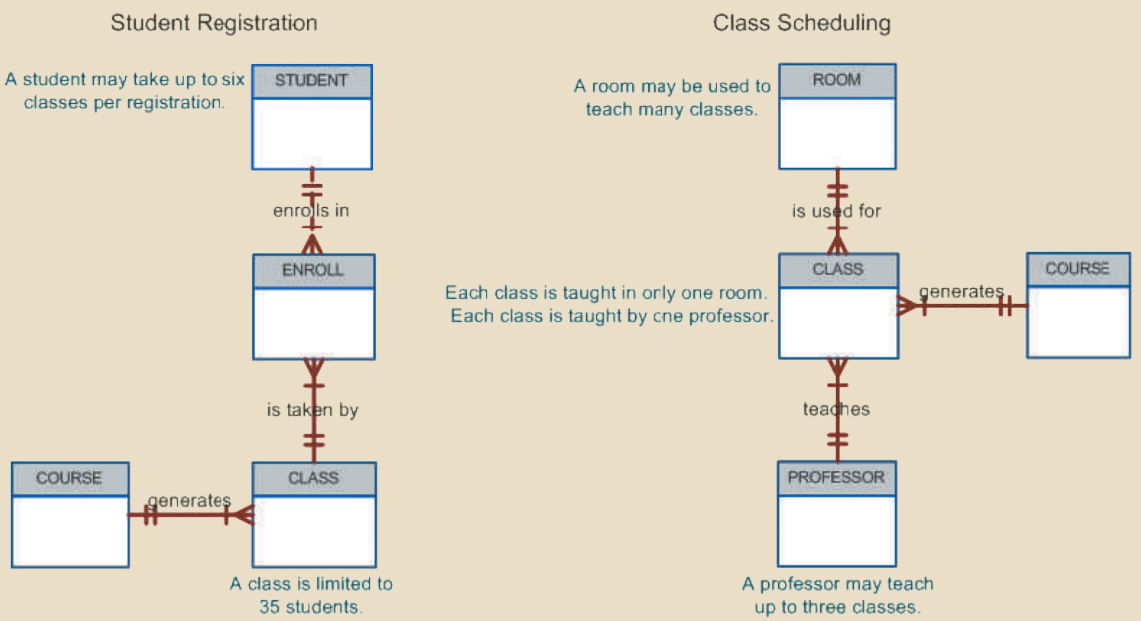
Because data is being modeled, ER diagrams will be used to represent the external views. A specific representation of an external view is known as an **external schema**. To illustrate the external model's view, examine the data environment of Tiny College.

Figure 2.7 presents the external schemas for two Tiny College business units: student registration and class scheduling. Each external schema includes the appropriate entities, relationships, processes, and constraints imposed by the business unit. Also note that *although the application views are isolated from each other, each view shares a common entity with the other view*. For example, the registration and scheduling external schemas share the entities CLASS and COURSE.

external model
The application programmer's view of the data environment. Given its business focus, an external model works with a data subset of the global database schema.

external schema
The specific representation of an external view; the end user's view of the data environment.

FIGURE 2.7 EXTERNAL MODELS FOR TINY COLLEGE



Note the ERs represented in Figure 2.7:

- A PROFESSOR may teach many CLASSes, and each CLASS is taught by only one PROFESSOR; there is a 1:M relationship between PROFESSOR and CLASS.
- A CLASS may ENROLL many students, and each STUDENT may ENROLL in many CLASSes, thus creating an M:N relationship between STUDENT and CLASS. (You will learn about the precise nature of the ENROLL entity in Chapter 4.)
- Each COURSE may generate many CLASSes, but each CLASS references a single COURSE. For example, there may be several classes (sections) of a database course that have a course code of CIS-420. One of those classes might be offered on MWF from 8:00 a.m. to 8:50 a.m., another might be offered on MWF from 1:00 p.m. to 1:50

p.m., while a third might be offered on Thursdays from 6:00 p.m. to 8:40 p.m. Yet, all three classes have the course code CIS-420.

- Finally, a CLASS requires one ROOM, but a ROOM may be scheduled for many CLASSes. That is, each classroom may be used for several classes: one at 9:00 a.m., one at 11:00 a.m., and one at 1:00 p.m., for example. In other words, there is a 1:M relationship between ROOM and CLASS.

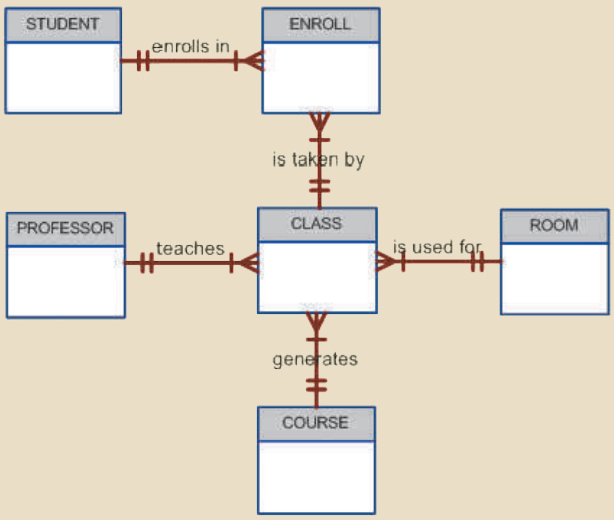
The use of external views that represent subsets of the database has some important advantages:

- It is easy to identify specific data required to support each business unit's operations.
- It makes the designer's job easy by providing feedback about the model's adequacy. Specifically, the model can be checked to ensure that it supports all processes as defined by their external models, as well as all operational requirements and constraints.
- It helps to ensure *security* constraints in the database design. Damaging an entire database is more difficult when each business unit works with only a subset of data.
- It makes application program development much simpler.

2-6b The Conceptual Model

The **conceptual model** represents a global view of the entire database by the entire organization. That is, the conceptual model integrates all external views (entities, relationships, constraints, and processes) into a single global view of the data in the enterprise, as shown in Figure 2.8. Also known as a **conceptual schema**, it is the basis for the identification and high-level description of the main data objects (avoiding any database model-specific details).

FIGURE 2.8 A CONCEPTUAL MODEL FOR TINY COLLEGE



conceptual model
The output of the conceptual design process. The conceptual model provides a global view of an entire database and describes the main data objects, avoiding details.

conceptual schema
A representation of the conceptual model, usually expressed graphically. See also *conceptual model*.

The most widely used conceptual model is the ER model. Remember that the ER model is illustrated with the help of the ERD, which is effectively the basic database blueprint. The ERD is used to graphically *represent* the conceptual schema.

The conceptual model yields some important advantages. First, it provides a bird's-eye (macro level) view of the data environment that is relatively easy to understand. For example, you can get a summary of Tiny College's data environment by examining the conceptual model in Figure 2.8.

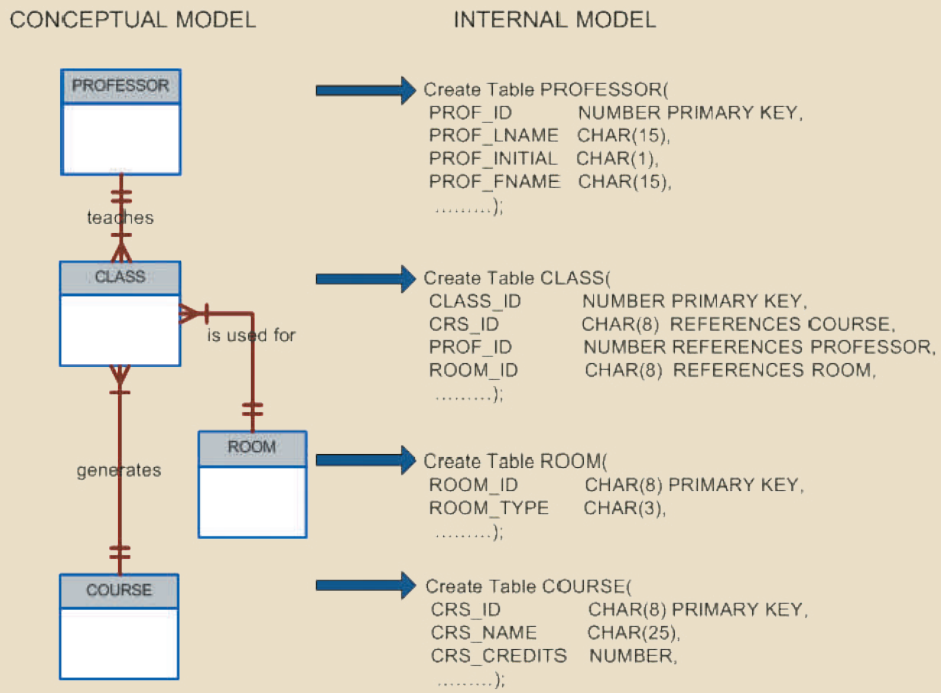
Second, the conceptual model is independent of both software and hardware. **Software independence** means that the model does not depend on the DBMS software used to implement the model. **Hardware independence** means that the model does not depend on the hardware used in the implementation of the model. Therefore, changes in either the hardware or the DBMS software will have no effect on the database design at the conceptual level. Generally, the term **logical design** refers to the task of creating a conceptual data model that could be implemented in any DBMS.

2-6c The Internal Model

Once a specific DBMS has been selected, the internal model maps the conceptual model to the DBMS. The **internal model** is the representation of the database as "seen" by the DBMS. In other words, the internal model requires the designer to match the conceptual model's characteristics and constraints to those of the selected implementation model. An **internal schema** depicts a specific representation of an internal model, using the database constructs supported by the chosen database.

Because this book focuses on the relational model, a relational database was chosen to implement the internal model. Therefore, the internal schema should map the conceptual model to the relational model constructs. In particular, the entities in the conceptual model are mapped to tables in the relational model. Likewise, because a relational database has been selected, the internal schema is expressed using SQL, the standard language for relational databases. In the case of the conceptual model for Tiny College depicted in Figure 2.8, the internal model was implemented by creating the tables PROFESSOR, COURSE, CLASS, STUDENT, ENROLL, and ROOM. A simplified version of the internal model for Tiny College is shown in Figure 2.9.

FIGURE 2.9 INTERNAL MODEL FOR TINY COLLEGE



The development of a detailed internal model is especially important to database designers who work with hierarchical or network models because those models require precise specification of data storage location and data access paths. In contrast,

software independence
A property of any model or application that does not depend on the software used to implement it.

hardware independence
A condition in which a model does not depend on the hardware used in the model's implementation. Therefore, changes in the hardware will have no effect on the database design at the conceptual level.

logical design
A stage in the design phase that matches the conceptual design to the requirements of the selected DBMS and is therefore software-dependent. Logical design is used to translate the conceptual design into the internal model for a selected database management system, such as DB2, SQL Server, Oracle, IMS, Informix, Access, or Ingress.

internal model
In database modeling, a level of data abstraction that adapts the conceptual model to a specific DBMS model for implementation. The internal model is the representation of a database as "seen" by the DBMS. In other words, the internal model requires a designer to match the conceptual model's characteristics and constraints to those of the selected implementation model.

internal schema
A representation of an internal model using the database constructs supported by the chosen database.

the relational model requires less detail in its internal model because most RDBMSs handle data access path definition *transparently*; that is, the designer need not be aware of the data access path details. Nevertheless, even relational database software usually requires specifications of data storage locations, especially in a mainframe environment. For example, DB2 requires that you specify the data storage group, the location of the database within the storage group, and the location of the tables within the database.

Because the internal model depends on specific database software, it is said to be software dependent. Therefore, a change in the DBMS software requires that the internal model be changed to fit the characteristics and requirements of the implementation database model. When you can change the internal model without affecting the conceptual model, you have **logical independence**. However, the internal model is still hardware independent because it is unaffected by the type of computer on which the software is installed. Therefore, a change in storage devices or even a change in operating systems will not affect the internal model.

logical independence
A condition in which the internal model can be changed without affecting the conceptual model. (The internal model is hardware-independent because it is unaffected by the computer on which the software is installed. Therefore, a change in storage devices or operating systems will not affect the internal model.)

physical model
A model in which physical characteristics such as location, path, and format are described for the data. The physical model is both hardware- and software-dependent. See also *physical design*.

physical independence
A condition in which the physical model can be changed without affecting the internal model.

2-6d The Physical Model

The **physical model** operates at the lowest level of abstraction, describing the way data is saved on storage media such as magnetic, solid state, or optical media. The physical model requires the definition of both the physical storage devices and the (physical) access methods required to reach the data within those storage devices, making it both software and hardware dependent. The storage structures used are dependent on the software (the DBMS and the operating system) and on the type of storage devices the computer can handle. The precision required in the physical model's definition demands that database designers have a detailed knowledge of the hardware and software used to implement the database design.

Early data models forced the database designer to take the details of the physical model's data storage requirements into account. However, the now-dominant relational model is aimed largely at the logical level rather than at the physical level; therefore, it does not require the physical-level details common to its predecessors.

Although the relational model does not require the designer to be concerned about the data's physical storage characteristics, the *implementation* of a relational model may require physical-level fine-tuning for increased performance. Fine-tuning is especially important when very large databases are installed in a mainframe environment, yet even such performance fine-tuning at the physical level does not require knowledge of physical data storage characteristics.

As noted earlier, the physical model is dependent on the DBMS, methods of accessing files, and types of hardware storage devices supported by the operating system. When you can change the physical model without affecting the internal model, you have **physical independence**. Therefore, a change in storage devices or methods and even a change in operating system will not affect the internal model.

The levels of data abstraction are summarized in Table 2.4.

TABLE 2.4

LEVELS OF DATA ABSTRACTION

MODEL	DEGREE OF ABSTRACTION	FOCUS	INDEPENDENT OF
External	High	End-user views	Hardware and software
Conceptual	↕	Global view of data (database model independent)	Hardware and software
Internal	↕	Specific database model	Hardware
Physical	Low	Storage and access methods	Neither hardware nor software

Summary

- A data model is an abstraction of a complex real-world data environment. Database designers use data models to communicate with programmers and end users. The basic data-modeling components are entities, attributes, relationships, and constraints. Business rules are used to identify and define the basic modeling components within a specific real-world environment.
- The hierarchical and network data models were early models that are no longer used, but some of the concepts are found in current data models.
- The relational model is the current database implementation standard. In the relational model, the end user perceives the data as being stored in tables. Tables are related to each other by means of common values in common attributes. The entity relationship (ER) model is a popular graphical tool for data modeling that complements the relational model. The ER model allows database designers to visually present different views of the data—as seen by database designers, programmers, and end users—and to integrate the data into a common framework.
- The object-oriented data model (OODM) uses objects as the basic modeling structure. Like the relational model's entity, an object is described by its factual content. Unlike an entity, however, the object also includes information about relationships between the facts, as well as relationships with other objects, thus giving its data more meaning.
- The relational model has adopted many object-oriented (OO) extensions to become the extended relational data model (ERDM). Object/relational database management systems (O/R DBMS) were developed to implement the ERDM. At this point, the OODM is largely used in specialized engineering and scientific applications, while the ERDM is primarily geared to business applications.
- Big Data technologies such as Hadoop, MapReduce, and NoSQL provide distributed, fault-tolerant, and cost-efficient support for Big Data analytics. NoSQL databases are a new generation of databases that do not use the relational model and are geared to support the very specific needs of Big Data organizations. NoSQL databases offer distributed data stores that provide high scalability, availability, and fault tolerance by sacrificing data consistency and shifting the burden of maintaining relationships and data integrity to the program code.
- Data-modeling requirements are a function of different data views (global versus local) and the level of data abstraction. The American National Standards Institute Standards Planning and Requirements Committee (ANSI/SPARC) describes three levels of data abstraction: external, conceptual, and internal. The fourth and lowest level of data abstraction, called the physical level, is concerned exclusively with physical storage methods.

Key Terms

3 Vs
American National Standards Institute (ANSI)
attribute
Big Data
business rule
Chen notation
class
class diagram
class diagram notation
class hierarchy
client node
conceptual model
conceptual schema
connectivity
constraint
Crow's Foot notation
data definition language (DDL)
data manipulation language (DML)
data model
data modeling
data node
entity
entity instance
entity occurrence
entity relationship (ER) model (ERM)
entity relationship diagram (ERD)
entity set
extended relational data model (ERDM)
Extensible Markup Language (XML)
external model
external schema
Hadoop
Hadoop Distributed File System (HDFS)
hardware independence
hierarchical model
inheritance
internal model
internal schema
logical design
logical independence
MapReduce
many-to-many (M:N or *..*) relationship
method
name node
network model
NoSQL
object
object/relational database management system (O/R DBMS)
object-oriented data model (OODM)
object-oriented database management system (OODBMS)
one-to-many (1:M or 1..*) relationship
one-to-one (1:1 or 1..1) relationship
physical independence
physical model
relation
relational database management system (RDBMS)
relational diagram
relational model
relationship
schema
segment
semantic data model
software independence
subschema
table
tuple
Unified Modeling Language (UML)

Review Questions

1. Discuss the importance of data models.
2. What is a business rule, and what is its purpose in data modeling?
3. How do you translate business rules into data model components?
4. Describe the basic features of the relational data model and discuss their importance to the end user and the designer.

5. Explain how the entity relationship (ER) model helped produce a more structured relational database design environment.
6. Consider the scenario described by the statement "A customer can make many payments, but each payment is made by only one customer." Use this scenario as the basis for an entity relationship diagram (ERD) representation.
7. Why is an object said to have greater semantic content than an entity?
8. What is the difference between an object and a class in the object-oriented data model (OODM)?
9. How would you model Question 6 with an OODM? (Use Figure 2.4 as your guide.)
10. What is an ERDM, and what role does it play in the modern (production) database environment?
11. What is a relationship, and what three types of relationships exist?
12. Give an example of each of the three types of relationships.
13. What is a table, and what role does it play in the relational model?
14. What is a relational diagram? Give an example.
15. What is connectivity? (Use a Crow's Foot ERD to illustrate connectivity.)
16. Describe the Big Data phenomenon.
17. What does the term *3 Vs* refer to?
18. What is Hadoop, and what are its basic components?
19. What are the basic characteristics of a NoSQL database?
20. Using the example of a medical clinic with patients and tests, provide a simple representation of how to model this example using the relational model and how it would be represented using the key-value data modeling technique.
21. What is logical independence?
22. What is physical independence?

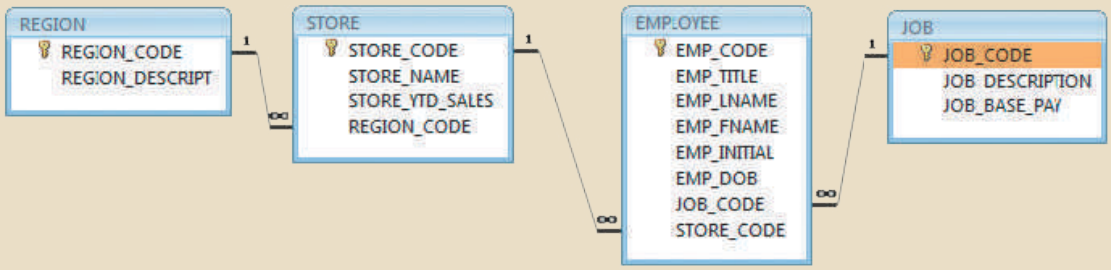
Problems

Use the contents of Figure 2.1 to work Problems 1–3.

1. Write the business rule(s) that govern the relationship between AGENT and CUSTOMER.
2. Given the business rule(s) you wrote in Problem 1, create the basic Crow's Foot ERD.
3. Using the ERD you drew in Problem 2, create the equivalent object representation and UML class diagram. (Use Figure 2.4 as your guide.)

Using Figure P2.4 as your guide, work Problems 4–5. The DealCo relational diagram shows the initial entities and attributes for the DealCo stores, which are located in two regions of the country.

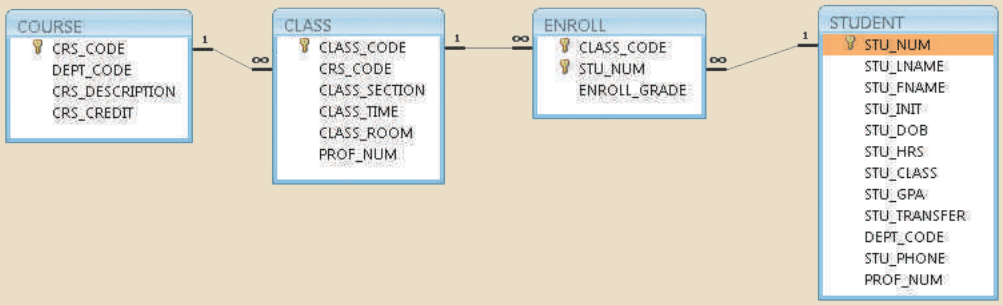
FIGURE P2.4 THE DEALCO RELATIONAL DIAGRAM



4. Identify each relationship type and write all of the business rules.
5. Create the basic Crow's Foot ERD for DealCo.

Using Figure P2.6 as your guide, work Problems 6–8. The Tiny College relational diagram shows the initial entities and attributes for the college.

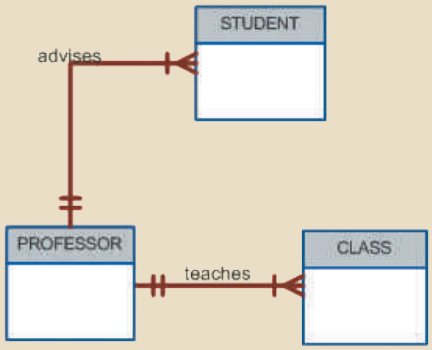
FIGURE P2.6 THE TINY COLLEGE RELATIONAL DIAGRAM



6. Identify each relationship type and write all of the business rules.
7. Create the basic Crow's Foot ERD for Tiny College.
8. Create the UML class diagram that reflects the entities and relationships you identified in the relational diagram.
9. Typically, a hospital patient receives medications that have been ordered by a particular doctor. Because the patient often receives several medications per day, there is a 1:M relationship between PATIENT and ORDER. Similarly, each order can include several medications, creating a 1:M relationship between ORDER and MEDICATION.
   a. Identify the business rules for PATIENT, ORDER, and MEDICATION.
   b. Create a Crow's Foot ERD that depicts a relational database model to capture these business rules.
10. United Broke Artists (UBA) is a broker for not-so-famous artists. UBA maintains a small database to track painters, paintings, and galleries. A painting is created by a particular artist and then exhibited in a particular gallery. A gallery can exhibit many paintings, but each painting can be exhibited in only one gallery. Similarly, a painting is created by a single painter, but each painter can create many paintings. Using PAINTER, PAINTING, and GALLERY, in terms of a relational database:
    a. What tables would you create, and what would the table components be?
    b. How might the (independent) tables be related to one another?

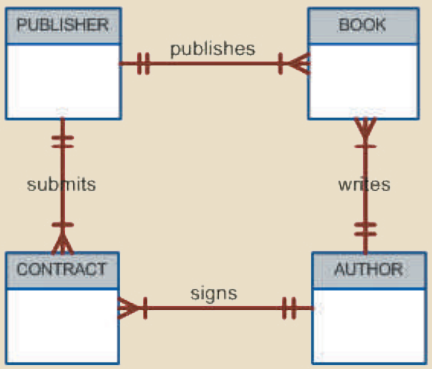
11. Using the ERD from Problem 10, create the relational schema. (Create an appropriate collection of attributes for each of the entities. Make sure you use the appropriate naming conventions to name the attributes.)
12. Convert the ERD from Problem 10 into a corresponding UML class diagram.
13. Describe the relationships (identify the business rules) depicted in the Crow's Foot ERD shown in Figure P2.13.

FIGURE P2.13 THE CROW'S FOOT ERD FOR PROBLEM 13



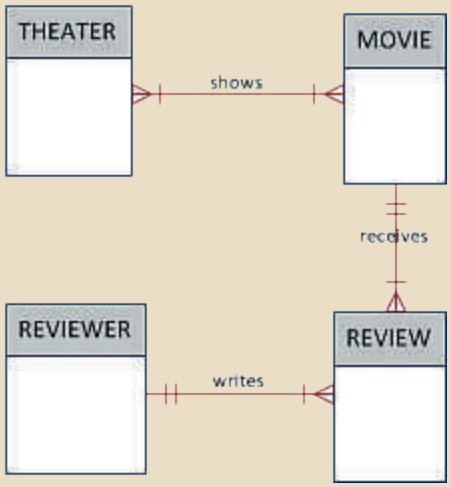
14. Create a Crow's Foot ERD to include the following business rules for the ProdCo company:
    a. Each sales representative writes many invoices.
    b. Each invoice is written by one sales representative.
    c. Each sales representative is assigned to one department.
    d. Each department has many sales representatives.
    e. Each customer can generate many invoices.
    f. Each invoice is generated by one customer.
15. Write the business rules that are reflected in the ERD shown in Figure P2.15. (Note that the ERD reflects some simplifying assumptions. For example, each book is written by only one author. Also, remember that the ERD is always read from the "1" to the "M" side, regardless of the orientation of the ERD components.)

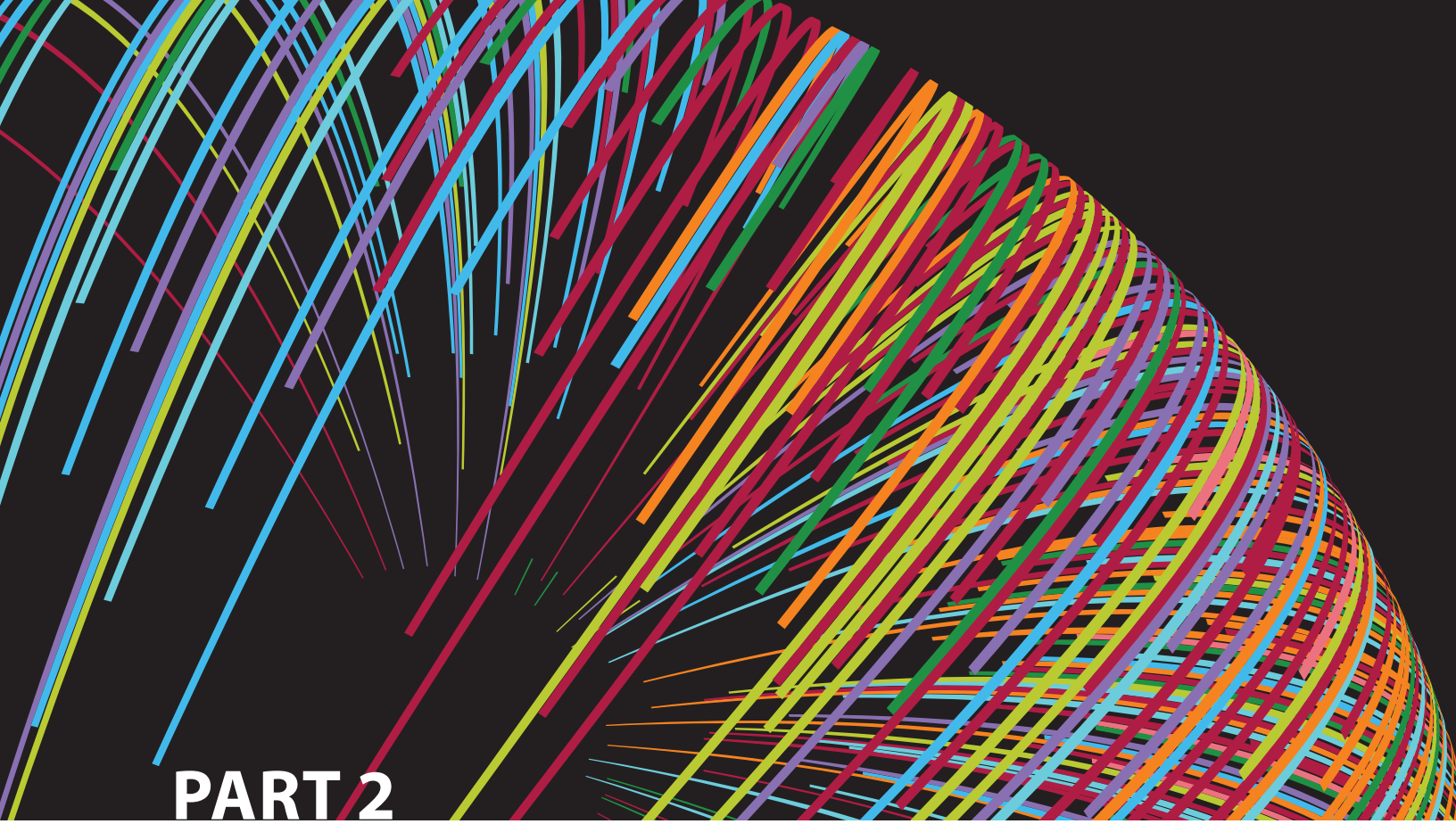
FIGURE P2.15 THE CROW'S FOOT ERD FOR PROBLEM 15



16. Create a Crow's Foot ERD for each of the following descriptions. (Note that the word *many* merely means *more than one* in the database modeling environment.)
    a. Each of the MegaCo Corporation's divisions is composed of many departments. Each department has many employees assigned to it, but each employee works for only one department. Each department is managed by one employee, and each of those managers can manage only one department at a time.
    b. During some period of time, a customer can download many ebooks from BooksOnline. Each of the ebooks can be downloaded by many customers during that period of time.
    c. An airliner can be assigned to fly many flights, but each flight is flown by only one airliner.
    d. The KwikTite Corporation operates many factories. Each factory is located in a region, and each region can be "home" to many of KwikTite's factories. Each factory has many employees, but each employee is employed by only one factory.
    e. An employee may have earned many degrees, and each degree may have been earned by many employees.
17. Write the business rules that are reflected in the ERD shown in Figure P2.17.

FIGURE P2.17 THE CROW'S FOOT ERD FOR PROBLEM 17





PART 2
Design Concepts

3 The Relational Database Model

4 Entity Relationship (ER) Modeling

5 Advanced Data Modeling

6 Normalization of Database Tables

Chapter 3

The Relational Database Model

After completing this chapter, you will be able to:

- Describe the relational database model's logical structure
- Identify the relational model's basic components and explain the structure, contents, and characteristics of a relational table
- Use relational database operators to manipulate relational table contents
- Explain the purpose and components of the data dictionary and system catalog
- Identify appropriate entities and then the relationships among the entities in the relational database model
- Describe how data redundancy is handled in the relational database model
- Explain the purpose of indexing in a relational database

Preview

In this chapter, you will learn about the relational model's logical structure and more about how entity relationship diagrams (ERDs) can be used to design a relational database. You will also learn how the relational database's basic data components fit into a logical construct known as a table, and how tables within a database can be related to one another.

After learning about tables, their components, and their relationships, you will be introduced to basic table design concepts and the characteristics of well-designed and poorly designed tables. These concepts will become your gateway to the next few chapters.

Data Files and Available Formats

	MS Access	Oracle	MS SQL	My SQL		MS Access	Oracle	MS SQL	My SQL
CH03_CollegeTry	✓	✓	✓	✓	CH03_AviaCo	✓	✓	✓	✓
CH03_CollegeTry2	✓	✓	✓	✓	CH03_BeneCo	✓	✓	✓	✓
CH03_InsureCo	✓	✓	✓	✓	CH03_CollegeQue	✓	✓	✓	✓
CH03_Museum	✓	✓	✓	✓	CH03_NoComp	✓	✓	✓	✓
CH03_SaleCo	✓	✓	✓	✓	CH03_StoreCo	✓	✓	✓	✓
CH03_TinyCollege	✓	✓	✓	✓	CH03_Theater	✓	✓	✓	✓
CH03_Relational_DB	✓	✓	✓	✓	CH03_TransCo	✓	✓	✓	✓
					CH03_VendingCo	✓	✓	✓	✓

Data Files Available on cengagebrain.com

Note

The relational model, introduced by E. F. Codd in 1970, is based on predicate logic and set theory. **Predicate logic**, used extensively in mathematics, provides a framework in which an assertion (statement of fact) can be verified as either true or false. For example, suppose that a student with a student ID of 12345678 is named Melissa Sanduski. This assertion can easily be demonstrated to be true or false. **Set theory** is a mathematical science that deals with sets, or groups of things, and is used as the basis for data manipulation in the relational model. For example, assume that set A contains three numbers: 16, 24, and 77. This set is represented as A(16, 24, 77). Furthermore, set B contains four numbers, 44, 77, 90, and 11, and so is represented as B(44, 77, 90, 11). Given this information, you can conclude that the intersection of A and B yields a result set with a single number, 77. This result can be expressed as A ∩ B = 77. In other words, A and B share a common value, 77.

Based on these concepts, the relational model has three well-defined components:

1. A logical data structure represented by relations (see Sections 3-1, 3-2, and 3-5)
2. A set of integrity rules to enforce that the data is consistent and remains consistent over time (see Sections 3-3, 3-6, 3-7, and 3-8)
3. A set of operations that defines how data is manipulated (see Section 3-4)

3-1 A Logical View of Data

In Chapter 1, Database Systems, you learned that a database stores and manages both data and metadata. You also learned that the DBMS manages and controls access to the data and the database structure. Such an arrangement—placing the DBMS between the application and the database—eliminates most of the file system's inherent limitations. The result of such flexibility, however, is a far more complex physical structure. In fact, the database structures required by both the hierarchical and network database models often become complicated enough to diminish efficient database design. The relational data model changed all of that by allowing the designer to focus on the logical representation of the data and its relationships, rather than on the physical storage details. To use an automotive analogy, the relational database uses an automatic transmission to relieve you of the need to manipulate clutch pedals and gearshifts. In short, the relational model enables you to view data *logically* rather than *physically*.

The practical significance of taking the logical view is that it serves as a reminder of the simple file concept of data storage. Although the use of a table, quite unlike that of a file, has the advantages of structural and data independence, a table does resemble a file from a conceptual point of view. Because you can think of related records as being stored in independent tables, the relational database model is much easier to understand than the hierarchical and network models. Logical simplicity tends to yield simple and effective database design methodologies.

Because the table plays such a prominent role in the relational model, it deserves a closer look. Therefore, our discussion begins by exploring the details of table structure and contents.

predicate logic
Used extensively in mathematics to provide a framework in which an assertion (statement of fact) can be verified as either true or false.

set theory
A part of mathematical science that deals with sets, or groups of things, and is used as the basis for data manipulation in the relational model.

3-1a Tables and Their Characteristics

The logical view of the relational database is facilitated by the creation of data relationships based on a logical construct known as a relation. Because a relation is a mathematical construct, end users find it much easier to think of a relation as a table. A *table* is perceived as a two-dimensional structure composed of rows and columns. A table is also

called a *relation* because the relational model's creator, E. F. Codd, used the two terms as synonyms. You can think of a table as a *persistent* representation of a logical relation—that is, a relation whose contents can be permanently saved for future use. As far as the table's user is concerned, a table contains *a group of related entity occurrences*—that is, an entity set. For example, a STUDENT table contains a collection of entity occurrences, each representing a student. For that reason, the terms *entity set* and *table* are often used interchangeably.

Note

The word *relation*, also known as a *dataset* in Microsoft Access, is based on the mathematical set theory from which Codd derived his model. Because the relational model uses attribute values to establish relationships among tables, many database users incorrectly assume that the term *relation* refers to such relationships. Many then incorrectly conclude that only the relational model permits the use of relationships.

You will discover that the table view of data makes it easy to spot and define entity relationships, thereby greatly simplifying the task of database design. The characteristics of a relational table are summarized in Table 3.1.

TABLE 3.1

CHARACTERISTICS OF A RELATIONAL TABLE

1	A table is perceived as a two-dimensional structure composed of rows and columns.
2	Each table row (**tuple**) represents a single entity occurrence within the entity set.
3	Each table column represents an attribute, and each column has a distinct name.
4	Each intersection of a row and column represents a single data value.
5	All values in a column must conform to the same data format.
6	Each column has a specific range of values known as the **attribute domain**.
7	The order of the rows and columns is immaterial to the DBMS.
8	Each table must have an attribute or combination of attributes that uniquely identifies each row.

Note

Relational database terminology is very precise. Unfortunately, file system terminology sometimes creeps into the database environment. Thus, rows are sometimes referred to as *records*, and columns are sometimes labeled as *fields*. Occasionally, tables are labeled *files*. Technically speaking, this substitution of terms is not always appropriate. The database table is a logical concept rather than a physical concept, and the terms *file*, *record*, and *field* describe physical concepts. Nevertheless, as long as you recognize that the table is actually a logical concept rather than a physical construct, you may think of table rows as records and table columns as fields. In fact, many database software vendors still use this familiar file system terminology.

tuple
In the relational model, a table row.

attribute domain
In data modeling, the construct used to organize and describe an attribute's set of possible values.

The database table shown in Figure 3.1 illustrates the characteristics listed in Table 3.1.

FIGURE 3.1 STUDENT TABLE ATTRIBUTE VALUES

Table name: STUDENT **Database name: Ch03_TinyCollege**

STU_NUM	STU_LNAME	STU_FNAME	STU_INIT	STU_DOB	STU_HRS	STU_CLASS	STU_GPA	STU_TRANSFER	DEPT_CODE	STU_PHONE	PROF_NUM
321452	Bowser	William	C	12-Feb-1985	42	So	2.84	No	BIOL	2134	205
324257	Smithson	Anne	K	15-Nov-1991	81	Jr	3.27	Yes	CIS	2256	222
324258	Brewer	Juliette		23-Aug-1979	36	So	2.26	Yes	ACCT	2256	228
324269	Oblonski	Walter	H	16-Sep-1986	66	Jr	3.09	No	CIS	2114	222
324273	Smith	John	D	30-Dec-1968	102	Sr	2.11	Yes	ENGL	2231	199
324274	Katinga	Raphael	P	21-Oct-1989	114	Sr	3.15	No	ACCT	2267	228
324291	Robertson	Gerald	T	08-Apr-1983	120	Sr	3.87	No	EDU	2267	311
324299	Smith	John	B	30-Nov-1996	15	Fr	2.92	No	ACCT	2315	230

STU_NUM = **Student number**
STU_LNAME = **Student last name**
STU_FNAME = **Student first name**
STU_INIT = **Student middle initial**
STU_DOB = **Student date of birth**
STU_HRS = **Credit hours earned**
STU_CLASS = **Student classification**
STU_GPA = **Grade point average**
STU_TRANSFER = **Student transferred from another institution**
DEPT_CODE = **Department code**
STU_PHONE = **4-digit campus phone extension**
PROF_NUM = **Number of the professor who is the student's advisor**

Using the STUDENT table shown in Figure 3.1, you can draw the following conclusions corresponding to the points in Table 3.1:

1. The STUDENT table is perceived to be a two-dimensional structure composed of 8 rows (tuples) and 12 columns (attributes).
2. Each row in the STUDENT table describes a single entity occurrence within the entity set. (The entity set is represented by the STUDENT table.) For example, row 4 in Figure 3.1 describes a student named Walter H. Oblonski. Given the table contents, the STUDENT entity set includes eight distinct entities (rows) or students.
3. Each column represents an attribute, and each column has a distinct name.
4. All of the values in a column match the attribute's characteristics. For example, the grade point average (STU_GPA) column contains only STU_GPA entries for each of the table rows. Data must be classified according to its format and function. Although various DBMSs can support different data types, most support at least the following:
   a. *Numeric*. You can use numeric data to perform meaningful arithmetic procedures. For example, in Figure 3.1, STU_HRS and STU_GPA are numeric attributes.
   b. *Character*. Character data, also known as text data or string data, can contain any character or symbol not intended for mathematical manipulation. In Figure 3.1, STU_CLASS and STU_PHONE are examples of character attributes.
   c. *Date*. Date attributes contain calendar dates stored in a special format known as the Julian date format. In Figure 3.1, STU_DOB is a date attribute.
   d. Logical. Logical data can only have true or false (yes or no) values. In Figure 3.1, the STU_TRANSFER attribute uses a logical data format.
5. The column's range of permissible values is known as its domain. Because the STU_GPA values are limited to the range 0–4, inclusive, the domain is [0,4].
6. The order of rows and columns is immaterial to the user.

Online Content

The databases used to illustrate the material in this chapter (see the Data Files list at the beginning of the chapter) are available at *www.cengagebrain.com*. The database names match the database names shown in the figures.

7. Each table must have a primary key. In general terms, the **primary key (PK)** is an attribute or combination of attributes that uniquely identifies any given row. In this case, STU_NUM (the student number) is the primary key. Using the data in Figure 3.1, observe that a student's last name (STU_LNAME) would not be a good primary key because several students have the last name of Smith. Even the combination of the last name and first name (STU_FNAME) would not be an appropriate primary key because more than one student is named John Smith.

primary key (PK)
In the relational model, an identifier composed of one or more attributes that uniquely identifies a row. Also, a candidate key selected as a unique entity identifier. See also *key*.

key
One or more attributes that determine other attributes. See also *candidate key*, *foreign key*, *primary key (PK)*, *secondary key*, and *superkey*.

determination
The role of a key. In the context of a database table, the statement "A determines B" indicates that knowing the value of attribute A means that the value of attribute B can be looked up.

functional dependence
Within a relation R, an attribute B is functionally dependent on an attribute A if and only if a given value of attribute A determines exactly one value of attribute B. The relationship "B is dependent on A" is equivalent to "A determines B" and is written as A → B.

determinant
Any attribute in a specific row whose value directly determines other values in that row. See also *Boyce-Codd normal form (BCNF)*.

dependent
An attribute whose value is determined by another attribute.

3-2 Keys

In the relational model, keys are important because they are used to ensure that each row in a table is uniquely identifiable. They are also used to establish relationships among tables and to ensure the integrity of the data. A **key** consists of one or more attributes that determine other attributes. For example, an invoice number identifies all of the invoice attributes, such as the invoice date and the customer name.

One type of key, the primary key, has already been introduced. Given the structure of the STUDENT table shown in Figure 3.1, defining and describing the primary key seem simple enough. However, because the primary key plays such an important role in the relational environment, you will examine the primary key's properties more carefully. In this section, you also will become acquainted with superkeys, candidate keys, and secondary keys.

3-2a Dependencies

The role of a key is based on the concept of determination. **Determination** is the state in which knowing the value of one attribute makes it possible to determine the value of another. The idea of determination is not unique to the database environment. You are familiar with the formula revenue − cost = profit. This is a form of determination, because if you are given the *revenue* and the *cost*, you can determine the *profit*. Given *profit* and *revenue*, you can determine the *cost*. Given any two values, you can determine the third. Determination in a database environment, however, is not normally based on a formula but on the relationships among the attributes.

If you consider what the attributes of the STUDENT table in Figure 3.1 actually represent, you will see a relationship among the attributes. If you are given a value for STU_NUM, then you can determine the value for STU_LNAME because one and only one value of STU_LNAME is associated with any given value of STU_NUM. A specific terminology and notation is used to describe relationships based on determination. The relationship is called **functional dependence**, which means that the value of one or more attributes determines the value of one or more other attributes. The standard notation for representing the relationship between STU_NUM and STU_LNAME is as follows:

STU_NUM → STU_LNAME

In this functional dependency, the attribute whose value determines another is called the **determinant** or the key. The attribute whose value is determined by the other attribute is called the **dependent**. Using this terminology, it would be correct to say that STU_NUM is the determinant and STU_LNAME is the dependent. STU_NUM functionally determines STU_LNAME, and STU_LNAME is functionally dependent on STU_NUM. As stated earlier, functional dependence can involve a determinant that comprises more than one attribute and multiple dependent attributes. Refer to the STUDENT table for the following example:

STU_NUM → (STU_LNAME, STU_FNAME, STU_GPA)

and

(STU_FNAME, STU_LNAME, STU_INIT, STU_PHONE) → (STU_DOB, STU_HRS, STU_GPA)

Determinants made of more than one attribute require special consideration. It is possible to have a functional dependency in which the determinant contains attributes that are not necessary for the relationship. Consider the following two functional dependencies:

STU_NUM → STU_GPA

(STU_NUM, STU_LNAME) → STU_GPA

In the second functional dependency, the determinant includes STU_LNAME, but this attribute is not necessary for the relationship. The functional dependency is valid because given a pair of values for STU_NUM and STU_LNAME, only one value would occur for STU_GPA. A more specific term, **full functional dependence**, is used to refer to functional dependencies in which the entire collection of attributes in the determinant is necessary for the relationship. Therefore, the dependency shown in the preceding example is a functional dependency, but not a full functional dependency.

3-2b Types of Keys

Recall that a key is an attribute or group of attributes that can determine the values of other attributes. Therefore, keys are determinants in functional dependencies. Several different types of keys are used in the relational model, and you need to be familiar with them.

A **composite key** is a key that is composed of more than one attribute. An attribute that is a part of a key is called a **key attribute**. For example,

STU_NUM → STU_GPA

(STU_LNAME, STU_FNAME, STU_INIT, STU_PHONE) → STU_HRS

In the first functional dependency, STU_NUM is an example of a key composed of only one key attribute. In the second functional dependency, (STU_LNAME, STU_FNAME, STU_INIT, STU_PHONE) is a composite key composed of four key attributes.

A **superkey** is a key that can uniquely identify any row in the table. In other words, a superkey functionally determines every attribute in the row. In the STUDENT table, STU_NUM is a superkey, as are the composite keys (STU_NUM, STU_LNAME), (STU_NUM, STU_LNAME, STU_INIT) and (STU_LNAME, STU_FNAME, STU_INIT, STU_PHONE). In fact, because STU_NUM alone is a superkey, any composite key that has STU_NUM as a key attribute will also be a superkey. Be careful, however, because not all keys are superkeys. For example, Gigantic State University determines its student classification based on hours completed, as shown in Table 3.2.

Therefore, you can write STU_HRS → STU_CLASS.

However, the specific number of hours is not dependent on the classification. It is quite possible to find a junior with 62 completed hours or one with 84 completed hours. In other words, the classification (STU_CLASS) does not determine one and only one value for completed hours (STU_HRS).

full functional dependence
A condition in which an attribute is functionally dependent on a composite key but not on any subset of the key.

composite key
A multiple-attribute key.

key attribute
An attribute that is part of a primary key. See also *prime attribute*.

superkey
An attribute or attributes that uniquely identify each entity in a table. See *key*.

TABLE 3.2

STUDENT CLASSIFICATION

HOURS COMPLETED	CLASSIFICATION
Less than 30	Fr
30–59	So
60–89	Jr
90 or more	Sr

One specific type of superkey is called a candidate key. A **candidate key** is a minimal superkey—that is, a superkey without any unnecessary attributes. A candidate key is based on a full functional dependency. For example, STU_NUM would be a candidate key, as would (STU_LNAME, STU_FNAME, STU_INIT, STU_PHONE). On the other hand, (STU_NUM, STU_LNAME) is a superkey, but it is not a candidate key because STU_LNAME could be removed and the key would still be a superkey. A table can have many different candidate keys. If the STUDENT table also included the students' Social Security numbers as STU_SSN, then it would appear to be a candidate key. Candidate keys are called *candidates* because they are the eligible options from which the designer will choose when selecting the primary key. The primary key is the candidate key chosen to be the primary means by which the rows of the table are uniquely identified.

Entity integrity is the condition in which each row (entity instance) in the table has its own unique identity. To ensure entity integrity, the primary key has two requirements: (1) all of the values in the primary key must be unique and (2) no key attribute in the primary key can contain a null.

Note

A null is no value at all. It does *not* mean a zero or a space. A null is created when you press the Enter key or the Tab key to move to the next entry without making an entry of any kind. Pressing the Spacebar creates a blank (or a space).

Null values are problematic in the relational model. A **null** is the absence of any data value, and it is never allowed in any part of the primary key. From a theoretical perspective, it can be argued that a table that contains a null is not properly a relational table at all. From a practical perspective, however, some nulls cannot be reasonably avoided. For example, not all students have a middle initial. As a general rule, nulls should be avoided as much as reasonably possible. In fact, an abundance of nulls is often a sign of a poor design. Also, nulls should be avoided in the database because their meaning is not always identifiable. For example, a null could represent any of the following:

- An unknown attribute value
- A known, but missing, attribute value
- A "not applicable" condition

Depending on the sophistication of the application development software, nulls can create problems when functions such as COUNT, AVERAGE, and SUM are used. In addition, nulls can create logical problems when relational tables are linked.

In addition to its role in providing a unique identity to each row in the table, the primary key may play an additional role in the controlled redundancy that allows

candidate key
A minimal superkey; that is, a key that does not contain a subset of attributes that is itself a superkey. See *key*.

entity integrity
The property of a relational table that guarantees each entity has a unique value in a primary key and that the key has no null values.

null
The absence of an attribute value. Note that a null is not a blank.

the relational model to work. Recall from Chapter 2, Data Models, that a hallmark of the relational model is that relationships between tables are implemented through common attributes as a form of controlled redundancy. For example, Figure 3.2 shows PRODUCT and VENDOR tables that are linked through a common attribute, VEND_CODE. VEND_CODE is referred to as a foreign key in the PRODUCT table. A **foreign key (FK)** is the primary key of one table that has been placed into another table to create a common attribute. In Figure 3.2, the primary key of VENDOR, VEND_CODE, was placed in the PRODUCT table; therefore, VEND_CODE is a foreign key in PRODUCT. One advantage of using a proper naming convention for table attributes is that you can identify foreign keys more easily. For example, because the STUDENT table in Figure 3.1 used a proper naming convention, you can identify two foreign keys in the table (DEPT_CODE and PROF_NUM) that imply the existence of two other tables in the database (DEPARTMENT and PROFESSOR) related to STUDENT.

FIGURE 3.2 AN EXAMPLE OF A SIMPLE RELATIONAL DATABASE

Table name: PRODUCT **Database name: Ch03_SaleCo**
Primary key: PROD_CODE
Foreign key: VEND_CODE

PROD_CODE	PROD_DESCRIPT	PROD_PRICE	PROD_ON_HAND	VEND_CODE
001278-AB	Claw hammer	12.95	23	232
123-21UUY	Houselite chain saw, 16-in. bar	189.99	4	235
QER-34256	Sledge hammer, 16-lb. head	18.63	6	231
SRE-657UG	Rat-tail file	2.99	15	232
ZZX/3245Q	Steel tape, 12-ft. length	6.79	8	235

link

Table name: VENDOR
Primary key: VEND_CODE
Foreign key: none

VEND_CODE	VEND_CONTACT	VEND_AREACODE	VEND_PHONE
230	Shelly K. Smithson	608	555-1234
231	James Johnson	615	123-4536
232	Annelise Crystall	608	224-2134
233	Candice Wallace	904	342-6567
234	Arthur Jones	615	123-3324
235	Henry Ortozo	615	899-3425

Just as the primary key has a role in ensuring the integrity of the database, so does the foreign key. Foreign keys are used to ensure **referential integrity**, the condition in which every reference to an entity instance by another entity instance is valid. In other words, every foreign key entry must either be null or a valid value in the primary key of the related table. Note that the PRODUCT table has referential integrity because every entry in VEND_CODE in the PRODUCT table is either null or a valid value in VEND_CODE in the VENDOR table. Every vendor referred to by a row in the PRODUCT table is a valid vendor.

Finally, a **secondary key** is defined as a key that is used strictly for data retrieval purposes. Suppose that customer data is stored in a CUSTOMER table in which the customer number is the primary key. Do you think that most customers will remember their numbers? Data retrieval for a customer is easier when the customer's last name and phone number are used. In that case, the primary key is the customer number; the secondary key is the combination of the customer's last name and phone number. Keep in mind that a secondary key does not necessarily yield a unique outcome. For example, a customer's last name and home telephone number could easily yield several matches in which one family lives together and shares a phone line. A less efficient secondary key would be the combination of the last name and zip code; this could yield dozens of matches, which could then be combed for a specific match.

foreign key (FK)
An attribute or attributes in one table whose values must match the primary key in another table or whose values must be null. See *key*.

referential integrity
A condition by which a dependent table's foreign key must have either a null entry or a matching entry in the related table.

secondary key
A key used strictly for data retrieval purposes. For example, customers are not likely to know their customer number (primary key), but the combination of last name, first name, middle initial, and telephone number will probably match the appropriate table row. See also *key*.

A secondary key's effectiveness in narrowing down a search depends on how restrictive the key is. For instance, although the secondary key CUS_CITY is legitimate from a database point of view, the attribute values *New York* or *Sydney* are not likely to produce a usable return unless you want to examine millions of possible matches. (Of course, CUS_CITY is a better secondary key than CUS_COUNTRY.)

Table 3.3 summarizes the various relational database table keys.

TABLE 3.3

RELATIONAL DATABASE KEYS

KEY TYPE	DEFINITION
Superkey	An attribute or combination of attributes that uniquely identifies each row in a table
Candidate key	A minimal (irreducible) superkey; a superkey that does not contain a subset of attributes that is itself a superkey
Primary key	A candidate key selected to uniquely identify all other attribute values in any given row; cannot contain null entries
Foreign key	An attribute or combination of attributes in one table whose values must either match the primary key in another table or be null
Secondary key	An attribute or combination of attributes used strictly for data retrieval purposes

3-3 Integrity Rules

Relational database integrity rules are very important to good database design. Relational database management systems (RDBMSs) enforce integrity rules automatically, but it is much safer to make sure your application design conforms to the entity and referential integrity rules mentioned in this chapter. Those rules are summarized in Table 3.4.

TABLE 3.4

INTEGRITY RULES

ENTITY INTEGRITY	DESCRIPTION
Requirement	All primary key entries are unique, and no part of a primary key may be null.
Purpose	Each row will have a unique identity, and foreign key values can properly reference primary key values.
Example	No invoice can have a duplicate number, nor can it be null; in short, all invoices are uniquely identified by their invoice number.
REFERENTIAL INTEGRITY	**DESCRIPTION**
Requirement	A foreign key may have either a null entry, as long as it is not a part of its table's primary key, or an entry that matches the primary key value in a table to which it is related (every non-null foreign key value *must* reference an *existing* primary key value).
Purpose	It is possible for an attribute *not* to have a corresponding value, but it will be impossible to have an invalid entry; the enforcement of the referential integrity rule makes it impossible to delete a row in one table whose primary key has mandatory matching foreign key values in another table.
Example	A customer might not yet have an assigned sales representative (number), but it will be impossible to have an invalid sales representative (number).

The integrity rules summarized in Table 3.4 are illustrated in Figure 3.3.

FIGURE 3.3 AN ILLUSTRATION OF INTEGRITY RULES

Table name: CUSTOMER **Database name: Ch03_InsureCo**
Primary key: CUS_CODE
Foreign key: AGENT_CODE

CUS_CODE	CUS_LNAME	CUS_FNAME	CUS_INITIAL	CUS_RENEW_DATE	AGENT_CODE
10010	Ramas	Alfred	A	05-Apr-2018	502
10011	Dunne	Leona	K	16-Jun-2018	501
10012	Smith	Kathy	W	29-Jan-2019	502
10013	Olowski	Paul	F	14-Oct-2018	
10014	Orlando	Myron		28-Dec-2018	501
10015	O'Brian	Amy	B	22-Sep-2018	503
10016	Brown	James	G	25-Mar-2019	502
10017	Williams	George		17-Jul-2018	503
10018	Farriss	Anne	G	03-Dec-2018	501
10019	Smith	Olette	K	14-Mar-2019	503

Table name: AGENT (only five selected fields are shown)
Primary key: AGENT_CODE
Foreign key: none

AGENT_CODE	AGENT_AREACODE	AGENT_PHONE	AGENT_LNAME	AGENT_YTD_SLS
501	713	228-1249	Alby	132736.75
502	615	882-1244	Hahn	138967.35
503	615	123-5589	Okon	127093.45

Note the following features of Figure 3.3.

- *Entity integrity*. The CUSTOMER table's primary key is CUS_CODE. The CUSTOMER primary key column has no null entries, and all entries are unique. Similarly, the AGENT table's primary key is AGENT_CODE, and this primary key column is also free of null entries.
- *Referential integrity*. The CUSTOMER table contains a foreign key, AGENT_CODE, that links entries in the CUSTOMER table to the AGENT table. The CUS_CODE row identified by the (primary key) number 10013 contains a null entry in its AGENT_CODE foreign key because Paul F. Olowski does not yet have a sales representative assigned to him. The remaining AGENT_CODE entries in the CUSTOMER table all match the AGENT_CODE entries in the AGENT table.

To avoid nulls, some designers use special codes, known as **flags**, to indicate the absence of some value. Using Figure 3.3 as an example, the code –99 could be used as the AGENT_CODE entry in the fourth row of the CUSTOMER table to indicate that customer Paul Olowski does not yet have an agent assigned to him. If such a flag is used, the AGENT table must contain a dummy row with an AGENT_CODE value of –99. Thus, the AGENT table's first record might contain the values shown in Table 3.5.

flags
Special codes implemented by designers to trigger a required response, alert end users to specified conditions, or encode values. Flags may be used to prevent nulls by bringing attention to the absence of a value in a table.

TABLE 3.5
A DUMMY VARIABLE VALUE USED AS A FLAG

AGENT_CODE	AGENT_AREACODE	AGENT_PHONE	AGENT_LNAME	AGENT_YTD_SLS
–99	000	000–0000	None	$0.00

Chapter 4, Entity Relationship (ER) Modeling, discusses several ways to handle nulls.

Other integrity rules that can be enforced in the relational model are the NOT NULL and UNIQUE constraints. The NOT NULL constraint can be placed on a column to ensure that every row in the table has a value for that column. The UNIQUE constraint is a restriction placed on a column to ensure that no duplicate values exist for that column.

3-4 Relational Algebra

The data in relational tables is of limited value unless the data can be manipulated to generate useful information. This section describes the basic data manipulation capabilities of the relational model. **Relational algebra** defines the theoretical way of manipulating table contents using relational operators. In Chapter 7, Introduction to Structured Query Language (SQL), and Chapter 8, Advanced SQL, you will learn how SQL commands can be used to accomplish relational algebra operations.

Note

The degree of relational completeness can be defined by the extent to which relational algebra is supported. To be considered minimally relational, the DBMS must support the key relational operators SELECT, PROJECT, and JOIN.

3-4a Formal Definitions and Terminology

Recall that the relational model is actually based on mathematical principles, and manipulating the data in the database can be described in mathematical terms. The good news is that, as database professionals, we do not have to write mathematical formulas to work with our data. Data is manipulated by database developers and programmers using powerful languages like SQL that hide the underlying math. However, understanding the underlying principles can give you a good feeling for the types of operations that can be performed, and it can help you to understand how to write your queries more efficiently and effectively.

One advantage of using formal mathematical representations of operations is that mathematical statements are unambiguous. These statements are very specific, and they require that database designers be specific in the language used to explain them. As previously explained, it is common to use the terms *relation* and *table* interchangeably. However, since the mathematical terms need to be precise, we will use the more specific term relation when discussing the formal definitions of the various relational algebra operators.

Before considering the specific relational algebra operators, it is necessary to formalize our understanding of a table.

relational algebra
A set of mathematical principles that form the basis for manipulating relational table contents; the eight main functions are SELECT, PROJECT, JOIN, INTERSECT, UNION, DIFFERENCE, PRODUCT, and DIVIDE.

relvar
Short for relation variable, a variable that holds a relation. A relvar is a container (variable) for holding relation data, not the relation itself.

One important aspect of using the specific term *relation* is that it acknowledges the distinction between the relation and the relation variable, or *relvar*, for short. A relation is the data that we see in our tables. A **relvar** is a variable that holds a relation. For example, imagine you were writing a program and created a variable named *qty* for holding integer data. The variable *qty* is not an integer itself; it is a container for holding integers. Similarly, when you create a table, the table structure holds the table data. The structure is properly called a relvar, and the data in the structure would be a relation. The relvar is a container (variable) for holding relation data, not the relation itself. The data in the table is a relation.

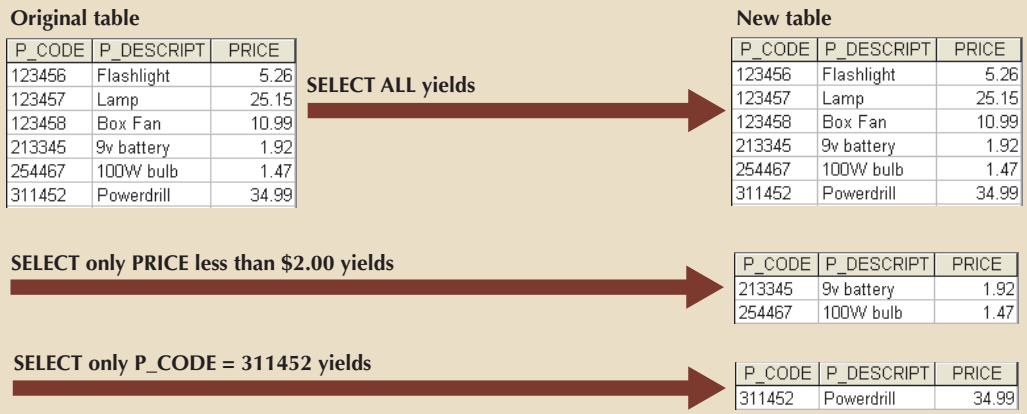
A relvar has two parts: the heading and the body. The relvar heading contains the names of the attributes, while the relvar body contains the relation. To conveniently maintain this distinction in formulas, an unspecified relation is often assigned a lowercase letter (e.g., "r"), while the relvar is assigned an uppercase letter (e.g., "R"). We could then say that *r* is a relation of type *R*, or *r(R)*.

3-4b Relational Set Operators

The relational operators have the property of **closure**; that is, the use of relational algebra operators on existing relations (tables) produces new relations. Numerous operators have been defined. Some operators are fundamental, while others are convenient but can be derived using the fundamental operators. In this section, the focus will be on the SELECT (or RESTRICT), PROJECT, UNION, INTERSECT, DIFFERENCE, PRODUCT, JOIN, and DIVIDE operators.

Select (Restrict) **SELECT**, also known as **RESTRICT**, is referred to as a unary operator because it only uses one table as input. It yields values for all rows found in the table that satisfy a given condition. SELECT can be used to list all of the rows, or it can yield only rows that match a specified criterion. In other words, SELECT yields a horizontal subset of a table. SELECT will not limit the attributes returned so all attributes of the table will be included in the result. The effect of a SELECT operation is shown in Figure 3.4.

FIGURE 3.4 SELECT



Note

Formally, SELECT is denoted by the lowercase Greek letter sigma (σ). Sigma is followed by the condition to be evaluated (called a predicate) as a subscript, and then the relation is listed in parentheses. For example, to SELECT all of the rows in the CUSTOMER table that have the value "10010" in the CUS_CODE attribute, you would write the following:

$$\sigma_{cus\_code = 10010} (customer)$$

Project **PROJECT** yields all values for selected attributes. It is also a unary operator, accepting only one table as input. PROJECT will return only the attributes requested, in the order in which they are requested. In other words, PROJECT yields a vertical subset of a table. PROJECT will not limit the rows returned, so all rows of the specified attributes will be included in the result. The effect of a PROJECT operation is shown in Figure 3.5.

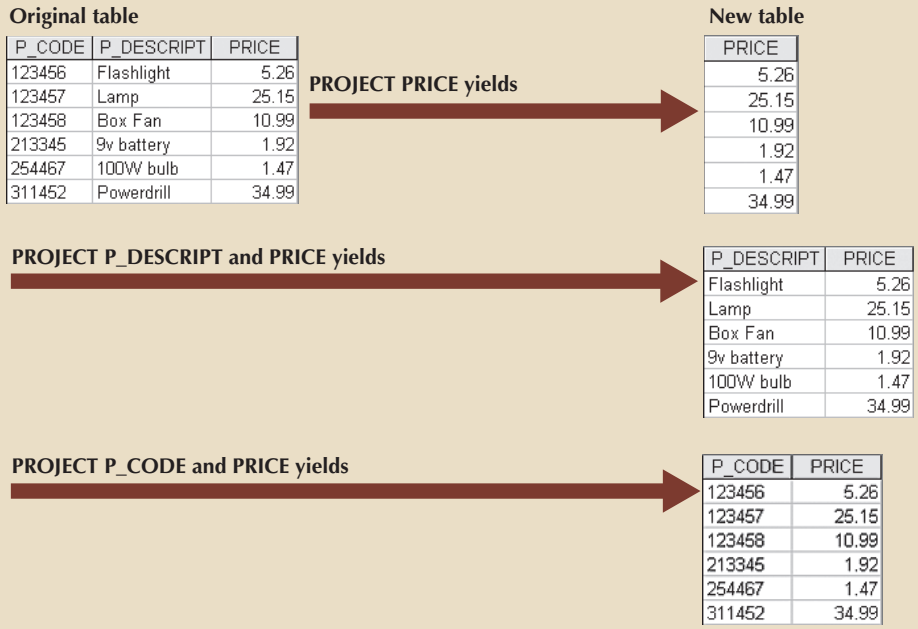
closure
A property of relational operators that permits the use of relational algebra operators on existing tables (relations) to produce new relations.

SELECT
In relational algebra, an operator used to select a subset of rows. Also known as *RESTRICT*.

RESTRICT
See *SELECT*.

PROJECT
In relational algebra, an operator used to select a subset of columns.

FIGURE 3.5 PROJECT



Note

Formally, PROJECT is denoted by the Greek letter pi (π). Some sources use the uppercase letter, and other sources use the lowercase letter. Codd used the lowercase π in his original article on the relational model, and that is what we use here. Pi is followed by the list of attributes to be returned as subscripts and then the relation listed in parentheses. For example, to PROJECT the CUS_FNAME and CUS_LNAME attributes in the CUSTOMER table, you would write the following:

$$\pi_{cus\_fname,\ cus\_lname}\ (customer)$$

Since relational operators have the property of closure, that is, they accept relations as input and produce relations as output, it is possible to combine operators. For example, you can combine the two previous operators to find the first and last name of the customer with customer code 10010:

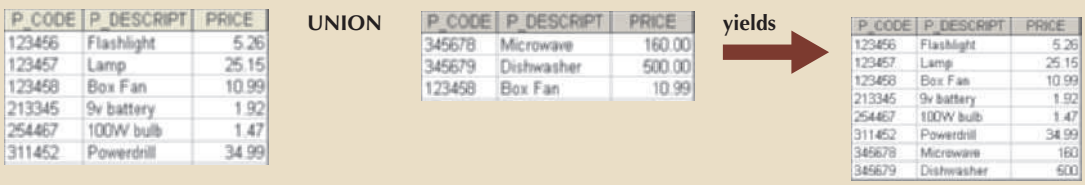
$$\pi_{cus\_fname,\ cus\_lname}\ (\sigma_{cus\_code\ =\ 10010}\ (customer))$$

UNION
In relational algebra, an operator used to merge (append) two tables into a new table, dropping the duplicate rows. The tables must be *union-compatible*.

union-compatible
Two or more tables that have the same number of columns and the corresponding columns have compatible domains.

Union **UNION** combines all rows from two tables, excluding duplicate rows. To be used in the UNION, the tables must have the same attribute characteristics; in other words, the columns and domains must be compatible. When two or more tables share the same number of columns, and when their corresponding columns share the same or compatible domains, they are said to be **union-compatible**. The effect of a UNION operation is shown in Figure 3.6.

FIGURE 3.6 UNION



Note

UNION is denoted by the symbol ∪. If the relations SUPPLIER and VENDOR are union-compatible, then a UNION between them would be denoted as follows:

supplier ∪ vendor

It is rather unusual to find two relations that are union-compatible in a database. Typically, PROJECT operators are applied to relations to produce results that are union-compatible. For example, assume the SUPPLIER and VENDOR tables are not union-compatible. If you wish to produce a listing of all vendor and supplier names, then you can PROJECT the names from each table and then perform a UNION with them.

$\pi_{supplier\_name}$ (supplier) ∪ π_{vendor_name} (vendor)

Intersect **INTERSECT** yields only the rows that appear in both tables. As with UNION, the tables must be union-compatible to yield valid results. For example, you cannot use INTERSECT if one of the attributes is numeric and one is character-based. For the rows to be considered the same in both tables and appear in the result of the INTERSECT, the entire rows must be exact duplicates. The effect of an INTERSECT operation is shown in Figure 3.7.

FIGURE 3.7 INTERSECT

STU_FNAME	STU_LNAME
George	Jones
Jane	Smith
Peter	Robinson
Franklin	Johnson
Martin	Lopez

INTERSECT

EMP_FNAME	EMP_LNAME
Franklin	Lopez
William	Turner
Franklin	Johnson
Susan	Rogers

yields

STU_FNAME	STU_LNAME
Franklin	Johnson

Note

INTERSECT is denoted by the symbol ∩. If the relations SUPPLIER and VENDOR are union-compatible, then an INTERSECT between them would be denoted as follows:

supplier ∩ vendor

Just as with the UNION operator, it is unusual to find two relations that are union-compatible in a database, so PROJECT operators are applied to relations to produce results that can be manipulated with an INTERSECT operator. For example, again assume the SUPPLIER and VENDOR tables are not union-compatible. If you wish to produce a listing of any vendor and supplier names that are the same in both tables, then you can PROJECT the names from each table and then perform an INTERSECT with them.

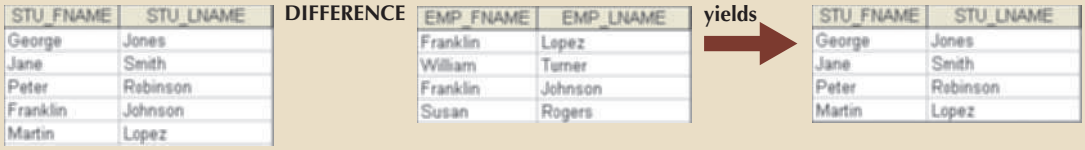
$\pi_{supplier\_name}$ (supplier) ∩ π_{vendor_name} (vendor)

Difference **DIFFERENCE** yields all rows in one table that are not found in the other table; that is, it subtracts one table from the other. As with UNION, the tables must be union-compatible to yield valid results. The effect of a DIFFERENCE operation is shown in Figure 3.8. However, note that subtracting the first table from the second table is not the same as subtracting the second table from the first table.

INTERSECT
In relational algebra, an operator used to yield only the rows that are common to two union-compatible tables.

DIFFERENCE
In relational algebra, an operator used to yield all rows from one table that are not found in another union-compatible table.

FIGURE 3.8 DIFFERENCE



STU_FNAME	STU_LNAME
George	Jones
Jane	Smith
Peter	Robinson
Franklin	Johnson
Martin	Lopez

DIFFERENCE

EMP_FNAME	EMP_LNAME
Franklin	Lopez
William	Turner
Franklin	Johnson
Susan	Rogers

yields

STU_FNAME	STU_LNAME
George	Jones
Jane	Smith
Peter	Robinson
Martin	Lopez

Note

DIFFERENCE is denoted by the minus symbol –. If the relations SUPPLIER and VENDOR are union-compatible, then a DIFFERENCE of SUPPLIER minus VENDOR would be written as follows:

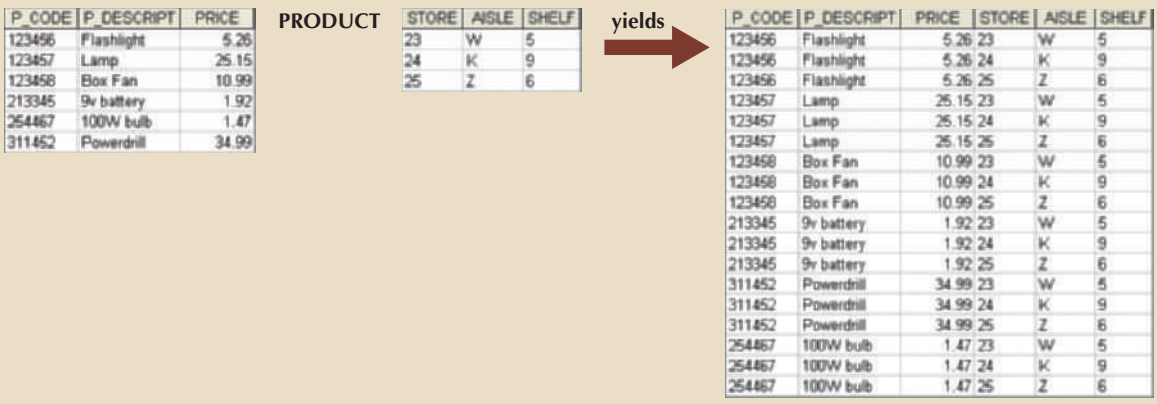
supplier – vendor

Assuming the SUPPLIER and VENDOR tables are not union-compatible, producing a list of any supplier names that do not appear as vendor names, then you can use a DIFFERENCE operator.

$\pi_{supplier\_name}(supplier) - \pi_{vendor\_name}(vendor)$

Product **PRODUCT** yields all possible pairs of rows from two tables—also known as the Cartesian product. Therefore, if one table has 6 rows and the other table has 3 rows, the PRODUCT yields a list composed of 6 × 3 = 18 rows. The effect of a PRODUCT operation is shown in Figure 3.9.

FIGURE 3.9 PRODUCT



P_CODE	P_DESCRIPT	PRICE
123456	Flashlight	5.26
123457	Lamp	25.15
123458	Box Fan	10.99
213345	9v battery	1.92
254467	100W bulb	1.47
311452	Powerdrill	34.99

PRODUCT

STORE	AISLE	SHELF
23	W	5
24	K	9
25	Z	6

yields

P_CODE	P_DESCRIPT	PRICE	STORE	AISLE	SHELF
123456	Flashlight	5.26	23	W	5
123456	Flashlight	5.26	24	K	9
123456	Flashlight	5.26	25	Z	6
123457	Lamp	25.15	23	W	5
123457	Lamp	25.15	24	K	9
123457	Lamp	25.15	25	Z	6
123458	Box Fan	10.99	23	W	5
123458	Box Fan	10.99	24	K	9
123458	Box Fan	10.99	25	Z	6
213345	9v battery	1.92	23	W	5
213345	9v battery	1.92	24	K	9
213345	9v battery	1.92	25	Z	6
311452	Powerdrill	34.99	23	W	5
311452	Powerdrill	34.99	24	K	9
311452	Powerdrill	34.99	25	Z	6
254467	100W bulb	1.47	23	W	5
254467	100W bulb	1.47	24	K	9
254467	100W bulb	1.47	25	Z	6

Note

PRODUCT is denoted by the multiplication symbol ×. The PRODUCT of the CUSTOMER and AGENT relations would be written as follows:

customer × agent

A Cartesian product produces a set of sequences in which every member of one set is paired with every member of another set. In terms of relations, this means that every tuple in one relation is paired with every tuple in the second relation.

PRODUCT
In relational algebra, an operator used to yield all possible pairs of rows from two tables. Also known as the *Cartesian product*.

Join **JOIN** allows information to be intelligently combined from two or more tables. JOIN is the real power behind the relational database, allowing the use of independent tables linked by common attributes. The CUSTOMER and AGENT tables shown in Figure 3.10 will be used to illustrate several types of joins.

FIGURE 3.10 TWO TABLES THAT WILL BE USED IN JOIN ILLUSTRATIONS

Table name: CUSTOMER

CUS_CODE	CUS_LNAME	CUS_ZIP	AGENT_CODE
1132445	Walker	32145	231
1217782	Adares	32145	125
1312243	Rakowski	34129	167
1321242	Rodriguez	37134	125
1542311	Smithson	37134	421
1657399	Vanloo	32145	231

Table name: AGENT

AGENT_CODE	AGENT_PHONE
125	6152439887
167	6153426778
231	6152431124
333	9041234445

A **natural join** links tables by selecting only the rows with common values in their common attribute(s). A natural join is the result of a three-stage process:

1. First, a PRODUCT of the tables is created, yielding the results shown in Figure 3.11.

FIGURE 3.11 NATURAL JOIN, STEP 1: PRODUCT

CUS_CODE	CUS_LNAME	CUS_ZIP	CUSTOMER.AGENT_CODE	AGENT.AGENT_CODE	AGENT_PHONE
1132445	Walker	32145	231	125	6152439887
1132445	Walker	32145	231	167	6153426778
1132445	Walker	32145	231	231	6152431124
1132445	Walker	32145	231	333	9041234445
1217782	Adares	32145	125	125	6152439887
1217782	Adares	32145	125	167	6153426778
1217782	Adares	32145	125	231	6152431124
1217782	Adares	32145	125	333	9041234445
1312243	Rakowski	34129	167	125	6152439887
1312243	Rakowski	34129	167	167	6153426778
1312243	Rakowski	34129	167	231	6152431124
1312243	Rakowski	34129	167	333	9041234445
1321242	Rodriguez	37134	125	125	6152439887
1321242	Rodriguez	37134	125	167	6153426778
1321242	Rodriguez	37134	125	231	6152431124
1321242	Rodriguez	37134	125	333	9041234445
1542311	Smithson	37134	421	125	6152439887
1542311	Smithson	37134	421	167	6153426778
1542311	Smithson	37134	421	231	6152431124
1542311	Smithson	37134	421	333	9041234445
1657399	Vanloo	32145	231	125	6152439887
1657399	Vanloo	32145	231	167	6153426778
1657399	Vanloo	32145	231	231	6152431124
1657399	Vanloo	32145	231	333	9041234445

2. Second, a SELECT is performed on the output of Step 1 to yield only the rows for which the AGENT_CODE values are equal. The common columns are referred to as the **join columns**. Step 2 yields the results shown in Figure 3.12.

JOIN
In relational algebra, a type of operator used to yield rows from two tables based on criteria. There are many types of joins, such as natural join, theta join, equijoin, and outer join.

natural join
A relational operation that yields a new table composed of only the rows with common values in their common attribute(s).

join columns
Columns that are used in the criteria of join operations. The join columns generally share similar values.

FIGURE 3.12 NATURAL JOIN, STEP 2: SELECT

CUS_CODE	CUS_LNAME	CUS_ZIP	CUSTOMER.AGENT_CODE	AGENT.AGENT_CODE	AGENT_PHONE
1217782	Adares	32145	125	125	6152439887
1321242	Rodriguez	37134	125	125	6152439887
1312243	Rakowski	34129	167	167	6153426778
1132445	Walker	32145	231	231	6152431124
1657399	Vanloo	32145	231	231	6152431124

3. A PROJECT is performed on the results of Step 2 to yield a single copy of each attribute, thereby eliminating duplicate columns. Step 3 yields the output shown in Figure 3.13.

FIGURE 3.13 NATURAL JOIN, STEP 3: PROJECT

CUS_CODE	CUS_LNAME	CUS_ZIP	AGENT_CODE	AGENT_PHONE
1217782	Adares	32145	125	6152439887
1321242	Rodriguez	37134	125	6152439887
1312243	Rakowski	34129	167	6153426778
1132445	Walker	32145	231	6152431124
1657399	Vanloo	32145	231	6152431124

The final outcome of a natural join yields a table that does not include unmatched pairs and provides only the copies of the matches.

Note a few crucial features of the natural join operation:

- If no match is made between the table rows, the new table does not include the unmatched row. In that case, neither AGENT_CODE 421 nor the customer whose last name is Smithson is included. Smithson's AGENT_CODE 421 does not match any entry in the AGENT table.
- The column on which the join was made—that is, AGENT_CODE—occurs only once in the new table.
- If the same AGENT_CODE were to occur several times in the AGENT table, a customer would be listed for each match. For example, if the AGENT_CODE 167 occurred three times in the AGENT table, the customer named Rakowski would also occur three times in the resulting table because Rakowski is associated with AGENT_CODE 167. (Of course, a good AGENT table cannot yield such a result because it would contain unique primary key values.)

Note

Natural join is normally just referred to as JOIN in formal treatments. JOIN is denoted by the symbol ⋈. The JOIN of the CUSTOMER and AGENT relations would be written as follows:

customer ⋈ agent

Notice that the JOIN of two relations returns all of the attributes of both relations, except only one copy of the common attribute is returned. Formally, this is described as a UNION of the relvar headings. Therefore, the JOIN of the relations (c ⋈ a) includes the UNION of the relvars (C ∪ A). Also note that, as described above, JOIN is not a fundamental relational algebra operator. It can be derived from other operators as follows:

$$\pi_{\text{cus\_code, cus\_lname, cus\_fname, cus\_initial, cus\_renew\_date, agent\_code, agent\_areacode, agent\_phone, agent\_lname, agent\_ytd\_sls}} (\sigma_{\text{customer.agent\_code = agent.agent\_code}} (\text{customer} \times \text{agent}))$$

Another form of join, known as an **equijoin**, links tables on the basis of an equality condition that compares specified columns of each table. The outcome of the equijoin does not eliminate duplicate columns, and the condition or criterion used to join the tables must be explicitly defined. In fact, the result of an equijoin looks just like the outcome shown in Figure 3.12 for Step 2 of a natural join. The equijoin takes its name from the equality comparison operator (=) used in the condition. If any other comparison operator is used, the join is called a **theta join**.

Note

In formal terms, theta join is considered an extension of natural join. Theta join is denoted by adding a theta subscript after the JOIN symbol: $\bowtie_{\theta}$. Equijoin is then a special type of theta join.

Each of the preceding joins is often classified as an inner join. An **inner join** only returns matched records from the tables that are being joined. In an **outer join**, the matched pairs would be retained, and any unmatched values in the other table would be left null. It is an easy mistake to think that an outer join is the opposite of an inner join. However, it is more accurate to think of an outer join as an "inner join plus." The outer join still returns all of the matched records that the inner join returns, plus it returns the unmatched records from one of the tables. More specifically, if an outer join is produced for tables CUSTOMER and AGENT, two scenarios are possible:

- A **left outer join** yields all of the rows in the CUSTOMER table, including those that do not have a matching value in the AGENT table. An example of such a join is shown in Figure 3.14.

FIGURE 3.14 LEFT OUTER JOIN

CUS_CODE	CUS_LNAME	CUS_ZIP	CUSTOMER.AGENT_CODE	AGENT.AGENT_CODE	AGENT_PHONE
1217782	Adares	32145	125	125	6152439887
1321242	Rodriguez	37134	125	125	6152439887
1312243	Rakowski	34129	167	167	6153426778
1132445	Walker	32145	231	231	6152431124
1657399	Vanloo	32145	231	231	6152431124
1542311	Smithson	37134	421		

- A **right outer join** yields all of the rows in the AGENT table, including those that do not have matching values in the CUSTOMER table. An example of such a join is shown in Figure 3.15.

FIGURE 3.15 RIGHT OUTER JOIN

CUS_CODE	CUS_LNAME	CUS_ZIP	CUSTOMER.AGENT_CODE	AGENT.AGENT_CODE	AGENT_PHONE
1217782	Adares	32145	125	125	6152439887
1321242	Rodriguez	37134	125	125	6152439887
1312243	Rakowski	34129	167	167	6153426778
1132445	Walker	32145	231	231	6152431124
1657399	Vanloo	32145	231	231	6152431124
				333	9041234445

Outer joins are especially useful when you are trying to determine what values in related tables cause referential integrity problems. Such problems are created when foreign key values do not match the primary key values in the related table(s). In fact, if you are asked to convert large spreadsheets or other "nondatabase" data into relational database tables,

equijoin
A join operator that links tables based on an equality condition that compares specified columns of the tables.

theta join
A join operator that links tables using an inequality comparison operator (<, >, <=, >=) in the join condition.

inner join
A join operation in which only rows that meet a given criterion are selected. The criterion can be an equality condition (natural join or equijoin) or an inequality condition (theta join). The most commonly used type of join.

outer join
A join operation that produces a table in which all unmatched pairs are retained; unmatched values in the related table are left null.

left outer join
A join operation that yields all the rows in the left table, including those that have no matching values in the other table.

right outer join
A join operation that yields all of the rows in the right table, including the ones with no matching values in the other table.

you will discover that the outer joins save you vast amounts of time and uncounted headaches when you encounter referential integrity errors after the conversions.

You may wonder why the outer joins are labeled "left" and "right." The labels refer to the order in which the tables are listed in the SQL command. Chapter 7 explores such joins in more detail.

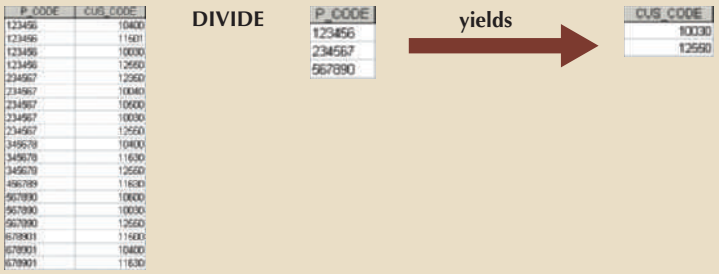
Note

Outer join is also an extension of JOIN. Outer joins are the application of JOIN, DIFFERENCE, UNION, and PRODUCT. A JOIN returns the matched tuples, DIFFERENCE finds the tuples in one table that have values in the common attribute that do not appear in the common attribute of the other relation, these unmatched tuples are combined with NULL values through a PRODUCT, and then a UNION combines these results into a single relation. Clearly, a defined outer join is a great simplification! Left and right outer joins are denoted by the symbols ⋈ and ⋈, respectively.

Divide The **DIVIDE** operator is used to answer questions about one set of data being associated with all values of data in another set of data. The DIVIDE operation uses one double-column table (Table 1) as the dividend and one single-column table (Table 2) as the divisor. For example, Figure 3.16 shows a list of customers and the products purchased in Table 1 on the left. Table 2 in the center contains a set of products that are of interest to the users. A DIVIDE operation can be used to determine which customers, if any, purchased every product shown in Table 2. In the figure, the dividend contains the P_CODE and CUS_CODE columns. The divisor contains the P_CODE column. The tables must have a common column—in this case, the P_CODE column. The output of the DIVIDE operation on the right is a single column that contains all values from the second column of the dividend (CUS_CODE) that are associated with every row in the divisor.

Using the example shown in Figure 3.16, note the following:

FIGURE 3.16 DIVIDE



DIVIDE
In relational algebra, an operator that answers queries about one set of data being associated with all values of data in another set of data.

- Table 1 is "divided" by Table 2 to produce Table 3. Tables 1 and 2 both contain the P_CODE column but do not share the CUS_CODE column.
- To be included in the resulting Table 3, a value in the unshared column (CUS_CODE) must be associated with every value in Table 2.
- The only customers associated with all of products 123456, 234567, and 567890 are customers 10030 and 12550.

Note

The DIVIDE operator is denoted by the division symbol ÷. Given two relations, R and S, the DIVISION of them would be written as r ÷ s.

3-5 The Data Dictionary and the System Catalog

The **data dictionary** provides a detailed description of all tables in the database created by the user and designer. Thus, the data dictionary contains at least all of the attribute names and characteristics for each table in the system. In short, the data dictionary contains metadata—data about data. Using the small database presented in Figure 3.3, you might picture its data dictionary as shown in Table 3.6.

Note

The data dictionary in Table 3.6 is an example of the *human* view of the entities, attributes, and relationships. The purpose of this data dictionary is to ensure that all members of database design and implementation teams use the same table and attribute names and characteristics. The DBMS's internally stored data dictionary contains additional information about relationship types, entity and referential integrity checks and enforcement, and index types and components. This additional information is generated during the database implementation stage.

The data dictionary is sometimes described as "the database designer's database" because it records the design decisions about tables and their structures.

Like the data dictionary, the system catalog contains metadata. The **system catalog** can be described as a detailed system data dictionary that describes all objects within the database, including data about table names, table's creator and creation date, number of columns in each table, data type corresponding to each column, index filenames, index creators, authorized users, and access privileges. Because the system catalog contains all required data dictionary information, the terms *system catalog* and *data dictionary* are often used interchangeably. In fact, current relational database software generally provides only a system catalog, from which the designer's data dictionary information may be derived. The system catalog is actually a system-created database whose tables store the user/designer-created database characteristics and contents. Therefore, the system catalog tables can be queried just like any user/designer-created table.

In effect, the system catalog automatically produces database documentation. As new tables are added to the database, that documentation also allows the RDBMS to check for and eliminate homonyms and synonyms. In general terms, **homonyms** are similar-sounding words with different meanings, such as *boar* and *bore*, or a word with different meanings, such as *fair* (which means "just" in some contexts and "festival" in others). In a database context, the word *homonym* indicates the use of the same name to label different attributes. For example, you might use C_NAME to label a customer name attribute in a CUSTOMER table and use C_NAME to label a consultant name attribute in a CONSULTANT table. To lessen confusion, you should avoid database homonyms; the data dictionary is very useful in this regard.

data dictionary
A DBMS component that stores metadata—data about data. Thus, the data dictionary contains the data definition as well as their characteristics and relationships. A data dictionary may also include data that are external to the DBMS. Also known as an *information resource dictionary*. See also *active data dictionary*, *metadata*, and *passive data dictionary*.

system catalog
A detailed system data dictionary that describes all objects in a database.

homonym
The use of the same name to label different attributes. Homonyms generally should be avoided. See also *synonym*.

TABLE 3.6

A SAMPLE DATA DICTIONARY

TABLE NAME	ATTRIBUTE NAME	CONTENTS	TYPE	FORMAT	RANGE	REQUIRED	PK OR FK	FK REFERENCED TABLE
CUSTOMER	CUS_CODE	Customer account code	CHAR(5)	99999	10000–99999	Y	PK	
	CUS_LNAME	Customer last name	VARCHAR(20)	Xxxxxxxx		Y		
	CUS_FNAME	Customer first name	VARCHAR(20)	Xxxxxxxx		Y		
	CUS_INITIAL	Customer initial	CHAR(1)	X				
	CUS_RENEW_DATE	Customer insurance renewal date	DATE	dd-mmm-yyyy				
	AGENT_CODE	Agent code	CHAR(3)	999			FK	AGENT
AGENT	AGENT_CODE	Agent code	CHAR(3)	999		Y	PK	
	AGENT_AREACODE	Agent area code	CHAR(3)	999		Y		
	AGENT_PHONE	Agent telephone number	CHAR(8)	999–9999		Y		
	AGENT_LNAME	Agent last name	VARCHAR(20)	Xxxxxxxx		Y		
	AGENT_YTD_SLS	Agent year-to-date sales	NUMBER(9,2)	9,999,999.99				

FK	= Foreign key
PK	= Primary key
CHAR	= Fixed character length data (1 – 255 characters)
VARCHAR	= Variable character length data (1 – 2,000 characters)
NUMBER	= Numeric data. NUMBER (9,2) is used to specify numbers with up to nine digits, including two digits to the right of the decimal place. Some RDBMS permit the use of a MONEY or CURRENCY data type.

Note

Telephone area codes are always composed of digits 0–9, but because area codes are not used arithmetically, they are most efficiently stored as character data. Also, the area codes are always composed of three digits. Therefore, the area code data type is defined as CHAR(3). On the other hand, names do not conform to a standard length. Therefore, the customer first names are defined as VARCHAR(20), indicating that up to 20 characters may be used to store the names. Character data are shown as left-aligned.

In a database context, a **synonym** is the opposite of a homonym and indicates the use of different names to describe the same attribute. For example, *car* and *auto* refer to the same object. Synonyms must be avoided whenever possible.

3-6 Relationships within the Relational Database

You already know that relationships are classified as one-to-one (1:1), one-to-many (1:M), and many-to-many (M:N or M:M). This section explores those relationships further to help you apply them properly when you start developing database designs. This section focuses on the following points:

- The 1:M relationship is the relational modeling ideal. Therefore, this relationship type should be the norm in any relational database design.
- The 1:1 relationship should be rare in any relational database design.
- M:N relationships cannot be implemented as such in the relational model. Later in this section, you will see how any M:N relationship can be changed into two 1:M relationships.

3-6a The 1:M Relationship

The 1:M relationship is the norm for relational databases. To see how such a relationship is modeled and implemented, consider the PAINTER and PAINTING example shown in Figure 3.17.

FIGURE 3.17 THE 1:M RELATIONSHIP BETWEEN PAINTER AND PAINTING

Compare the data model in Figure 3.17 with its implementation in Figure 3.18. As you examine the PAINTER and PAINTING table contents in Figure 3.18, note the following features:

- Each painting was created by one and only one painter, but each painter could have created many paintings. Note that painter 123 (Georgette P. Ross) has three works stored in the PAINTING table.
- There is only one row in the PAINTER table for any given row in the PAINTING table, but there may be many rows in the PAINTING table for any given row in the PAINTER table.

synonym
The use of different names to identify the same object, such as an entity, an attribute, or a relationship; synonyms should generally be avoided. See also *homonym*.

Note

The one-to-many (1:M) relationship is easily implemented in the relational model by putting the *primary key of the "1" side in the table of the "many" side as a foreign key*.

FIGURE 3.18 THE IMPLEMENTED 1:M RELATIONSHIP BETWEEN PAINTER AND PAINTING

Table name: PAINTER
Primary key: PAINTER_NUM
Foreign key: none

Database name: Ch03_Museum

PAINTER_NUM	PAINTER_LNAME	PAINTER_FNAME	PAINTER_INITIAL
123	Ross	Georgette	P
126	Itero	Julio	G

Table name: PAINTING
Primary key: PAINTING_NUM
Foreign key: PAINTER_NUM

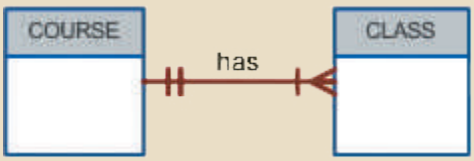
PAINTING_NUM	PAINTING_TITLE	PAINTER_NUM
1338	Dawn Thunder	123
1339	Vanilla Roses To Nowhere	123
1340	Tired Flounders	126
1341	Hasty Exit	123
1342	Plastic Paradise	126

The 1:M relationship is found in any database environment. Students in a typical college or university will discover that each COURSE can generate many CLASSes but that each CLASS refers to only one COURSE. For example, an Accounting II course might yield two classes: one offered on Monday, Wednesday, and Friday (MWF) from 10:00 a.m. to 10:50 a.m. and one offered on Thursday (Th) from 6:00 p.m. to 8:40 p.m. Therefore, the 1:M relationship between COURSE and CLASS might be described this way:

- Each COURSE can have many CLASSes, but each CLASS references only one COURSE.
- There will be only one row in the COURSE table for any given row in the CLASS table, but there can be many rows in the CLASS table for any given row in the COURSE table.

Figure 3.19 maps the entity relationship model (ERM) for the 1:M relationship between COURSE and CLASS.

FIGURE 3.19 THE 1:M RELATIONSHIP BETWEEN COURSE AND CLASS



The 1:M relationship between COURSE and CLASS is further illustrated in Figure 3.20.

Using Figure 3.20, take a minute to review some important terminology. Note that CLASS_CODE in the CLASS table uniquely identifies each row. Therefore, CLASS_CODE has been chosen to be the primary key. However, the combination CRS_CODE and CLASS_SECTION will also uniquely identify each row in the class table. In other words, the *composite key* composed of CRS_CODE and CLASS_SECTION is a candidate key. Any *candidate key* must have the not-null and unique constraints enforced. (You will see how this is done when you learn SQL in Chapter 8.)

FIGURE 3.20 THE IMPLEMENTED 1:M RELATIONSHIP BETWEEN COURSE AND CLASS

Table name: COURSE
Primary key: CRS_CODE
Foreign key: none

Database name: Ch03_TinyCollege

CRS_CODE	DEPT_CODE	CRS_DESCRIPTION	CRS_CREDIT
ACCT-211	ACCT	Accounting I	3
ACCT-212	ACCT	Accounting II	3
CIS-220	CIS	Intro. to Microcomputing	3
CIS-420	CIS	Database Design and Implementation	4
QM-261	CIS	Intro. to Statistics	3
QM-362	CIS	Statistical Applications	4

Table name: CLASS
Primary key: CLASS_CODE
Foreign key: CRS_CODE

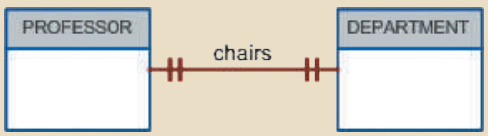
CLASS_CODE	CRS_CODE	CLASS_SECTION	CLASS_TIME	CLASS_ROOM	PROF_NUM
10012	ACCT-211	1	MWF 8:00-8:50 a.m.	BUS311	105
10013	ACCT-211	2	MWF 9:00-9:50 a.m.	BUS200	105
10014	ACCT-211	3	TTh 2:30-3:45 p.m.	BUS252	342
10015	ACCT-212	1	MWF 10:00-10:50 a.m.	BUS311	301
10016	ACCT-212	2	Th 6:00-8:40 p.m.	BUS252	301
10017	CIS-220	1	MWF 9:00-9:50 a.m.	KLR209	228
10018	CIS-220	2	MWF 9:00-9:50 a.m.	KLR211	114
10019	CIS-220	3	MWF 10:00-10:50 a.m.	KLR209	228
10020	CIS-420	1	W 6:00-8:40 p.m.	KLR209	162
10021	QM-261	1	MWF 8:00-8:50 a.m.	KLR200	114
10022	QM-261	2	TTh 1:00-2:15 p.m.	KLR200	114
10023	QM-362	1	MWF 11:00-11:50 a.m.	KLR200	162
10024	QM-362	2	TTh 2:30-3:45 p.m.	KLR200	162

For example, note in Figure 3.18 that the PAINTER table's primary key, PAINTER_NUM, is included in the PAINTING table as a foreign key. Similarly, in Figure 3.20, the COURSE table's primary key, CRS_CODE, is included in the CLASS table as a foreign key.

3-6b The 1:1 Relationship

As the 1:1 label implies, one entity in a 1:1 relationship can be related to only one other entity, and vice versa. For example, one department chair—a professor—can chair only one department, and one department can have only one department chair. The entities PROFESSOR and DEPARTMENT thus exhibit a 1:1 relationship. (You might argue that not all professors chair a department and professors cannot be *required* to chair a department. That is, the relationship between the two entities is optional. However, at this stage of the discussion, you should focus your attention on the basic 1:1 relationship. (Optional relationships will be addressed in Chapter 4.) The basic 1:1 relationship is modeled in Figure 3.21, and its implementation is shown in Figure 3.22.

FIGURE 3.21 THE 1:1 RELATIONSHIP BETWEEN PROFESSOR AND DEPARTMENT



As you examine the tables in Figure 3.22, note several important features:

- Each professor is a Tiny College employee. Therefore, the professor identification is through the EMP_NUM. (However, note that not all employees are professors—there's another optional relationship.)
- The 1:1 "PROFESSOR chairs DEPARTMENT" relationship is implemented by having the EMP_NUM foreign key in the DEPARTMENT table. Note that the 1:1 relationship is treated as a special case of the 1:M relationship in which the "many" side is restricted to a single occurrence. In this case, DEPARTMENT contains the EMP_NUM as a foreign key to indicate that it is the *department* that has a chair.

FIGURE 3.22 THE IMPLEMENTED 1:1 RELATIONSHIP BETWEEN PROFESSOR AND DEPARTMENT

Table name: PROFESSOR
Primary key: EMP_NUM
Foreign key: DEPT_CODE

Database name: Ch03_TinyCollege

EMP_NUM	DEPT_CODE	PROF_OFFICE	PROF_EXTENSION	PROF_HIGH_DEGREE
103	HIST	DRE 156	6783	Ph.D.
104	ENG	DRE 102	5561	MA
105	ACCT	KLR 229D	8665	Ph.D.
106	MKT/MGT	KLR 126	3899	Ph.D.
110	BIOL	AAK 160	3412	Ph.D.
114	ACCT	KLR 211	4436	Ph.D.
155	MATH	AAK 201	4440	Ph.D.
160	ENG	DRE 102	2248	Ph.D.
162	CIS	KLR 203E	2359	Ph.D.
191	MKT/MGT	KLR 409B	4016	DBA
195	PSYCH	AAK 297	3550	Ph.D.
209	CIS	KLR 333	3421	Ph.D.
228	CIS	KLR 300	3000	Ph.D.
297	MATH	AAK 194	1145	Ph.D.
299	ECON/FIN	KLR 284	2851	Ph.D.
301	ACCT	KLR 244	4683	Ph.D.
335	ENG	DRE 208	2000	Ph.D.
342	SOC	BBG 208	5514	Ph.D.
387	BIOL	AAK 230	8665	Ph.D.
401	HIST	DRE 156	6783	MA
425	ECON/FIN	KLR 284	2851	MBA
435	ART	BBG 185	2278	Ph.D.

The 1:M DEPARTMENT employs PROFESSOR relationship is implemented through the placement of the DEPT_CODE foreign key in the PROFESSOR table.

The 1:1 PROFESSOR chairs DEPARTMENT relationship is implemented through the placement of the EMP_NUM foreign key in the DEPARTMENT table.

Table name: DEPARTMENT
Primary key: DEPT_CODE
Foreign key: EMP_NUM

DEPT_CODE	DEPT_NAME	SCHOOL_CODE	EMP_NUM	DEPT_ADDRESS	DEPT_EXTENSION
ACCT	Accounting	BUS	114	KLR 211, Box 52	3119
ART	Fine Arts	A&SCI	435	BBG 185, Box 128	2278
BIOL	Biology	A&SCI	387	AAK 230, Box 415	4117
CIS	Computer Info. Systems	BUS	209	KLR 333, Box 56	3245
ECON/FIN	Economics/Finance	BUS	299	KLR 284, Box 63	3126
ENG	English	A&SCI	160	DRE 102, Box 223	1004
HIST	History	A&SCI	103	DRE 156, Box 284	1867
MATH	Mathematics	A&SCI	297	AAK 194, Box 422	4234
MKT/MGT	Marketing/Management	BUS	106	KLR 126, Box 55	3342
PSYCH	Psychology	A&SCI	195	AAK 297, Box 438	4110
SOC	Sociology	A&SCI	342	BBG 208, Box 132	2008

- Also note that the PROFESSOR table contains the DEPT_CODE foreign key to implement the 1:M "DEPARTMENT employs PROFESSOR" relationship. This is a good example of how two entities can participate in two (or even more) relationships simultaneously.

The preceding "PROFESSOR chairs DEPARTMENT" example illustrates a proper 1:1 relationship. *In fact, the use of a 1:1 relationship ensures that two entity sets are not placed in the same table when they should not be.* However, the existence of a 1:1 relationship sometimes means that the entity components were not defined properly. It could indicate that the two entities actually belong in the same table!

Although 1:1 relationships should be rare, certain conditions absolutely require their use. In Chapter 5, Advanced Data Modeling, you will explore a concept called a generalization hierarchy, which is a powerful tool for improving database designs under specific conditions to avoid a proliferation of nulls. One characteristic of generalization hierarchies is that they are implemented as 1:1 relationships.

Online Content

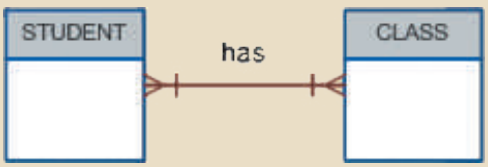
If you open the Ch03_TinyCollege database at *www.cengagebrain.com*, you will see that the STUDENT and CLASS entities still use PROF_NUM as their foreign key. PROF_NUM and EMP_NUM are labels for the same attribute, which is an example of the use of synonyms—that is, different names for the same attribute. These synonyms will be eliminated in future chapters as the Tiny College database continues to be improved.

3-6c The M:N Relationship

A many-to-many (M:N) relationship is not supported directly in the relational environment. However, M:N relationships can be implemented by creating a new entity in 1:M relationships with the original entities.

To explore the M:N relationship, consider a typical college environment. The ER model in Figure 3.23 shows this M:N relationship.

FIGURE 3.23 THE ERM'S M:N RELATIONSHIP BETWEEN STUDENT AND CLASS



Note the features of the ERM in Figure 3.23.

- Each CLASS can have many STUDENTs, and each STUDENT can take many CLASSes.
- There can be many rows in the CLASS table for any given row in the STUDENT table, and there can be many rows in the STUDENT table for any given row in the CLASS table.

To examine the M:N relationship more closely, imagine a small college with two students, each of whom takes three classes. Table 3.7 shows the enrollment data for the two students.

Online Content

If you look at the Ch03_AviaCo database at *www.cengagebrain.com*, you will see the implementation of the 1:1 PILOT to EMPLOYEE relationship. This relationship is based on a generalization hierarchy, which you will learn about in Chapter 5.

TABLE 3.7

SAMPLE STUDENT ENROLLMENT DATA

STUDENT'S LAST NAME	SELECTED CLASSES
Bowser	Accounting 1, ACCT-211, code 10014 Intro to Microcomputing, CIS-220, code 10018 Intro to Statistics, QM-261, code 10021
Smithson	Accounting 1, ACCT-211, code 10014 Intro to Microcomputing, CIS-220, code 10018 Intro to Statistics, QM-261, code 10021

Given such a data relationship and the sample data in Table 3.7, you could wrongly assume that you could implement this M:N relationship simply by adding a foreign key in the "many" side of the relationship that points to the primary key of the related table, as shown in Figure 3.24.

FIGURE 3.24 THE WRONG IMPLEMENTATION OF THE M:N RELATIONSHIP BETWEEN STUDENT AND CLASS

Table name: STUDENT
Primary key: STU_NUM
Foreign key: none

Database name: Ch03_CollegeTry

STU_NUM	STU_LNAME	CLASS_CODE
321452	Bowser	10014
321452	Bowser	10018
321452	Bowser	10021
324257	Smithson	10014
324257	Smithson	10018
324257	Smithson	10021

Table name: CLASS
Primary key: CLASS_CODE
Foreign key: STU_NUM

CLASS_CODE	STU_NUM	CRS_CODE	CLASS_SECTION	CLASS_TIME	CLASS_ROOM	PROF_NUM
10014	321452	ACCT-211	3	TTh 2:30-3:45 p.m.	BUS252	342
10014	324257	ACCT-211	3	TTh 2:30-3:45 p.m.	BUS252	342
10018	321452	CIS-220	2	MWF 9:00-9:50 a.m.	KLR211	114
10018	324257	CIS-220	2	MWF 9:00-9:50 a.m.	KLR211	114
10021	321452	QM-261	1	MWF 8:00-8:50 a.m.	KLR200	114
10021	324257	QM-261	1	MWF 8:00-8:50 a.m.	KLR200	114

However, the M:N relationship should *not* be implemented as shown in Figure 3.24 for two good reasons:

- The tables create many redundancies. For example, note that the STU_NUM values occur many times in the STUDENT table. In a real-world situation, additional student attributes such as address, classification, major, and home phone would also be contained in the STUDENT table, and each of those attribute values would be repeated in each of the records shown here. Similarly, the CLASS table contains much duplication: each student taking the class generates a CLASS record. The problem would be even worse if the CLASS table included such attributes as credit hours and course description. Those redundancies lead to the anomalies discussed in Chapter 1.
- Given the structure and contents of the two tables, the relational operations become very complex and are likely to lead to system efficiency errors and output errors.

composite entity
An entity designed to transform an M:N relationship into two 1:M relationships. The composite entity's primary key comprises at least the primary keys of the entities that it connects. Also known as a *bridge entity or associative entity*. See also *linking table*.

bridge entity
See *composite entity*.

associative entity
See *composite entity*.

Fortunately, the problems inherent in the M:N relationship can easily be avoided by creating a **composite entity** (also referred to as a **bridge entity** or an **associative entity**). Because such a table is used to link the tables that were originally related in an M:N relationship, the composite entity structure includes—as foreign keys—*at least* the primary keys of the tables that are to be linked. The database designer has two main options when defining a composite table's primary key: use the combination of those foreign keys or create a new primary key.

Remember that each entity in the ERM is represented by a table. Therefore, you can create the composite ENROLL table shown in Figure 3.25 to link the tables CLASS and STUDENT. In this example, the ENROLL table's primary key is the combination of its foreign keys CLASS_CODE and STU_NUM. However, the designer could have decided to create a single-attribute new primary key such as ENROLL_LINE, using a different

line value to identify each ENROLL table row uniquely. (Microsoft Access users might use the Autonumber data type to generate such line values automatically.)

FIGURE 3.25 CONVERTING THE M:N RELATIONSHIP INTO TWO 1:M RELATIONSHIPS

Table name: STUDENT
Primary key: STU_NUM
Foreign key: none

Database name: Ch03_CollegeTry2

STU_NUM	STU_LNAME
321452	Bowser
324257	Smithson

Table name: ENROLL
Primary key: CLASS_CODE + STU_NUM
Foreign key: CLASS_CODE, STU_NUM

CLASS_CODE	STU_NUM	ENROLL_GRADE
10014	321452	C
10014	324257	B
10018	321452	A
10018	324257	B
10021	321452	C
10021	324257	C

Table name: CLASS
Primary key: CLASS_CODE
Foreign key: CRS_CODE

CLASS_CODE	CRS_CODE	CLASS_SECTION	CLASS_TIME	CLASS_ROOM	PROF_NUM
10014	ACCT-211	3	TTh 2:30-3:45 p.m.	BUS252	342
10018	CIS-220	2	MWF 9:00-9:50 a.m.	KLR211	114
10021	QM-261	1	MWF 8:00-8:50 a.m.	KLR200	114

Because the ENROLL table in Figure 3.25 links two tables, STUDENT and CLASS, it is also called a **linking table**. In other words, a linking table is the implementation of a composite entity.

Note

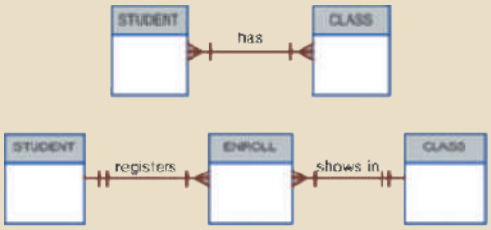
In addition to the linking attributes, the composite ENROLL table can also contain such relevant attributes as the grade earned in the course. In fact, a composite table can contain any number of attributes that the designer wants to track. Keep in mind that the composite entity, *although implemented as an actual table*, is *conceptually* a logical entity that was created as a means to an end: to eliminate the potential for multiple redundancies in the original M:N relationship.

The ENROLL table shown in Figure 3.25 yields the required M:N to 1:M conversion. Observe that the composite entity represented by the ENROLL table must contain at least the primary keys of the CLASS and STUDENT tables (CLASS_CODE and STU_NUM, respectively) for which it serves as a connector. Also note that the STUDENT and CLASS tables now contain only one row per entity. The ENROLL table contains multiple occurrences of the foreign key values, but those controlled redundancies are incapable of producing anomalies as long as referential integrity is enforced. Additional attributes may be assigned as needed. In this case, ENROLL_GRADE is selected to satisfy a reporting requirement. Also note that ENROLL_GRADE is fully dependent on the composite primary key. Naturally, the conversion is reflected in the ERM, too. The revised relationship is shown in Figure 3.26.

As you examine Figure 3.26, note that the composite entity named ENROLL represents the linking table between STUDENT and CLASS.

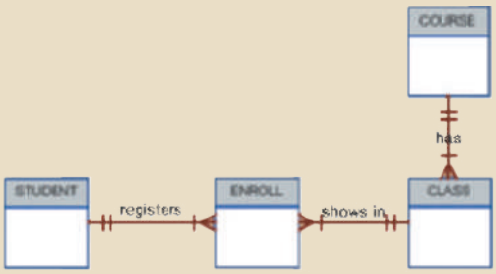
linking table
In the relational model, a table that implements an M:M relationship. See also *composite entity*.

FIGURE 3.26 CHANGING THE M:N RELATIONSHIPS TO TWO 1:M RELATIONSHIPS



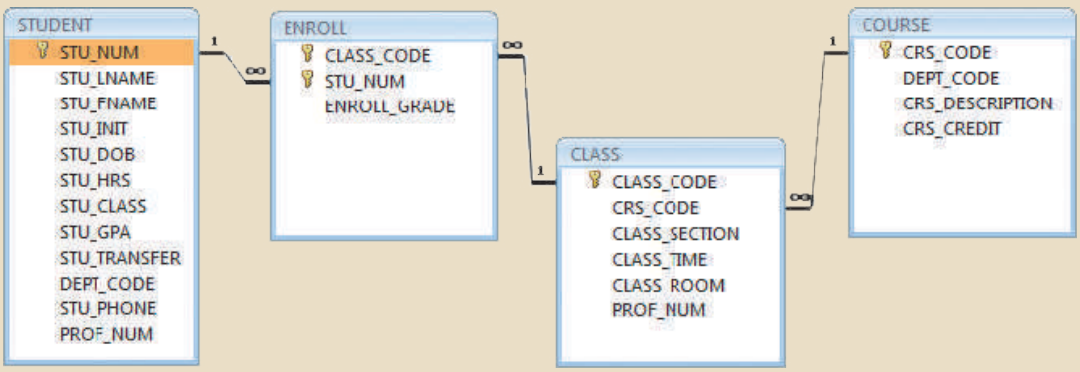
The 1:M relationship between COURSE and CLASS was first illustrated in Figure 3.19 and Figure 3.20. You can increase the amount of available information even as you control the database's redundancies. Thus, Figure 3.27 shows the expanded ERM, including the 1:M relationship between COURSE and CLASS shown in Figure 3.19. Note that the model can handle multiple sections of a CLASS while controlling redundancies by making sure that all of the COURSE data common to each CLASS are kept in the COURSE table.

FIGURE 3.27 THE EXPANDED ER MODEL



The relational diagram that corresponds to the ERM in Figure 3.27 is shown in Figure 3.28.

FIGURE 3.28 THE RELATIONAL DIAGRAM FOR THE CH03_TINYCOLLEGE DATABASE



The ERM will be examined in greater detail in Chapter 4 to show you how it is used to design more complex databases. The ERM will also be used as the basis for developing and implementing a realistic database design of a university computer lab in Appendixes B and C. These appendixes are available at *www.cengagebrain.com*.

3-7 Data Redundancy Revisited

In Chapter 1, you learned that data redundancy leads to data anomalies, which can destroy the effectiveness of the database. You also learned that the relational database makes it possible to control data redundancies by using common attributes that are shared by tables, called foreign keys.

The proper use of foreign keys is crucial to controlling data redundancy, although they do not totally eliminate the problem because the foreign key values can be repeated many times. However, the proper use of foreign keys *minimizes* data redundancies and the chances that destructive data anomalies will develop.

Note

The real test of redundancy is *not* how many copies of a given attribute are stored, *but whether the elimination of an attribute will eliminate information*. Therefore, if you delete an attribute and the original information can still be generated through relational algebra, the inclusion of that attribute would be redundant. Given that view of redundancy, proper foreign keys are clearly not redundant in spite of their multiple occurrences in a table. However, even when you use this less restrictive view of redundancy, keep in mind that *controlled* redundancies are often designed as part of the system to ensure transaction speed and/or information requirements.

You will learn in Chapter 4 that database designers must reconcile three often contradictory requirements: design elegance, processing speed, and information requirements. Also, you will learn in Chapter 13, Business Intelligence and Data Warehouses, that proper data warehousing design requires carefully defined and controlled data redundancies to function properly. Regardless of how you describe data redundancies, the potential for damage is limited by proper implementation and careful control.

As important as it is to control data redundancy, sometimes the level of data redundancy must actually be increased to make the database serve crucial information purposes. You will learn about such redundancies in Chapter 13. Also, data redundancies sometimes *seem* to exist to preserve the historical accuracy of the data. For example, consider a small invoicing system. The system includes the CUSTOMER, who may buy one or more PRODUCTs, thus generating an INVOICE. Because a customer may buy more than one product at a time, an invoice may contain several invoice LINEs, each providing details about the purchased product. The PRODUCT table should contain the product price to provide a consistent pricing input for each product that appears on the invoice. The tables that are part of such a system are shown in Figure 3.29. The system's relational diagram is shown in Figure 3.30.

As you examine the tables and relationships in the two figures, note that you can keep track of typical sales information. For example, by tracing the relationships among the four tables, you discover that customer 10014 (Myron Orlando) bought two items on March 8, 2018, that were written to invoice number 1001: one Houselite chain saw with a 16-inch bar and three rat-tail files. In other words, trace the CUS_CODE number 10014 in the CUSTOMER table to the matching CUS_CODE value in the INVOICE table. Next, trace the INV_NUMBER 1001 to the first two rows in the LINE table. Finally, match the two PROD_CODE values in LINE with the PROD_CODE values in PRODUCT. Application software will be used to write the correct bill by multiplying each invoice line item's LINE_UNITS by its LINE_PRICE, adding the results, and applying appropriate taxes. Later, other application software might use the same technique to write sales reports that track and compare sales by week, month, or year.

FIGURE 3.29 A SMALL INVOICING SYSTEM

Database name: Ch03_SaleCo

Table name: CUSTOMER
Primary key: CUS_CODE
Foreign key: none

CUS_CODE	CUS_LNAME	CUS_FNAME	CUS_INITIAL	CUS_AREACODE	CUS_PHONE
10010	Ramas	Alfred	A	615	844-2573
10011	Dunne	Leona	K	713	894-1238
10012	Smith	Kathy	W	615	894-2285
10013	Olowski	Paul	F	615	894-2180
10014	Orlando	Myron		615	222-1672
10015	O'Brian	Amy	B	713	442-3381
10016	Brown	James	G	615	297-1228
10017	Williams	George		615	290-2556
10018	Farriss	Anne	G	713	382-7185
10019	Smith	Olette	K	615	297-3809

Table name: INVOICE
Primary key: INV_NUMBER
Foreign key: CUS_CODE

INV_NUMBER	CUS_CODE	INV_DATE
1001	10014	08-Mar-18
1002	10011	08-Mar-18
1003	10012	08-Mar-18
1004	10011	09-Mar-18

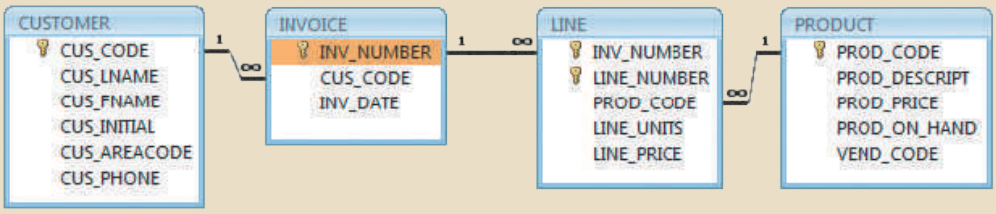
Table name: LINE
Primary key: INV_NUMBER + LINE_NUMBER
Foreign key: INV_NUMBER, PROD_CODE

INV_NUMBER	LINE_NUMBER	PROD_CODE	LINE_UNITS	LINE_PRICE
1001	1	123-21UUY	1	189.99
1001	2	SRE-657UO	3	2.99
1002	1	QER-34256	2	18.63
1003	1	ZZX/3245Q	1	6.79
1003	2	SRE-657UO	1	2.99
1003	3	001278-AB	1	12.95
1004	1	001278-AB	1	12.95
1004	2	SRE-657UO	2	2.99

Table name: PRODUCT
Primary key: PROD_CODE
Foreign key: none

PROD_CODE	PROD_DESCRIPT	PROD_PRICE	PROD_ON_HAND	VEND_CODE
001278-AB	Claw hammer	12.95	23	232
123-21UUY	Houselite chain saw, 16-in. bar	189.99	4	235
QER-34256	Sledge hammer, 16-lb. head	18.63	6	231
SRE-657UO	Rat-tail file	2.99	15	232
ZZX/3245Q	Steel tape, 12-ft. length	6.79	8	235

FIGURE 3.30 THE RELATIONAL DIAGRAM FOR THE INVOICING SYSTEM



As you examine the sales transactions in Figure 3.29, you might reasonably suppose that the product price billed to the customer is derived from the PRODUCT table because the product data is stored there. *But why does that same product price occur again in the LINE table? Is that not a data redundancy?* It certainly appears to be, but this time, the apparent redundancy is crucial to the system's success. Copying the product price from the PRODUCT table to the LINE table maintains the *historical accuracy of the transactions*. Suppose, for instance, that you fail to write the LINE_PRICE in the LINE table and that you use the PROD_PRICE from the PRODUCT table to calculate the sales revenue. Now suppose that the PRODUCT table's PROD_PRICE changes, as prices frequently do. This price change will be properly reflected in all subsequent sales revenue calculations. However, the calculations of past sales revenues will also reflect the new product price, which was not in effect when the transaction took

place! As a result, the revenue calculations for all past transactions will be incorrect, thus eliminating the possibility of making proper sales comparisons over time. On the other hand, if the price data is copied from the PRODUCT table and stored with the transaction in the LINE table, that price will always accurately reflect the transaction that took place *at that time*. You will discover that such planned "redundancies" are common in good database design.

Finally, you might wonder why the LINE_NUMBER attribute was used in the LINE table in Figure 3.29. Wouldn't the combination of INV_NUMBER and PROD_CODE be a sufficient composite primary key—and, therefore, isn't the LINE_NUMBER redundant? Yes, it is, but this redundancy is common practice on invoicing software that typically generates such line numbers automatically. In this case, the redundancy is not necessary, but given its automatic generation, the redundancy is not a source of anomalies. The inclusion of LINE_NUMBER also adds another benefit: the order of the retrieved invoicing data will always match the order in which the data was entered. If product codes are used as part of the primary key, indexing will arrange those product codes as soon as the invoice is completed and the data is stored. You can imagine the potential confusion when a customer calls and says, "The second item on my invoice has an incorrect price," and you are looking at an invoice whose lines show a different order from those on the customer's copy!

3-8 Indexes

Suppose you want to locate a book in a library. Does it make sense to look through every book until you find the one you want? Of course not; you use the library's catalog, which is indexed by title, topic, and author. The index (in either a manual or computer library catalog) points you to the book's location, making retrieval a quick and simple matter. An **index** is an orderly arrangement used to logically access rows in a table.

Or, suppose you want to find a topic in this book, such as *ER model*. Does it make sense to read through every page until you stumble across the topic? Of course not; it is much simpler to go to the book's index, look up the phrase *ER model*, and read the references that point you to the appropriate page(s). In each case, an index is used to locate a needed item quickly.

Indexes in the relational database environment work like the indexes described in the preceding paragraphs. From a conceptual point of view, an index is composed of an index key and a set of pointers. The **index key** is, in effect, the index's reference point. More formally, an index is an ordered arrangement of keys and pointers. Each key points to the location of the data identified by the key.

For example, suppose you want to look up all of the paintings created by a given painter in the Ch03_Museum database in Figure 3.18. Without an index, you must read each row in the PAINTING table and see if the PAINTER_NUM matches the requested painter. However, if you index the PAINTER table and use the index key PAINTER_NUM, you merely need to look up the appropriate PAINTER_NUM in the index and find the matching pointers. Conceptually speaking, the index would resemble the presentation in Figure 3.31.

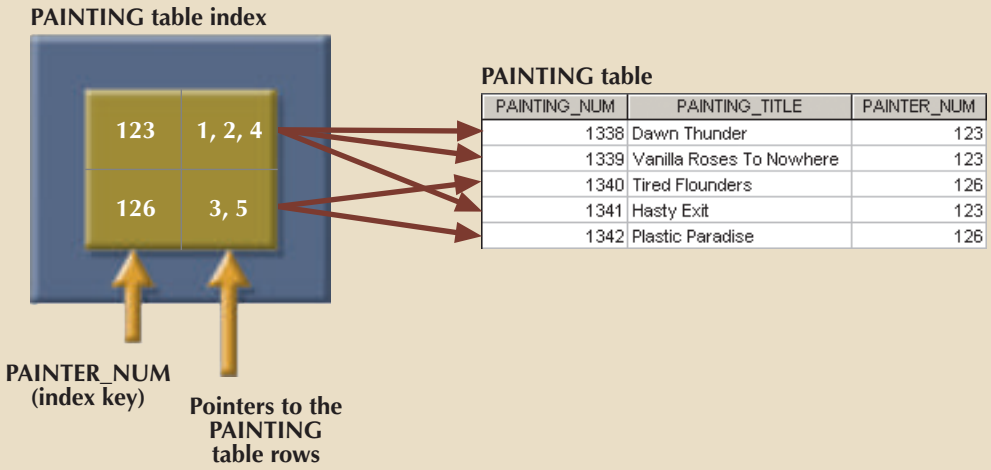
As you examine Figure 3.31, note that the first PAINTER_NUM index key value (123) is found in records 1, 2, and 4 of the PAINTING table. The second PAINTER_NUM index key value (126) is found in records 3 and 5 of the PAINTING table.

DBMSs use indexes for many different purposes. You just learned that an index can be used to retrieve data more efficiently, but indexes can also be used by a DBMS to retrieve data ordered by a specific attribute or attributes. For example, creating an index

index
An ordered array of index key values and row ID values (pointers). Indexes are generally used to speed up and facilitate data retrieval. Also known as an *index key*.

index key
See *index*.

FIGURE 3.31 COMPONENTS OF AN INDEX



on a customer's last name will allow you to retrieve the customer data alphabetically by the customer's last name. Also, an index key can be composed of one or more attributes. For example, in Figure 3.29, you can create an index on VEND_CODE and PROD_CODE to retrieve all rows in the PRODUCT table ordered by vendor and, within vendor, ordered by product.

Indexes play an important role in DBMSs for the implementation of primary keys. When you define a table's primary key, the DBMS automatically creates a unique index on the primary key column(s) you declared. For example, in Figure 3.29, when you declare CUS_CODE to be the primary key of the CUSTOMER table, the DBMS automatically creates a unique index on that attribute. In a **unique index**, as its name implies, the index key can have only one pointer value (row) associated with it. (The index in Figure 3.31 is not a unique index because the PAINTER_NUM has multiple pointer values associated with it. For example, painter number 123 points to three rows—1, 2, and 4—in the PAINTING table.)

A table can have many indexes, but each index is associated with only one table. The index key can have multiple attributes (a composite index). Creating an index is easy. You will learn in Chapter 8 that a simple SQL command produces any required index.

3-9 Codd's Relational Database Rules

In 1985, Dr. E. F. Codd published a list of 12 rules to define a relational database system.[1] He published the list out of concern that many vendors were marketing products as "relational" even though those products did not meet minimum relational standards. Dr. Codd's list, shown in Table 3.8, is a frame of reference for what a truly relational database should be. Bear in mind that even the dominant database vendors do not fully support all 12 rules.

unique index
An index in which the index key can have only one associated pointer value (row).

[1] Codd, E., "Is Your DBMS Really Relational?" and "Does Your DBMS Run by the Rules?" *Computerworld*, October 14 and 21, 1985.

TABLE 13.8

DR. CODD'S 12 RELATIONAL DATABASE RULES

RULE	RULE NAME	DESCRIPTION
1	Information	All information in a relational database must be logically represented as column values in rows within tables.
2	Guaranteed access	Every value in a table is guaranteed to be accessible through a combination of table name, primary key value, and column name.
3	Systematic treatment of nulls	Nulls must be represented and treated in a systematic way, independent of data type.
4	Dynamic online catalog based on the relational model	The metadata must be stored and managed as ordinary data—that is, in tables within the database; such data must be available to authorized users using the standard database relational language.
5	Comprehensive data sublanguage	The relational database may support many languages; however, it must support one well-defined, declarative language as well as data definition, view definition, data manipulation (interactive and by program), integrity constraints, authorization, and transaction management (begin, commit, and rollback).
6	View updating	Any view that is theoretically updatable must be updatable through the system.
7	High-level insert, update, and delete	The database must support set-level inserts, updates, and deletes.
8	Physical data independence	Application programs and ad hoc facilities are logically unaffected when physical access methods or storage structures are changed.
9	Logical data independence	Application programs and ad hoc facilities are logically unaffected when changes are made to the table structures that preserve the original table values (changing order of columns or inserting columns).
10	Integrity independence	All relational integrity constraints must be definable in the relational language and stored in the system catalog, not at the application level.
11	Distribution independence	The end users and application programs are unaware of and unaffected by the data location (distributed vs. local databases).
12	Nonsubversion	If the system supports low-level access to the data, users must not be allowed to bypass the integrity rules of the database.
13	Rule zero	All preceding rules are based on the notion that to be considered relational, a database must use its relational facilities exclusively for management.

Summary

- Tables are the basic building blocks of a relational database. A grouping of related entities, known as an entity set, is stored in a table. Conceptually speaking, the relational table is composed of intersecting rows (tuples) and columns. Each row represents a single entity, and each column represents the characteristics (attributes) of the entities.
- Keys are central to the use of relational tables. Keys define functional dependencies; that is, other attributes are dependent on the key and can therefore be found if the key value is known. A key can be classified as a superkey, a candidate key, a primary key, a secondary key, or a foreign key.
- Each table row must have a primary key. The primary key is an attribute or combination of attributes that uniquely identifies all remaining attributes found in any given row. Because a primary key must be unique, no null values are allowed if entity integrity is to be maintained.
- Although tables are independent, they can be linked by common attributes. Thus, the primary key of one table can appear as the foreign key in another table to which it is linked. Referential integrity dictates that the foreign key must contain values that match the primary key in the related table or must contain nulls.
- The relational model supports several relational algebra functions, including SELECT, PROJECT, JOIN, INTERSECT, UNION, DIFFERENCE, PRODUCT, and DIVIDE. Understanding the basic mathematical forms of these functions gives a broader understanding of the data manipulation options.
- A relational database performs much of the data manipulation work behind the scenes. For example, when you create a database, the RDBMS automatically produces a structure to house a data dictionary for your database. Each time you create a new table within the database, the RDBMS updates the data dictionary, thereby providing the database documentation.
- Once you know the basics of relational databases, you can concentrate on design. Good design begins by identifying appropriate entities and their attributes and then the relationships among the entities. Those relationships (1:1, 1:M, and M:N) can be represented using ERDs. The use of ERDs allows you to create and evaluate simple logical design. The 1:M relationship is most easily incorporated in a good design; just make sure that the primary key of the "1" is included in the table of the "many."

Key Terms

associative entity	full functional dependence	PRODUCT
attribute domain	functional dependence	PROJECT
bridge entity	homonym	referential integrity
candidate key	index	relational algebra
closure	index key	relvar
composite entity	inner join	RESTRICT
composite key	INTERSECT	right outer join
data dictionary	JOIN	secondary key
dependent	join column	SELECT
determinant	key	set theory
determination	key attribute	superkey
DIFFERENCE	left outer join	synonym
DIVIDE	linking table	system catalog
domain	natural join	theta join
entity integrity	null	tuple
equijoin	outer join	UNION
flags	predicate logic	union-compatible
foreign key (FK)	primary key (PK)	unique index

Review Questions

1. What is the difference between a database and a table?
2. What does it mean to say that a database displays both entity integrity and referential integrity?
3. Why are entity integrity and referential integrity important in a database?
4. What are the requirements that two relations must satisfy to be considered union-compatible?
5. Which relational algebra operators can be applied to a pair of tables that are not union-compatible?
6. Explain why the data dictionary is sometimes called "the database designer's database."
7. A database user manually notes that "The file contains two hundred records, each record containing nine fields." Use appropriate relational database terminology to "translate" that statement.

Use Figure Q3.8 to answer Questions 8–12.

8. Using the STUDENT and PROFESSOR tables, illustrate the difference between a natural join, an equijoin, and an outer join.
9. Create the table that would result from π_{stu_code} (student).

Online Content

All of the databases used in the questions and problems are available at *www.cengagebrain.com*. The database names match the database names shown in the figures.

10. Create the table that would result from $\pi_{stu\_code,\ dept\_code}$ (student ⋈ professor).
11. Create the basic ERD for the database shown in Figure Q3.8.
12. Create the relational diagram for the database shown in Figure Q3.8.

FIGURE Q3.8 THE CH03_COLLEGEQUE DATABASE TABLES

Database name: Ch03_CollegeQue

Table name: STUDENT

STU_CODE	PROF_CODE
100278	
128569	2
512272	4
531235	2
531268	
553427	1

Table name: PROFESSOR

PROF_CODE	DEPT_CODE
1	2
2	6
3	6
4	4

Use Figure Q3.13 to answer Questions 13–17.

FIGURE Q3.13 THE CH03_VENDINGCO DATABASE TABLES

Database name: Ch03_VendingCo

Table name: BOOTH

BOOTH_PRODUCT	BOOTH_PRICE
Chips	1.5
Cola	1.25
Energy Drink	2

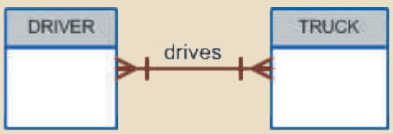
Table name: MACHINE

MACHINE_PRODUCT	MACHINE_PRICE
Chips	1.25
Chocolate Bar	1
Energy Drink	2

13. Write the relational algebra formula to apply a UNION relational operator to the tables shown in Figure Q3.13.
14. Create the table that results from applying a UNION relational operator to the tables shown in Figure Q3.13.
15. Write the relational algebra formula to apply an INTERSECT relational operator to the tables shown in Figure Q3.13.
16. Create the table that results from applying an INTERSECT relational operator to the tables shown in Figure Q3.13.
17. Using the tables in Figure Q3.13, create the table that results from MACHINE DIFFERENCE BOOTH.

Use Figure Q3.18 to answer Question 18.

FIGURE Q3.18 THE CROW'S FOOT ERD FOR DRIVER AND TRUCK



During some time interval, a DRIVER can drive many TRUCKs and any TRUCK can be driven by many DRIVERs

18. Suppose you have the ERD shown in Figure Q3.18. How would you convert this model into an ERM that displays only 1:M relationships? (Make sure you create the revised ERD.)
19. What are homonyms and synonyms, and why should they be avoided in database design?
20. How would you implement a l:M relationship in a database composed of two tables? Give an example.

Use Figure Q3.21 to answer Question 21.

FIGURE Q3.21 THE CH03_NOCOMP DATABASE EMPLOYEE TABLE

Table name: EMPLOYEE **Database name: Ch03_NoComp**

EMP_NUM	EMP_LNAME	EMP_INITIAL	EMP_FNAME	DEPT_CODE	JOB_CODE
11234	Friedman	K	Robert	MKTG	12
11238	Olanski	D	Delbert	MKTG	12
11241	Fontein		Juliette	INFS	5
11242	Cruazona	J	Maria	ENG	9
11245	Smithson	B	Bernard	INFS	6
11248	Washington	G	Oleta	ENGR	8
11256	McBride		Randall	ENGR	8
11257	Kachinn	D	Melanie	MKTG	14
11258	Smith	W	William	MKTG	14
11260	Ratula	A	Katrina	INFS	5

21. Identify and describe the components of the table shown in Figure Q3.21, using correct terminology. Use your knowledge of naming conventions to identify the table's probable foreign key(s).

Use the database shown in Figure Q3.22 to answer Questions 22–27.

FIGURE Q3.22 THE CH03_THEATER DATABASE TABLES

Database name: Ch03_Theater

Table name: DIRECTOR

DIR_NUM	DIR_LNAME	DIR_DOB
100	Broadway	12-Jan-65
101	Hollywoody	18-Nov-53
102	Goofy	21-Jun-62

Table name: PLAY

PLAY_CODE	PLAY_NAME	DIR_NUM
1001	Cat On a Cold, Bare Roof	102
1002	Hold the Mayo, Pass the Bread	101
1003	I Never Promised You Coffee	102
1004	Silly Putty Goes To Washington	100
1005	See No Sound, Hear No Sight	101
1006	Starstruck in Biloxi	102
1007	Stranger In Parrot Ice	101

22. Identify the primary keys.
23. Identify the foreign keys.
24. Create the ERM.
25. Create the relational diagram to show the relationship between DIRECTOR and PLAY.
26. Suppose you wanted quick lookup capability to get a listing of all plays directed by a given director. Which table would be the basis for the INDEX table, and what would be the index key?
27. What would be the conceptual view of the INDEX table described in Question 26? Depict the contents of the conceptual INDEX table.

Problems

FIGURE P3.1 THE CH03_STORECO DATABASE TABLES

Table name: EMPLOYEE **Database name: Ch03_StoreCo**

EMP_CODE	EMP_TITLE	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_DOB	STORE_CODE
1	Mr.	Williamson	John	W	21-May-84	3
2	Ms.	Ratula	Nancy		09-Feb-89	2
3	Ms.	Greenboro	Lottie	R	02-Oct-81	4
4	Mrs.	Rumpersfro	Jennie	S	01-Jun-91	5
5	Mr.	Smith	Robert	L	23-Nov-79	3
6	Mr.	Renselaer	Cary	A	25-Dec-85	1
7	Mr.	Ogallo	Roberto	S	31-Jul-82	3
8	Ms.	Johnsson	Elizabeth	I	10-Sep-88	1
9	Mr.	Eindsmar	Jack	W	19-Apr-75	2
10	Mrs.	Jones	Rose	R	06-Mar-86	4
11	Mr.	Broderick	Tom		21-Oct-92	3
12	Mr.	Washington	Alan	Y	08-Sep-94	2
13	Mr.	Smith	Peter	N	25-Aug-84	3
14	Ms.	Smith	Sherry	H	25-May-86	4
15	Mr.	Olenko	Howard	U	24-May-84	5
16	Mr.	Archialo	Barry	V	03-Sep-80	5
17	Ms.	Grimaldo	Jeanine	K	12-Nov-90	4
18	Mr.	Rosenberg	Andrew	D	24-Jan-91	4
19	Mr.	Rosten	Peter	F	03-Oct-88	4
20	Mr.	Mckee	Robert	S	06-Mar-90	1
21	Ms.	Baumann	Jennifer	A	11-Dec-94	3

Table name: STORE

STORE_CODE	STORE_NAME	STORE_YTD_SALES	REGION_CODE	EMP_CODE
1	Access Junction	1003455.76	2	8
2	Database Corner	1421987.39	2	12
3	Tuple Charge	986783.22	1	7
4	Attribute Alley	944568.56	2	3
5	Primary Key Point	2930098.45	1	15

Table name: REGION

REGION_CODE	REGION_DESCRIPT
1	East
2	West

Use the database shown in Figure P3.1 to answer Problems 1–9.

1. For each table, identify the primary key and the foreign key(s). If a table does not have a foreign key, write *None*.
2. Do the tables exhibit entity integrity? Answer yes or no, and then explain your answer.
3. Do the tables exhibit referential integrity? Answer yes or no, and then explain your answer. Write *NA* (Not Applicable) if the table does not have a foreign key.
4. Describe the type(s) of relationship(s) between STORE and REGION.
5. Create the ERD to show the relationship between STORE and REGION.
6. Create the relational diagram to show the relationship between STORE and REGION.
7. Describe the type(s) of relationship(s) between EMPLOYEE and STORE. (*Hint*: Each store employs many employees, one of whom manages the store.)

8. Create the ERD to show the relationships among EMPLOYEE, STORE, and REGION.
9. Create the relational diagram to show the relationships among EMPLOYEE, STORE, and REGION.

FIGURE P3.10 THE CH03_BENECO DATABASE TABLES

Database name: Ch03_BeneCo

Table name: EMPLOYEE

EMP_CODE	EMP_LNAME	JOB_CODE
14	Rudell	2
15	McDade	1
16	Ruellardo	1
17	Smith	3
20	Smith	2

Table name: BENEFIT

EMP_CODE	PLAN_CODE
15	2
15	3
16	1
17	1
17	3
17	4
20	3

Table name: JOB

JOB_CODE	JOB_DESCRIPTION
1	Clerical
2	Technical
3	Managerial

Table name: PLAN

PLAN_CODE	PLAN_DESCRIPTION
1	Term life
2	Stock purchase
3	Long-term disability
4	Dental

Use the database shown in Figure P3.10 to work Problems 10–16. Note that the database is composed of four tables that reflect these relationships:

- An EMPLOYEE has only one JOB_CODE, but a JOB_CODE can be held by many EMPLOYEEs.
- An EMPLOYEE can participate in many PLANs, and any PLAN can be assigned to many EMPLOYEEs.

Note also that the M:N relationship has been broken down into two 1:M relationships for which the BENEFIT table serves as the composite or bridge entity.

10. For each table in the database, identify the primary key and the foreign key(s). If a table does not have a foreign key, write *None*.
11. Create the ERD to show the relationship between EMPLOYEE and JOB.
12. Create the relational diagram to show the relationship between EMPLOYEE and JOB.
13. Do the tables exhibit entity integrity? Answer yes or no, and then explain your answer.
14. Do the tables exhibit referential integrity? Answer yes or no, and then explain your answer. Write *NA* (Not Applicable) if the table does not have a foreign key.
15. Create the ERD to show the relationships among EMPLOYEE, BENEFIT, JOB, and PLAN.
16. Create the relational diagram to show the relationships among EMPLOYEE, BENEFIT, JOB, and PLAN.

FIGURE P3.17 THE CH03_TRANSCO DATABASE TABLES

Table name: TRUCK **Database name: Ch03_TransCo**
Primary key: TRUCK_NUM
Foreign key: BASE_CODE, TYPE_CODE

TRUCK_NUM	BASE_CODE	TYPE_CODE	TRUCK_MILES	TRUCK_SERIAL_NUM
1001	501	1	32123.5	AA-322-12212-W11
1002	502	1	76984.3	AC-342-22134-Q23
1003	501	2	12346.6	AC-445-78656-Z99
1004		1	2894.3	WQ-112-23144-T34
1005	503	2	45673.1	FR-998-32245-W12
1006	501	2	193245.7	AD-456-00845-R45
1007	502	3	32012.3	AA-341-96573-Z84
1008	502	3	44213.6	DR-559-22189-D33
1009	503	2	10932.9	DE-887-98456-E94

Table name: BASE
Primary key: BASE_CODE
Foreign key: none

BASE_CODE	BASE_CITY	BASE_STATE	BASE_AREA_CODE	BASE_PHONE	BASE_MANAGER
501	Murfreesboro	TN	615	123-4567	Andrea D. Gallager
502	Lexington	KY	568	234-5678	George H. Delarosa
503	Cape Girardeau	MO	456	345-6789	Maria J. Talindo
504	Dalton	GA	901	456-7890	Peter F. McAvee

Table name: TYPE
Primary key: TYPE_CODE
Foreign key: none

TYPE_CODE	TYPE_DESCRIPTION
1	Single box, double-axle
2	Single box, single-axle
3	Tandem trailer, single-axle

Use the database shown in Figure P3.17 to answer Problems 17–23.

17. For each table, identify the primary key and the foreign key(s). If a table does not have a foreign key, write *None*.
18. Do the tables exhibit entity integrity? Answer yes or no, and then explain your answer.
19. Do the tables exhibit referential integrity? Answer yes or no, and then explain your answer. Write *NA* (Not Applicable) if the table does not have a foreign key.
20. Identify the TRUCK table's candidate key(s).
21. For each table, identify a superkey and a secondary key.
22. Create the ERD for this database.
23. Create the relational diagram for this database.

FIGURE P3.24 THE CH03_AVIACO DATABASE TABLES

Table name: CHARTER **Database name: Ch03_AviaCo**

CHAR_TRIP	CHAR_DATE	CHAR_PILOT	CHAR_COPILOT	AC_NUMBER	CHAR_DESTINATION	CHAR_DISTANCE	CHAR_HOURS_FLOWN	CHAR_HOURS_WAIT	CHAR_FUEL_GALLONS	CHAR_OIL_QTS	CUS_CODE
10001	05-Feb-18	104		2289L	ATL	936.0	5.1	2.2	354.1	1	10011
10002	05-Feb-18	101		2778V	BNA	328.0	1.6	0.0	72.6	0	10016
10003	05-Feb-18	105	109	4278Y	GNV	1574.0	7.8	0.0	339.8	2	10014
10004	06-Feb-18	106		1484P	STL	472.0	2.9	4.9	97.2	1	10019
10005	06-Feb-18	101		2289L	ATL	1023.0	5.7	3.5	397.7	2	10011
10006	06-Feb-18	109		4278Y	STL	472.0	2.5	5.2	117.1	0	10017
10007	06-Feb-18	104	105	2778V	GNV	1574.0	7.9	0.0	348.4	2	10012
10008	07-Feb-18	106		1484P	TYS	644.0	4.1	0.0	140.6	1	10014
10009	07-Feb-18	105		2289L	GNV	1574.0	6.6	23.4	459.9	0	10017
10010	07-Feb-18	109		4278Y	ATL	998.0	6.2	3.2	279.7	0	10016
10011	07-Feb-18	101	104	1484P	BNA	352.0	1.9	5.3	99.4	1	10012
10012	08-Feb-18	101		2778V	MOB	864.0	4.8	4.2	215.1	0	10010
10013	08-Feb-18	105		4278Y	TYS	644.0	3.9	4.5	174.3	1	10011
10014	09-Feb-18	106		4278Y	ATL	936.0	6.1	2.1	382.6	0	10017
10015	09-Feb-18	104	101	2289L	GNV	1645.0	6.7	0.0	459.5	2	10016
10016	09-Feb-18	109	105	2778V	MGV	312.0	1.5	0.0	67.2	0	10011
10017	10-Feb-18	101		1484P	STL	508.0	3.1	0.0	185.5	0	10014
10018	10-Feb-18	105	104	4278Y	TYS	644.0	3.8	4.5	167.4	0	10017

The destinations are indicated by standard three-letter airport codes. For example,
STL = St. Louis, MO ATL = Atlanta, GA BNA = Nashville, TN

Table name: AIRCRAFT

AC_NUMBER	MOD_CODE	AC_TTAF	AC_TTEL	AC_TTER
1484P	PA23-250	1833.1	1833.1	101.8
2289L	C-90A	4243.8	768.9	1123.4
2778V	PA31-350	7992.9	1513.1	789.5
4278Y	PA31-350	2147.3	622.1	243.2

AC-TTAF = Aircraft total time, airframe (hours)
AC-TTEL = Total time, left engine (hours)
AC_TTER = Total time, right engine (hours)

In a fully developed system, such attribute values would be updated by application software when the CHARTER table entries were posted.

Table name: MODEL

MOD_CODE	MOD_MANUFACTURER	MOD_NAME	MOD_SEATS	MOD_CHG_MILE
B200	Beechcraft	Super KingAir	10	1.93
C-90A	Beechcraft	KingAir	8	2.67
PA23-250	Piper	Aztec	6	1.93
PA31-350	Piper	Navajo Chieftain	10	2.35

Customers are charged per round-trip mile, using the MOD_CHG_MILE rate. The MOD_SEATS column lists the total number of seats in the airplane, including the pilot and copilot seats. Therefore, a PA31-350 trip that is flown by a pilot and a copilot has eight passenger seats available.

Use the database shown in Figure P3.24 to answer Problems 24–31. AviaCo is an aircraft charter company that supplies on-demand charter flight services using a fleet of four aircraft. Aircraft are identified by a unique registration number. Therefore, the aircraft registration number is an appropriate primary key for the AIRCRAFT table.

FIGURE P3.24 THE CH03_AVIACO DATABASE TABLES (CONTINUED)

Table name: PILOT **Database name: Ch03_AviaCo**

EMP_NUM	PIL_LICENSE	PIL_RATINGS	PIL_MED_TYPE	PIL_MED_DATE	PIL_PT135_DATE
101	ATP	ATP/SEL/MEL/Instr/CFII	1	20-Jan-18	11-Jan-18
104	ATP	ATP/SEL/MEL/Instr	1	18-Dec-17	17-Jan-18
105	COM	COMM/SEL/MEL/Instr/CFI	2	05-Jan-18	02-Jan-18
106	COM	COMM/SEL/MEL/Instr	2	10-Dec-17	02-Feb-18
109	COM	ATP/SEL/MEL/SES/Instr/CFII	1	22-Jan-18	15-Jan-18

The pilot licenses shown in the PILOT table include the ATP = Airline Transport Pilot and COM = Commercial Pilot. Businesses that operate "on demand" air services are governed by Part 135 of the Federal Air Regulations (FARs) that are enforced by the Federal Aviation Administration (FAA). Such businesses are known as "Part 135 operators." Part 135 operations require that pilots successfully complete flight proficiency checks each six months. The "Part 135" flight proficiency check date is recorded in PIL_PT135_DATE. To fly commercially, pilots must have at least a commercial license and a 2nd class medical certificate (PIL_MED_TYPE = 2.)

The PIL_RATINGS include
SEL = Single Engine, Land
SES = Single Engine (Sea)
CFI = Certified Flight Instructor
MEL = Multi-engine Land
Instr. = Instrument
CFII = Certified Flight Instructor, Instrument

Table name: EMPLOYEE

EMP_NUM	EMP_TITLE	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_DOB	EMP_HIRE_DATE
100	Mr.	Kolmycz	George	D	15-Jun-62	15-Mar-08
101	Ms.	Lewis	Rhonda	G	19-Mar-85	25-Apr-06
102	Mr.	Vandam	Rhett		14-Nov-78	18-May-13
103	Ms.	Jones	Anne	M	11-May-94	26-Jul-17
104	Mr.	Lange	John	P	12-Jul-91	20-Aug-10
105	Mr.	Williams	Robert	D	14-Mar-95	19-Jun-17
106	Mrs.	Duzak	Jeanine	K	12-Feb-88	13-Mar-18
107	Mr.	Diante	Jorge	D	01-May-95	02-Jul-16
108	Mr.	Wiesenbach	Paul	R	14-Feb-86	03-Jun-13
109	Ms.	Travis	Elizabeth	K	18-Jun-81	14-Feb-16
110	Mrs.	Genkazi	Leighla	W	19-May-90	29-Jun-10

Table name: CUSTOMER

CUS_CODE	CUS_LNAME	CUS_FNAME	CUS_INITIAL	CUS_AREACODE	CUS_PHONE	CUS_BALANCE
10010	Ramas	Alfred	A	615	844-2573	0.00
10011	Dunne	Leona	K	713	894-1238	0.00
10012	Smith	Kathy	W	615	894-2285	896.54
10013	Olowski	Paul	F	615	894-2180	1285.19
10014	Orlando	Myron		615	222-1672	673.21
10015	O'Brian	Amy	B	713	442-3381	1014.56
10016	Brown	James	G	615	297-1228	0.00
10017	Williams	George		615	290-2556	0.00
10018	Farriss	Anne	G	713	382-7185	0.00
10019	Smith	Olette	K	615	297-3809	453.98

The nulls in the CHARTER table's CHAR_COPILOT column indicate that a copilot is not required for some charter trips or for some aircraft. Federal Aviation Administration (FAA) rules require a copilot on jet aircraft and on aircraft that have a gross take-off weight over 12,500 pounds. None of the aircraft in the AIRCRAFT table are governed by this requirement; however, some customers may require the presence of a copilot for insurance reasons. All charter trips are recorded in the CHARTER table.

Note

Earlier in the chapter, you were instructed to avoid homonyms and synonyms. In this problem, both the pilot and the copilot are listed in the PILOT table, but EMP_NUM cannot be used for both in the CHARTER table. Therefore, the synonyms CHAR_PILOT and CHAR_COPILOT were used in the CHARTER table.

Although the solution works in this case, it is very restrictive, and it generates nulls when a copilot is not required. Worse, such nulls proliferate as crew requirements change. For example, if the AviaCo charter company grows and starts using larger aircraft, crew requirements may increase to include flight engineers and load masters. The CHARTER table would then have to be modified to include the additional crew assignments; such attributes as CHAR_FLT_ENGINEER and CHAR_LOADMASTER would have to be added to the CHARTER table. Given this change, each time a smaller aircraft flew a charter trip without the number of crew members required in larger aircraft, the missing crew members would yield additional nulls in the CHARTER table.

You will have a chance to correct those design shortcomings in Problem 27. The problem illustrates two important points:

1. Don't use synonyms. If your design requires the use of synonyms, revise the design!
2. To the greatest possible extent, design the database to accommodate growth without requiring structural changes in the database tables. Plan ahead and try to anticipate the effects of change on the database.

24. For each table, identify each of the following when possible:
    a. The primary key
    b. A superkey
    c. A candidate key
    d. The foreign key(s)
    e. A secondary key
25. Create the ERD. (*Hint*: Look at the table contents. You will discover that an AIRCRAFT can fly many CHARTER trips but that each CHARTER trip is flown by one AIRCRAFT, that a MODEL references many AIRCRAFT but that each AIRCRAFT references a single MODEL, and so on.)
26. Create the relational diagram.
27. Modify the ERD you created in Problem 25 to eliminate the problems created by the use of synonyms. (*Hint*: Modify the CHARTER table structure by eliminating the CHAR_PILOT and CHAR_COPILOT attributes; then create a composite table named CREW to link the CHARTER and EMPLOYEE tables. Some crew members, such as flight attendants, may not be pilots. That's why the EMPLOYEE table enters into this relationship.)
28. Create the relational diagram for the design you revised in Problem 27.

You want to see data on charters flown by either Robert Williams (employee number 105) or Elizabeth Travis (employee number 109) as pilot or copilot, but not charters flown by both of them. Complete Problems 29–31 to find this information.

29. Create the table that would result from applying the SELECT and PROJECT relational operators to the CHARTER table to return only the CHAR_TRIP, CHAR_PILOT, and CHAR_COPILOT attributes for charters flown by either employee 105 or employee 109.
30. Create the table that would result from applying the SELECT and PROJECT relational operators to the CHARTER table to return only the CHAR_TRIP, CHAR_PILOT, and CHAR_COPILOT attributes for charters flown by both employee 105 and employee 109.
31. Create the table that would result from applying a DIFFERENCE relational operator of your result from Problem 29 to your result from Problem 30.

Chapter 4

Entity Relationship (ER) Modeling

After completing this chapter, you will be able to:

- Identify the main characteristics of entity relationship components
- Describe how relationships between entities are defined, refined, and incorporated into the database design process
- See how ERD components affect database design and implementation
- Understand that real-world database design often requires the reconciliation of conflicting goals

Preview

This chapter expands coverage of the data-modeling aspect of database design. Data modeling is the first step in the database design journey, serving as a bridge between real-world objects and the database model that is implemented in the computer. Therefore, the importance of data-modeling details, expressed graphically through entity relationship diagrams (ERDs), cannot be overstated.

Most of the basic concepts and definitions used in the entity relationship model (ERM) were introduced in Chapter 2, Data Models. For example, the basic components of entities and relationships and their representation should now be familiar to you. This chapter goes much deeper, analyzing the graphic depiction of relationships among the entities and showing how those depictions help you summarize the wealth of data required to implement a successful design.

Finally, the chapter illustrates how conflicting goals can be a challenge in database design and might require design compromises.

Data Files and Available Formats

	MS Access	Oracle	MS SQL	My SQL		MS Access	Oracle	MS SQL	My SQL
CH04_TinyCollege	✓	✓	✓	✓	CH04_Clinic	✓	✓	✓	✓
CH04_TinyCollege_Alt	✓	✓	✓	✓	CH04_PartCo	✓	✓	✓	✓
CH04_ShortCo	✓	✓	✓	✓	CH04_CollegeTry	✓	✓	✓	✓

Data Files Available on cengagebrain.com

Note

Because this book generally focuses on the relational model, you might be tempted to conclude that the ERM is exclusively a relational tool. Actually, conceptual models such as the ERM can be used to understand and design the data requirements of an organization. Therefore, the ERM is independent of the database type. Conceptual models are used in the conceptual design of databases, while relational models are used in the logical design of databases. However, because you are familiar with the relational model from the previous chapter, the relational model is used extensively in this chapter to explain ER constructs and the way they are used to develop database designs.

4-1 The Entity Relationship Model

Recall from Chapter 2, Data Models, and Chapter 3, The Relational Database Model, that the entity relationship model (ERM) forms the basis of an ERD. The ERD represents the conceptual database as viewed by the end user. ERDs depict the database's main components: entities, attributes, and relationships. Because an entity represents a real-world object, the words *entity* and *object* are often used interchangeably. Thus, the entities (objects) of the Tiny College database design developed in this chapter include students, classes, teachers, and classrooms. The order in which the ERD components are covered in the chapter is dictated by the way the modeling tools are used to develop ERDs that can form the basis for successful database design and implementation.

In Chapter 2, you also learned about the various notations used with ERDs—the original Chen notation and the newer Crow's Foot and UML notations. The first two notations are used at the beginning of this chapter to introduce some basic ER modeling concepts. Some conceptual database modeling concepts can be expressed only using the Chen notation. However, because the emphasis is on *design and implementation* of databases, the Crow's Foot and UML class diagram notations are used for the final Tiny College ER diagram example. Because of its emphasis on implementation, the Crow's Foot notation can represent only what could be implemented. In other words:

- The Chen notation favors conceptual modeling.
- The Crow's Foot notation favors a more implementation-oriented approach.
- The UML notation can be used for both conceptual and implementation modeling.

Online Content

To learn how to create ER diagrams with the help of Microsoft Visio, go to *www.cengagebrain.com:*

- *Appendix A, Designing Databases with Visio Professional: A Tutorial* shows you how to create Crow's Foot ERDs.
- *Appendix H, Unified Modeling Language (UML)*, shows you how to create UML class diagrams.

4-1a Entities

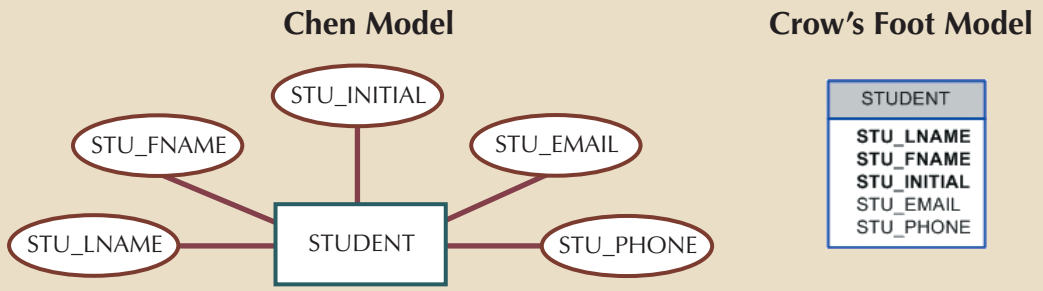
An entity is an object of interest to the end user. In Chapter 2, you learned that, at the ER modeling level, an entity actually refers to the *entity set* and not to a single entity occurrence. In other words, an *entity* in the ERM corresponds to a table—not to a row—in the relational environment. The ERM refers to a table row as an *entity instance* or *entity occurrence*. In the Chen, Crow's Foot, and UML notations, an entity is represented by a rectangle that contains the entity's name. The entity name, a noun, is usually written in all capital letters.

4-1b Attributes

Attributes are characteristics of entities. For example, the STUDENT entity includes the attributes STU_LNAME, STU_FNAME, and STU_INITIAL, among many others. In the original Chen notation, attributes are represented by ovals and are connected to the entity rectangle with a line. Each oval contains the name of the attribute it represents. In

the Crow's Foot notation, the attributes are written in the attribute box below the entity rectangle. (See Figure 4.1.) Because the Chen representation consumes more space, software vendors have adopted the Crow's Foot attribute display.

FIGURE 4.1 THE ATTRIBUTES OF THE STUDENT ENTITY: CHEN AND CROW'S FOOT



Required and Optional Attributes A **required attribute** is an attribute that must have a value; in other words, it cannot be left empty. As shown in Figure 4.1, the two boldfaced attributes in the Crow's Foot notation indicate that data entry will be required. STU_LNAME and STU_FNAME require data entries because all students are assumed to have a last name and a first name. However, students might not have a middle name, and perhaps they do not yet have a phone number and an email address. Therefore, those attributes are not presented in boldface in the entity box. An **optional attribute** is an attribute that does not require a value; therefore, it can be left empty.

Domains Attributes have a domain. A **domain** is the set of possible values for a given attribute. For example, the domain for a grade point average (GPA) attribute is written (0,4) because the lowest possible GPA value is 0 and the highest possible value is 4. The domain for a gender attribute consists of only two possibilities: M or F (or some other equivalent code). The domain for a company's date of hire attribute consists of all dates that fit in a range (e.g., company startup date to current date).

Attributes may share a domain. For instance, a student address and a professor address share the same domain of all possible addresses. In fact, the data dictionary may let a newly declared attribute inherit the characteristics of an existing attribute if the same attribute name is used. For example, the PROFESSOR and STUDENT entities may each have an attribute named ADDRESS and could therefore share a domain.

Identifiers (Primary Keys) The ERM uses **identifiers**—one or more attributes that uniquely identify each entity instance. In the relational model, entities are mapped to tables, and the entity identifier is mapped as the table's primary key (PK). Identifiers are underlined in the ERD. Key attributes are also underlined in a frequently used shorthand notation for the table structure, called a **relational schema**, that uses the following format:

TABLE NAME (**KEY_ATTRIBUTE 1**, ATTRIBUTE 2, ATTRIBUTE 3, … ATTRIBUTE K)

For example, a CAR entity may be represented by

CAR (**CAR_VIN**, MOD_CODE, CAR_YEAR, CAR_COLOR)

Each car is identified by a unique vehicle identification number, or CAR_VIN.

Composite Identifiers Ideally, an entity identifier is composed of only a single attribute. For example, the table in Figure 4.2 uses a single-attribute primary key named

domain
The possible set of values for a given attribute.

required attribute
In ER modeling, an attribute that must have a value. In other words, it cannot be left empty.

optional attribute
In ER modeling, an attribute that does not require a value; therefore, it can be left empty.

identifier
One or more attributes that uniquely identify each entity instance.

relational schema
The organization of a relational database as described by the database administrator.

CLASS_CODE. However, it is possible to use a **composite identifier**, a primary key composed of more than one attribute. For instance, the Tiny College database administrator may decide to identify each CLASS entity instance (occurrence) by using a composite primary key of CRS_CODE and CLASS_SECTION instead of using CLASS_CODE. Either approach uniquely identifies each entity instance. Given the structure of the CLASS table shown in Figure 4.2, CLASS_CODE is the primary key, and the combination of CRS_CODE and CLASS_SECTION is a proper candidate key. If the CLASS_CODE attribute is deleted from the CLASS entity, the candidate key (CRS_CODE and CLASS_SECTION) becomes an acceptable composite primary key.

FIGURE 4.2 THE CLASS TABLE (ENTITY) COMPONENTS AND CONTENTS

Database name: Ch04_TinyCollege

CLASS_CODE	CRS_CODE	CLASS_SECTION	CLASS_TIME	ROOM_CODE	PROF_NUM
10012	ACCT-211	1	MWF 8:00-8:50 a.m.	BUS311	105
10013	ACCT-211	2	MWF 9:00-9:50 a.m.	BUS200	105
10014	ACCT-211	3	TTh 2:30-3:45 p.m.	BUS252	342
10015	ACCT-212	1	MWF 10:00-10:50 a.m.	BUS311	301
10016	ACCT-212	2	Th 6:00-8:40 p.m.	BUS252	301
10017	CIS-220	1	MWF 9:00-9:50 a.m.	KLR209	228
10018	CIS-220	2	MWF 9:00-9:50 a.m.	KLR211	114
10019	CIS-220	3	MWF 10:00-10:50 a.m.	KLR209	228
10020	CIS-420	1	W 6:00-8:40 p.m.	KLR209	162
10021	QM-261	1	MWF 8:00-8:50 a.m.	KLR200	114
10022	QM-261	2	TTh 1:00-2:15 p.m.	KLR200	114
10023	QM-362	1	MWF 11:00-11:50 a.m.	KLR200	162
10024	QM-362	2	TTh 2:30-3:45 p.m.	KLR200	162
10025	MATH-243	1	Th 6:00-8:40 p.m.	DRE155	325

Note

Remember that Chapter 3 made a commonly accepted distinction between COURSE and CLASS. A CLASS constitutes a specific time and place of a COURSE offering. A class is defined by the course description and its time and place, or section. Consider a professor who teaches Database I, Section 2; Database I, Section 5; Database I, Section 8; and Spreadsheet II, Section 6. The professor teaches two courses (Database I and Spreadsheet II), but four classes. Typically, the COURSE offerings are printed in a course catalog, while the CLASS offerings are printed in a class schedule for each term.

composite identifier
In ER modeling, a key composed of more than one attribute.

composite attribute
An attribute that can be further subdivided to yield additional attributes. For example, a phone number such as 615-898-2368 may be divided into an area code (615), an exchange number (898), and a four-digit code (2368). Compare to *simple attribute*.

If the CLASS_CODE in Figure 4.2 is used as the primary key, the CLASS entity may be represented in shorthand form as follows:

CLASS (**CLASS_CODE**, CRS_CODE, CLASS_SECTION, CLASS_TIME, ROOM_CODE, PROF_NUM)

On the other hand, if CLASS_CODE is deleted, and the composite primary key is the combination of CRS_CODE and CLASS_SECTION, the CLASS entity may be represented as follows:

CLASS (**CRS_CODE, CLASS_SECTION**, CLASS_TIME, ROOM_CODE, PROF_NUM)

Note that *both* key attributes are underlined in the entity notation.

Composite and Simple Attributes

Attributes are classified as simple or composite. A **composite attribute**, not to be confused with a composite key, is an attribute that can

be further subdivided to yield additional attributes. For example, the attribute ADDRESS can be subdivided into street, city, state, and zip code. Similarly, the attribute PHONE_NUMBER can be subdivided into area code and exchange number. A **simple attribute** is an attribute that cannot be subdivided. For example, age, sex, and marital status would be classified as simple attributes. To facilitate detailed queries, it is wise to change composite attributes into a series of simple attributes.

The database designer must always be on the lookout for composite attributes. It is common for business rules to use composite attributes to simplify policies, and users often describe entities in their environment using composite attributes. For example, a user at Tiny College might need to know a student's name, address, and phone number. The designer must recognize that these are composite attributes and determine the correct way to decompose the composite into simple attributes.

simple attribute
An attribute that cannot be subdivided into meaningful components. Compare to *composite attribute*.

single-valued attribute
An attribute that can have only one value.

multivalued attribute
An attribute that can have many values for a single entity occurrence. For example, an EMP_DEGREE attribute might store the string "BBA, MBA, PHD" to indicate three different degrees held.

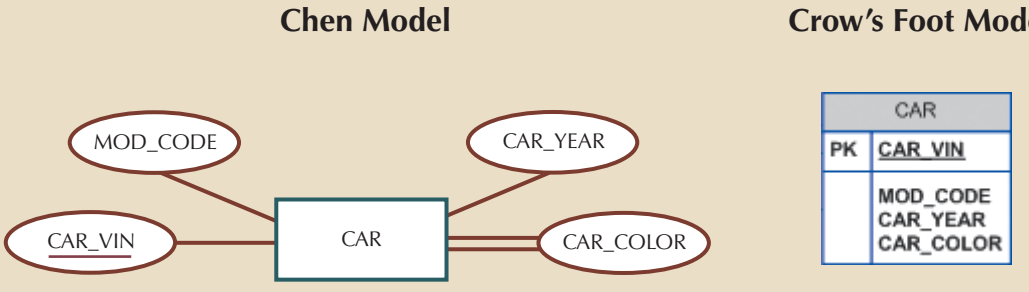
Single-Valued Attributes

A **single-valued attribute** is an attribute that can have only a single value. For example, a person can have only one Social Security number, and a manufactured part can have only one serial number. *Keep in mind that a single-valued attribute is not necessarily a simple attribute*. For instance, a part's serial number (such as SE-08-02-189935) is single-valued, but it is a composite attribute because it can be subdivided into the region in which the part was produced (SE), the plant within that region (08), the shift within the plant (02), and the part number (189935).

Multivalued Attributes

Multivalued attributes are attributes that can have many values. For instance, a person may have several college degrees, and a household may have several different phones, each with its own number. Similarly, a car's color may be subdivided into many colors for the roof, body, and trim. In the Chen ERM, multivalued attributes are shown by a double line connecting the attribute to the entity. The Crow's Foot notation does not identify multivalued attributes. The ERD in Figure 4.3 contains all of the components introduced thus far; note that CAR_VIN is the primary key, and CAR_COLOR is a multivalued attribute of the CAR entity.

FIGURE 4.3 A MULTIVALUED ATTRIBUTE IN AN ENTITY



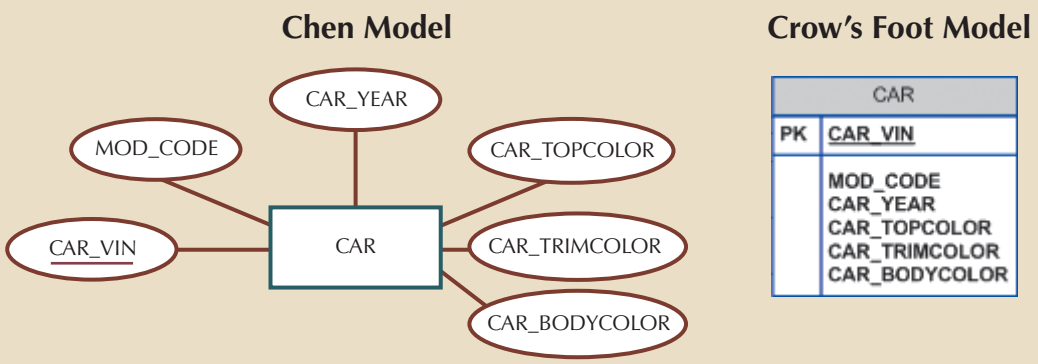
Note

In the ERD models in Figure 4.3, the CAR entity's foreign key (FK) has been typed as MOD_CODE. This attribute was manually added to the entity. Actually, proper use of database modeling software will automatically produce the FK when the relationship is defined. In addition, the software will label the FK appropriately and write the FK's implementation details in a data dictionary. (You can see how this works in Appendix A, Designing Databases with Visio Professional: A Tutorial, at *www.cengagebrain.com*.)

Implementing Multivalued Attributes Although the conceptual model can handle M:N relationships and multivalued attributes, *you should not implement them in the RDBMS*. Remember from Chapter 3 that in the relational table, each column and row intersection represents a single data value. So, if multivalued attributes exist, the designer must decide on one of two possible courses of action:

1. Within the original entity, create several new attributes, one for each component of the original multivalued attribute. For example, the CAR entity's attribute CAR_COLOR can be split to create the new attributes CAR_TOPCOLOR, CAR_BODYCOLOR, and CAR_TRIMCOLOR, which are then assigned to the CAR entity. (See Figure 4.4.)

FIGURE 4.4 SPLITTING THE MULTIVALUED ATTRIBUTE INTO NEW ATTRIBUTES



Although this solution seems to work, its adoption can lead to major structural problems in the table. It is only acceptable if every instance will have the same number of values for the multivalued attribute, and no instance will ever have more values. However, even in this case, it is a gamble that new changes in the environment will never create a situation where an instance would have more values than before. For example, if additional color components—such as a logo color—are added for some cars, the table structure must be modified to accommodate the new color section. In that case, cars that do not have such color sections generate nulls for the nonexistent components, or their color entries for those sections are entered as N/A to indicate "not applicable." (The solution in Figure 4.4 is to split a multivalued attribute into new attributes, but imagine the problems this type of solution would cause if it were applied to an employee entity that contains employee degrees and certifications. If some employees have 10 degrees and certifications while most have fewer or none, the number of degree/certification attributes would be 10, and most of those attribute values would be null for most employees.) In short, although you have seen solution 1 applied, it is not always acceptable.

2. Create a new entity composed of the original multivalued attribute's components. This new entity allows the designer to define color for different sections of the car (see Table 4.1). Then, this new CAR_COLOR entity is related to the original CAR entity in a 1:M relationship.

Using the approach illustrated in Table 4.1, you even get a fringe benefit: you can now assign as many colors as necessary without having to change the table structure. The ERM shown in Figure 4.5 reflects the components listed in Table 4.1. This is the preferred way to deal with multivalued attributes. Creating a new entity in a 1:M relationship with the original entity yields several benefits: it is a more flexible, expandable solution, and it is compatible with the relational model!

TABLE 4.1

COMPONENTS OF THE MULTIVALUED ATTRIBUTE

SECTION	COLOR
Top	White
Body	Blue
Trim	Gold
Interior	Blue

FIGURE 4.5 A NEW ENTITY SET COMPOSED OF A MULTIVALUED ATTRIBUTE'S COMPONENTS

Note

If you are used to looking at relational diagrams such as the ones produced by Microsoft Access, you expect to see the relationship line *in the relational diagram* drawn from the PK to the FK. However, the relational diagram convention is not necessarily reflected in the ERD. In an ERD, the focus is on the entities and the relationships between them, rather than how those relationships are anchored graphically. In a complex ERD that includes both horizontally and vertically placed entities, the placement of the relationship lines is largely dictated by the designer's decision to improve the readability of the design. (Remember that the ERD is used for communication between designers and end users.)

Derived Attributes Finally, a **derived attribute** is an attribute whose value is calculated (derived) from other attributes. The derived attribute need not be physically stored within the database; instead, it can be derived by using an algorithm. For example, an employee's age, EMP_AGE, may be found by computing the integer value of the difference between the current date and the EMP_DOB. If you use Microsoft Access, you would use the formula INT((DATE() – EMP_DOB)/365). In Microsoft SQL Server, you would use DATEDIFF("DAY", EMB_DOB, GETDATE())/365, where DATEDIFF is a function that computes the difference between dates. If you use Oracle, you would use TRUNC((SYSDATE – EMP_DOB)/365,0).

Similarly, the total cost of an order can be derived by multiplying the quantity ordered by the unit price. Or, the estimated average speed can be derived by dividing trip distance by the time spent in route. A derived attribute is indicated in the Chen notation by a dashed line that connects the attribute and the entity. (See Figure 4.6.) The Crow's Foot notation does not have a method for distinguishing the derived attribute from other attributes.

Derived attributes are sometimes referred to as *computed attributes*. Computing a derived attribute can be as simple as adding two attribute values located on the same row, or it can be the result of aggregating the sum of values located on many table rows (from the same table or from a different table). The decision to store derived attributes in

derived attribute
An attribute that does not physically exist within the entity and is derived via an algorithm. For example, the Age attribute might be derived by subtracting the birth date from the current date.

database tables depends on the processing requirements and the constraints placed on a particular application. The designer should be able to balance the design in accordance with such constraints. Table 4.2 shows the advantages and disadvantages of storing (or not storing) derived attributes in the database.

FIGURE 4.6 DEPICTION OF A DERIVED ATTRIBUTE

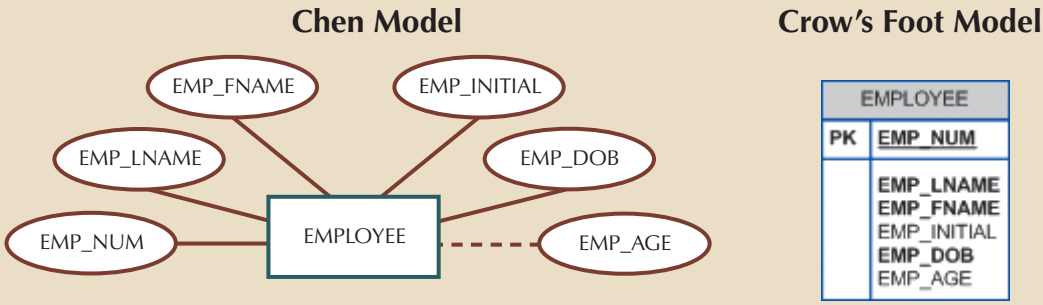


TABLE 4.2

ADVANTAGES AND DISADVANTAGES OF STORING DERIVED ATTRIBUTES

	DERIVED ATTRIBUTE	
	STORED	**NOT STORED**
Advantage	Saves CPU processing cycles Saves data access time Data value is readily available Can be used to keep track of historical data	Saves storage space Computation always yields current value
Disadvantage	Requires constant maintenance to ensure derived value is current, especially if any values used in the calculation change	Uses CPU processing cycles Increases data access time Adds coding complexity to queries

Note

Modern database management systems provide new data type definitions to support computed or calculated data. For example, in MS Access you can use the Calculated data type. SQL Server, Oracle, and MySQL also support defining derived or calculated attributes.

participants
An ER term for entities that participate in a relationship. For example, in the relationship "PROFESSOR teaches CLASS," the *teaches* relationship is based on the participants PROFESSOR and CLASS.

4-1c Relationships

Recall from Chapter 2 that a relationship is an association between entities. The entities that participate in a relationship are also known as **participants**, and each relationship is identified by a name that describes the relationship. The relationship name is an active or passive verb; for example, a STUDENT *takes* a CLASS, a PROFESSOR *teaches* a CLASS, a DEPARTMENT *employs* a PROFESSOR, a DIVISION *is managed by* an EMPLOYEE, and an AIRCRAFT *is flown by* a CREW.

Relationships between entities always operate in both directions. To define the relationship between the entities named CUSTOMER and INVOICE, you would specify that:

- A CUSTOMER may generate many INVOICEs.
- Each INVOICE is generated by one CUSTOMER.

Because you know both directions of the relationship between CUSTOMER and INVOICE, it is easy to see that this relationship can be classified as 1:M.

The relationship classification is difficult to establish if you know only one side of the relationship. For example, if you specify that:

A DIVISION is managed by one EMPLOYEE.

You don't know if the relationship is 1:1 or 1:M. Therefore, you should ask the question "Can an employee manage more than one division?" If the answer is yes, the relationship is 1:M, and the second part of the relationship is then written as:

An EMPLOYEE may manage many DIVISIONs.

If an employee cannot manage more than one division, the relationship is 1:1, and the second part of the relationship is then written as:

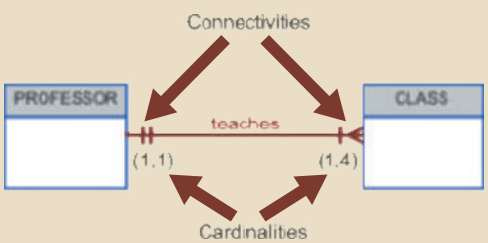
An EMPLOYEE may manage only one DIVISION.

4-1d Connectivity and Cardinality

You learned in Chapter 2 that entity relationships may be classified as one-to-one, one-to-many, or many-to-many. You also learned how such relationships were depicted in the Chen and Crow's Foot notations. The term **connectivity** is used to describe the relationship classification.

Cardinality expresses the minimum and maximum number of entity occurrences associated with one occurrence of the related entity. In the ERD, cardinality is indicated by placing the appropriate numbers beside the entities, using the format (x,y). The first value represents the minimum number of associated entities, while the second value represents the maximum number of associated entities. Many database designers who use Crow's Foot modeling notation do not depict the specific cardinalities on the ER diagram itself because the specific limits described by the cardinalities cannot be implemented directly through the database design. Correspondingly, some Crow's Foot ER modeling tools do not print the numeric cardinality range in the diagram; instead, you can add it as text if you want to have it shown. When the specific cardinalities are not included on the diagram in Crow's Foot notation, cardinality is implied by the use of the symbols shown in Figure 4.7, which describe the connectivity and participation (discussed next). The numeric cardinality range has been added using the Microsoft Visio text drawing tool.

FIGURE 4.7 CONNECTIVITY AND CARDINALITY IN AN ERD



Online Content

Because the careful definition of complete and accurate business rules is crucial to good database design, their derivation is examined in detail in *Appendix B, The University Lab: Conceptual Design*. The modeling skills you are learning in this chapter are applied in the development of a real database design in Appendix B. The initial design shown in Appendix B is then modified in *Appendix C, The University Lab: Conceptual Design Verification, Logical Design, and Implementation*. (Both appendixes are available at *www.cengagebrain.com*.)

connectivity
The classification of the relationship between entities. Classifications include 1:1, 1:M, and M:N.

cardinality
A property that assigns a specific value to connectivity and expresses the range of allowed entity occurrences associated with a single occurrence of the related entity.

Knowing the minimum and maximum number of entity occurrences is very useful at the application software level. For example, Tiny College might want to ensure that a class is not taught unless it has at least 10 students enrolled. Similarly, if the classroom can hold only 30 students, the application software should use that cardinality to limit enrollment in the class. However, keep in mind that the DBMS cannot handle the implementation of the cardinalities at the table level—that capability is provided by the application software or by triggers. You will learn how to create and execute triggers in Chapter 8, Advanced SQL.

As you examine the Crow's Foot diagram in Figure 4.7, keep in mind that the cardinalities represent the number of occurrences in the *related* entity. For example, the cardinality (1,4) next to the CLASS entity in the "PROFESSOR teaches CLASS" relationship indicates that each professor teaches up to four classes, which means that the PROFESSOR table's primary key value occurs at least once and no more than four times as foreign key values in the CLASS table. If the cardinality had been written as (1,N), there would be no upper limit to the number of classes a professor might teach. Similarly, the cardinality (1,1) next to the PROFESSOR entity indicates that each class is taught by one and only one professor. That is, each CLASS entity occurrence is associated with one and only one entity occurrence in PROFESSOR.

Connectivities and cardinalities are established by concise statements known as business rules, which were introduced in Chapter 2. Such rules, derived from a precise and detailed description of an organization's data environment, also establish the ERM's entities, attributes, relationships, connectivities, cardinalities, and constraints. Because business rules define the ERM's components, making sure that all appropriate business rules are identified is an important part of a database designer's job.

existence-dependent
A property of an entity whose existence depends on one or more other entities. In such an environment, the existence-independent table must be created and loaded first because the existence-dependent key cannot reference a table that does not yet exist.

existence-independent
A property of an entity that can exist apart from one or more related entities. Such a table must be created first when referencing an existence-dependent table.

strong entity
An entity that is existence-independent, that is, it can exist apart from all of its related entities.

regular entity
See *strong entity*.

Note

The placement of the cardinalities in the ER diagram is a matter of convention. The Chen notation places the cardinalities on the side of the related entity. The Crow's Foot and UML diagrams place the cardinalities next to the entity to which they apply.

4-1e Existence Dependence

An entity is said to be **existence-dependent** if it can exist in the database only when it is associated with another related entity occurrence. In implementation terms, an entity is existence-dependent if it has a mandatory foreign key—that is, a foreign key attribute that cannot be null. For example, if an employee wants to claim one or more dependents for tax-withholding purposes, the relationship "EMPLOYEE claims DEPENDENT" would be appropriate. In that case, the DEPENDENT entity is clearly existence-dependent on the EMPLOYEE entity because it is impossible for the dependent to exist apart from the EMPLOYEE in the database.

If an entity can exist apart from all of its related entities, then it is **existence-independent**, and it is referred to as a **strong entity** or **regular entity**. For example, suppose that the XYZ Corporation uses parts to produce its products. Furthermore, suppose that some of those parts are produced in-house and other parts are bought from vendors. In that scenario, it is quite possible for a PART to exist independently from a VENDOR in the relationship "PART is supplied by VENDOR" because at least some of the parts are not supplied by a vendor. Therefore, PART is existence-independent from VENDOR.

Note

The concept of relationship strength is not part of the original ERM. Instead, this concept applies directly to Crow's Foot diagrams. Because Crow's Foot diagrams are used extensively to design relational databases, it is important to understand relationship strength as it affects database implementation. The Chen ERD notation is oriented toward conceptual modeling and therefore does not distinguish between weak and strong relationships.

4-1f Relationship Strength

The concept of relationship strength is based on how the primary key of a related entity is defined. To implement a relationship, the primary key of one entity (the parent entity, normally on the "one" side of the one-to-many relationship) appears as a foreign key in the related entity (the child entity, mostly the entity on the "many" side of the one-to-many relationship). Sometimes, the foreign key also is a primary key component in the related entity. For example, in Figure 4.5, the CAR entity primary key (CAR_VIN) appears as both a primary key component and a foreign key in the CAR_COLOR entity. In this section, you will learn how various relationship strength decisions affect primary key arrangement in database design.

Weak (Non-Identifying) Relationships A **weak relationship**, also known as a **non-identifying relationship**, exists if the primary key of the related entity does not contain a primary key component of the parent entity. By default, relationships are established by having the primary key of the parent entity appear as a foreign key (FK) on the related entity (also known as the child entity). For example, suppose the 1:M relationship between COURSE and CLASS is defined as:

COURSE (**CRS_CODE**, DEPT_CODE, CRS_DESCRIPTION, CRS_CREDIT)

CLASS (**CLASS_CODE**, CRS_CODE, CLASS_SECTION, CLASS_TIME, ROOM_CODE, PROF_NUM)

In this example, the CLASS primary key did not inherit a primary key component from the COURSE entity. In this case, a weak relationship exists between COURSE and CLASS because CRS_CODE (the primary key of the parent entity) is only a foreign key in the CLASS entity.

Figure 4.8 shows how the Crow's Foot notation depicts a weak relationship by placing a dashed relationship line between the entities. The tables shown below the ERD illustrate how such a relationship is implemented.

weak (non-identifying) relationship
A relationship in which the primary key of the related entity does not contain a primary key component of the parent entity.

Strong (Identifying) Relationships A **strong (identifying) relationship** exists when the primary key of the related entity contains a primary key component of the parent entity. For example, suppose the 1:M relationship between COURSE and CLASS is defined as:

COURSE (**CRS_CODE**, DEPT_CODE, CRS_DESCRIPTION, CRS_CREDIT)

CLASS (**CRS_CODE, CLASS_SECTION**, CLASS_TIME, ROOM_CODE, PROF_NUM)

In this case, the CLASS entity primary key is composed of CRS_CODE and CLASS_SECTION. Therefore, a strong relationship exists between COURSE and CLASS because CRS_CODE (the primary key of the parent entity) is a primary key component in the CLASS entity. In other words, the CLASS primary key did inherit a primary key

strong (identifying) relationship
A relationship that occurs when two entities are existence-dependent; from a database design perspective, this relationship exists whenever the primary key of the related entity contains the primary key of the parent entity.

FIGURE 4.8 A WEAK (NON-IDENTIFYING) RELATIONSHIP BETWEEN COURSE AND CLASS

Table name: COURSE **Database name: Ch04_TinyCollege**

CRS_CODE	DEPT_CODE	CRS_DESCRIPTION	CRS_CREDIT
ACCT-211	ACCT	Accounting I	3
ACCT-212	ACCT	Accounting II	3
CIS-220	CIS	Intro. to Microcomputing	3
CIS-420	CIS	Database Design and Implementation	4
MATH-243	MATH	Mathematics for Managers	3
QM-261	CIS	Intro. to Statistics	3
QM-362	CIS	Statistical Applications	4

Table name: CLASS

CLASS_CODE	CRS_CODE	CLASS_SECTION	CLASS_TIME	ROOM_CODE	PROF_NUM
10012	ACCT-211	1	MWF 8:00-8:50 a.m.	BUS311	105
10013	ACCT-211	2	MWF 9:00-9:50 a.m.	BUS200	105
10014	ACCT-211	3	TTh 2:30-3:45 p.m.	BUS252	342
10015	ACCT-212	1	MWF 10:00-10:50 a.m.	BUS311	301
10016	ACCT-212	2	Th 6:00-8:40 p.m.	BUS252	301
10017	CIS-220	1	MWF 9:00-9:50 a.m.	KLR209	228
10018	CIS-220	2	MWF 9:00-9:50 a.m.	KLR211	114
10019	CIS-220	3	MWF 10:00-10:50 a.m.	KLR209	228
10020	CIS-420	1	W 6:00-8:40 p.m.	KLR209	162
10021	QM-261	1	MWF 8:00-8:50 a.m.	KLR200	114
10022	QM-261	2	TTh 1:00-2:15 p.m.	KLR200	114
10023	QM-362	1	MWF 11:00-11:50 a.m.	KLR200	162
10024	QM-362	2	TTh 2:30-3:45 p.m.	KLR200	162
10025	MATH-243	1	Th 6:00-8:40 p.m.	DRE155	325

component from the COURSE entity. (Note that the CRS_CODE in CLASS is *also* the FK to the COURSE entity.)

The Crow's Foot notation depicts the strong (identifying) relationship with a solid line between the entities, as shown in Figure 4.9.

As you examine Figure 4.9, you might wonder what the O symbol next to the CLASS entity signifies. You will discover the meaning of this cardinality in Section 4-1h, Relationship Participation.

In summary, whether the relationship between COURSE and CLASS is strong or weak depends on how the CLASS entity's primary key is defined. Remember that the nature of the relationship is often determined by the database designer, who must use professional judgment to determine which relationship type and strength best suit the database transaction, efficiency, and information requirements. That point will be emphasized in detail!

FIGURE 4.9 A STRONG (IDENTIFYING) RELATIONSHIP BETWEEN COURSE AND CLASS

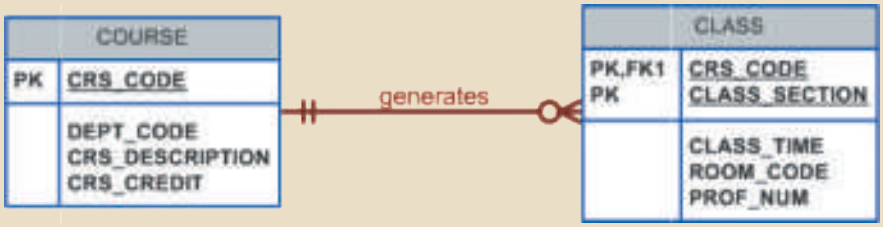


Table name: COURSE **Database name: Ch04_TinyCollege_Alt**

CRS_CODE	DEPT_CODE	CRS_DESCRIPTION	CRS_CREDIT
ACCT-211	ACCT	Accounting I	3
ACCT-212	ACCT	Accounting II	3
CIS-220	CIS	Intro. to Microcomputing	3
CIS-420	CIS	Database Design and Implementation	4
MATH-243	MATH	Mathematics for Managers	3
QM-261	CIS	Intro. to Statistics	3
QM-362	CIS	Statistical Applications	4

Table name: CLASS

CRS_CODE	CLASS_SECTION	CLASS_TIME	ROOM_CODE	PROF_NUM
ACCT-211	1	MWF 8:00-8:50 a.m.	BUS311	105
ACCT-211	2	MWF 9:00-9:50 a.m.	BUS200	105
ACCT-211	3	TTh 2:30-3:45 p.m.	BUS252	342
ACCT-212	1	MWF 10:00-10:50 a.m.	BUS311	301
ACCT-212	2	Th 6:00-8:40 p.m.	BUS252	301
CIS-220	1	MWF 9:00-9:50 a.m.	KLR209	228
CIS-220	2	MWF 9:00-9:50 a.m.	KLR211	114
CIS-220	3	MWF 10:00-10:50 a.m.	KLR209	228
CIS-420	1	W 6:00-8:40 p.m.	KLR209	162
MATH-243	1	Th 6:00-8:40 p.m.	DRE155	325
QM-261	1	MWF 8:00-8:50 a.m.	KLR200	114
QM-261	2	TTh 1:00-2:15 p.m.	KLR200	114
QM-362	1	MWF 11:00-11:50 a.m.	KLR200	162
QM-362	2	TTh 2:30-3:45 p.m.	KLR200	162

Note

Keep in mind that the *order in which the tables are created and loaded is very important*. For example, in the "COURSE generates CLASS" relationship, the COURSE table must be created before the CLASS table. After all, it would not be acceptable to have the CLASS table's foreign key refer to a COURSE table that did not yet exist. In fact, *you must load the data of the "1" side first in a 1:M relationship to avoid the possibility of referential integrity errors,* regardless of whether the relationships are weak or strong.

4-1g Weak Entities

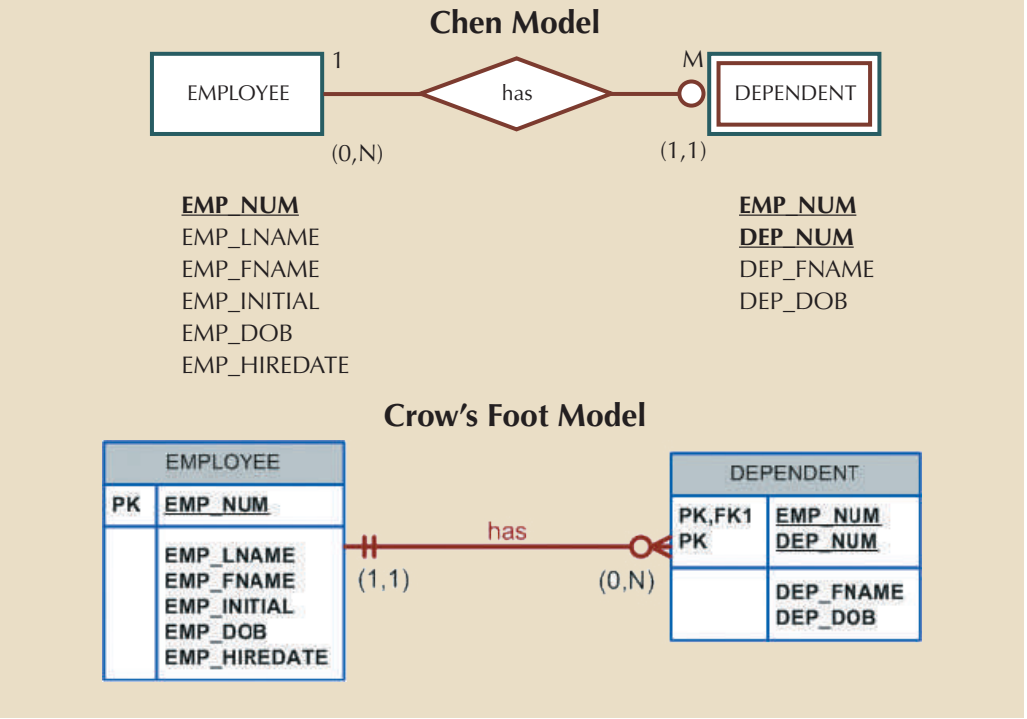
In contrast to the strong or regular entity mentioned in Section 4-1f, a **weak entity** is one that meets two conditions:

1. The entity is existence-dependent; it cannot exist without the entity with which it has a relationship.
2. The entity has a primary key that is partially or totally derived from the parent entity in the relationship.

weak entity
An entity that displays existence dependence and inherits the primary key of its parent entity. For example, a DEPENDENT requires the existence of an EMPLOYEE.

For example, a company insurance policy insures an employee and any dependents. For the purpose of describing an insurance policy, an EMPLOYEE might or might not have a DEPENDENT, but the DEPENDENT must be associated with an EMPLOYEE. Moreover, the DEPENDENT cannot exist without the EMPLOYEE; that is, a person cannot get insurance coverage as a dependent unless the person is a dependent of an employee. DEPENDENT is the weak entity in the relationship "EMPLOYEE has DEPENDENT." This relationship is shown in Figure 4.10.

FIGURE 4.10 A WEAK ENTITY IN AN ERD



Note that the Chen notation in Figure 4.10 identifies the weak entity by using a double-walled entity rectangle. The Crow's Foot notation generated by Visio Professional uses the relationship line and the PK/FK designation to indicate whether the related entity is weak. A strong (identifying) relationship indicates that the related entity is weak. Such a relationship means that both conditions have been met for the weak entity definition—the related entity is existence-dependent, and the PK of the related entity contains a PK component of the parent entity.

Remember that the weak entity inherits part of its primary key from its strong counterpart. For example, at least, part of the DEPENDENT entity's key shown in Figure 4.10 was inherited from the EMPLOYEE entity:

EMPLOYEE (**EMP_NUM**, EMP_LNAME, EMP_FNAME, EMP_INITIAL, EMP_DOB, EMP_HIREDATE)

DEPENDENT (**EMP_NUM, DEP_NUM**, DEP_FNAME, DEP_DOB)

Figure 4.11 illustrates the implementation of the relationship between the weak entity (DEPENDENT) and its parent or strong counterpart (EMPLOYEE). Note that DEPENDENT's primary key is composed of two attributes, EMP_NUM and DEP_NUM, and that EMP_NUM was inherited from EMPLOYEE.

FIGURE 4.11 A WEAK ENTITY IN A STRONG RELATIONSHIP

Table name: EMPLOYEE **Database name: Ch04_ShortCo**

EMP_NUM	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_DOB	EMP_HIREDATE
1001	Callifante	Jeanine	J	12-Mar-64	25-May-97
1002	Smithson	William	K	23-Nov-70	28-May-97
1003	Washington	Herman	H	15-Aug-68	28-May-97
1004	Chen	Lydia	B	23-Mar-74	15-Oct-98
1005	Johnson	Melanie		28-Sep-66	20-Dec-98
1006	Ortega	Jorge	G	12-Jul-79	05-Jan-02
1007	O'Donnell	Peter	D	10-Jun-71	23-Jun-02
1008	Brzenski	Barbara	A	12-Feb-70	01-Nov-03

Table name: DEPENDENT

EMP_NUM	DEP_NUM	DEP_FNAME	DEP_DOB
1001	1	Annelise	05-Dec-97
1001	2	Jorge	30-Sep-02
1003	1	Suzanne	25-Jan-04
1006	1	Carlos	25-May-01
1008	1	Michael	19-Feb-95
1008	2	George	27-Jun-98
1008	3	Katherine	18-Aug-03

Given this scenario, and with the help of this relationship, you can determine that:

Jeanine J. Callifante claims two dependents, Annelise and Jorge.

Keep in mind that the database designer usually determines whether an entity can be described as weak based on the business rules. An examination of Figure 4.8 might cause you to conclude that CLASS is a weak entity to COURSE. After all, it seems clear that a CLASS cannot exist without a COURSE, so there is existence dependence. For example, a student cannot enroll in the Accounting I class ACCT-211, Section 3 (CLASS_CODE 10014), unless there is an ACCT-211 course. However, note that the CLASS table's primary key is CLASS_CODE, which is not derived from the COURSE parent entity. That is, CLASS may be represented by:

CLASS (**CLASS_CODE**, CRS_CODE, CLASS_SECTION, CLASS_TIME, ROOM_CODE, PROF_NUM)

The second weak entity requirement has not been met; therefore, by definition, the CLASS entity in Figure 4.8 may not be classified as weak. On the other hand, if the CLASS entity's primary key had been defined as a composite key composed of the combination CRS_CODE and CLASS_SECTION, CLASS could be represented by:

CLASS (**CRS_CODE, CLASS_SECTION**, CLASS_TIME, ROOM_CODE, PROF_NUM)

In that case, as illustrated in Figure 4.9, the CLASS primary key is partially derived from COURSE because CRS_CODE is the COURSE table's primary key. Given this decision, CLASS is a weak entity by definition. (In Visio Professional Crow's Foot terms, the relationship between COURSE and CLASS is classified as strong, or identifying.) In any case, CLASS is always existence-dependent on COURSE, *whether or not it is defined as weak*.

4-1h Relationship Participation

Participation in an entity relationship is either optional or mandatory. Recall that relationships are bidirectional; that is, they operate in both directions. If COURSE is related to CLASS, then by definition, CLASS is related to COURSE. Because of the

bidirectional nature of relationships, it is necessary to determine the connectivity of the relationship from COURSE to CLASS and the connectivity of the relationship from CLASS to COURSE. Similarly, the specific maximum and minimum cardinalities must be determined in each direction for the relationship. Once again, you must consider the bidirectional nature of the relationship when determining participation.

Optional participation means that one entity occurrence does not *require* a corresponding entity occurrence in a particular relationship. For example, in the "COURSE generates CLASS" relationship, you noted that at least some courses do not generate a class. In other words, an entity occurrence (row) in the COURSE table does not necessarily require the existence of a corresponding entity occurrence in the CLASS table. (Remember that each entity is implemented as a table.) Therefore, the CLASS entity is considered to be *optional* to the COURSE entity. In Crow's Foot notation, an optional relationship between entities is shown by drawing a small circle (O) on the side of the optional entity, as illustrated in Figure 4.9. The existence of an *optional entity* indicates that its minimum cardinality is 0. (The term *optionality* is used to label any condition in which one or more optional relationships exist.)

Note

Remember that the burden of establishing the relationship is always placed on the entity that contains the foreign key. In most cases, that entity is on the "many" side of the relationship.

Mandatory participation means that one entity occurrence *requires* a corresponding entity occurrence in a particular relationship. If no optionality symbol is depicted with the entity, the entity is assumed to exist in a mandatory relationship with the related entity. If the mandatory participation is depicted graphically, it is typically shown as a small hash mark across the relationship line, similar to the Crow's Foot depiction of a connectivity of 1. The existence of a mandatory relationship indicates that the minimum cardinality is at least 1 for the mandatory entity.

optional participation
In ER modeling, a condition in which one entity occurrence does not require a corresponding entity occurrence in a particular relationship.

mandatory participation
A relationship in which one entity occurrence must have a corresponding occurrence in another entity. For example, an EMPLOYEE works in a DIVISION. (A person cannot be an employee without being assigned to a company's division.)

Note

You might be tempted to conclude that relationships are weak when they occur between entities in an optional relationship and that relationships are strong when they occur between entities in a mandatory relationship. However, this conclusion is not warranted. Keep in mind that relationship participation and relationship strength do not describe the same thing. You are likely to encounter a strong relationship when one entity is optional to another. For example, the relationship between EMPLOYEE and DEPENDENT is clearly a strong one, but DEPENDENT is clearly optional to EMPLOYEE. After all, you cannot require employees to have dependents. Also, it is just as possible for a weak relationship to be established when one entity is mandatory to another. The relationship strength depends on how the PK of the related entity is formulated, while the relationship participation depends on how the business rule is written. For example, the business rules "Each part must be supplied by a vendor" and "A part may or may not be supplied by a vendor" create different optionalities for the same entities! Failure to understand this distinction may lead to poor design decisions that cause major problems when table rows are inserted or deleted..

When you create a relationship in Microsoft Visio, the default relationship will be mandatory on the "1" side and optional on the "many" side. Table 4.3 shows the various connectivity and participation combinations that are supported by the Crow's Foot notation. Recall that these combinations are often referred to as cardinality in Crow's Foot notation when specific cardinalities are not used.

TABLE 4.3

CROW'S FOOT SYMBOLS

SYMBOL	CARDINALITY	COMMENT
	(0,N)	Zero or many; the "many" side is optional.
	(1,N)	One or many; the "many" side is mandatory.
	(1,1)	One and only one; the "1" side is mandatory.
	(0,1)	Zero or one; the "1" side is optional.

Because relationship participation is an important component of database design, you should examine a few more scenarios. Suppose that Tiny College employs some professors who conduct research without teaching classes. If you examine the "PROFESSOR teaches CLASS" relationship, it is quite possible for a PROFESSOR not to teach a CLASS. Therefore, CLASS is *optional* to PROFESSOR. On the other hand, a CLASS must be taught by a PROFESSOR. Therefore, PROFESSOR is *mandatory* to CLASS. Note that the ERD model in Figure 4.12 shows the cardinality next to CLASS to be (0,3), indicating that a professor may teach no classes or as many as three classes. Also, each CLASS table row references one and only one PROFESSOR row—assuming each class is taught by one and only one professor—represented by the (1,1) cardinality next to the PROFESSOR table.

FIGURE 4.12 AN OPTIONAL CLASS ENTITY IN THE RELATIONSHIP "PROFESSOR TEACHES CLASS"

It is important that you clearly understand the distinction between mandatory and optional participation in relationships. Otherwise, you might develop designs in which awkward and unnecessary temporary rows (entity instances) must be created just to accommodate the creation of required entities.

It is also important to understand that the semantics of a problem might determine the type of participation in a relationship. For example, suppose that Tiny College offers several courses; each course has several classes. Note again the distinction between *class* and *course* in this discussion: a CLASS constitutes a specific offering (or section) of a COURSE. Typically, courses are listed in the university's course catalog, while classes are listed in the class schedules that students use to register for their classes.

By analyzing the CLASS entity's contribution to the "COURSE generates CLASS" relationship, it is easy to see that a CLASS cannot exist without a COURSE. Therefore, you can conclude that the COURSE entity is *mandatory* in the relationship. However, two scenarios for the CLASS entity may be written, as shown in Figures 4.13 and 4.14.

FIGURE 4.13 CLASS IS OPTIONAL TO COURSE

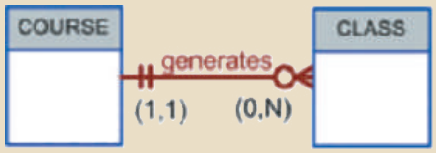
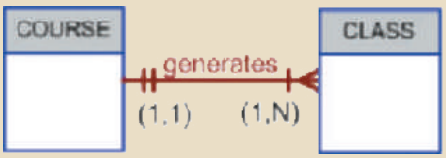


FIGURE 4.14 COURSE AND CLASS IN A MANDATORY RELATIONSHIP



The different scenarios are a function of the problem's semantics; that is, they depend on how the relationship is defined.

1. *CLASS is optional*. It is possible for the department to create the COURSE entity first and then create the CLASS entity after making the teaching assignments. In the real world, such a scenario is very likely; there may be courses for which sections (classes) have not yet been defined. In fact, some courses are taught only once a year and do not generate classes each semester.
2. *CLASS is mandatory*. This condition is created by the constraint imposed by the semantics of the statement "Each COURSE generates one or more CLASSes." In ER terms, each COURSE in the "generates" relationship must have at least one CLASS. Therefore, a CLASS must be created as the COURSE is created to comply with the semantics of the problem.

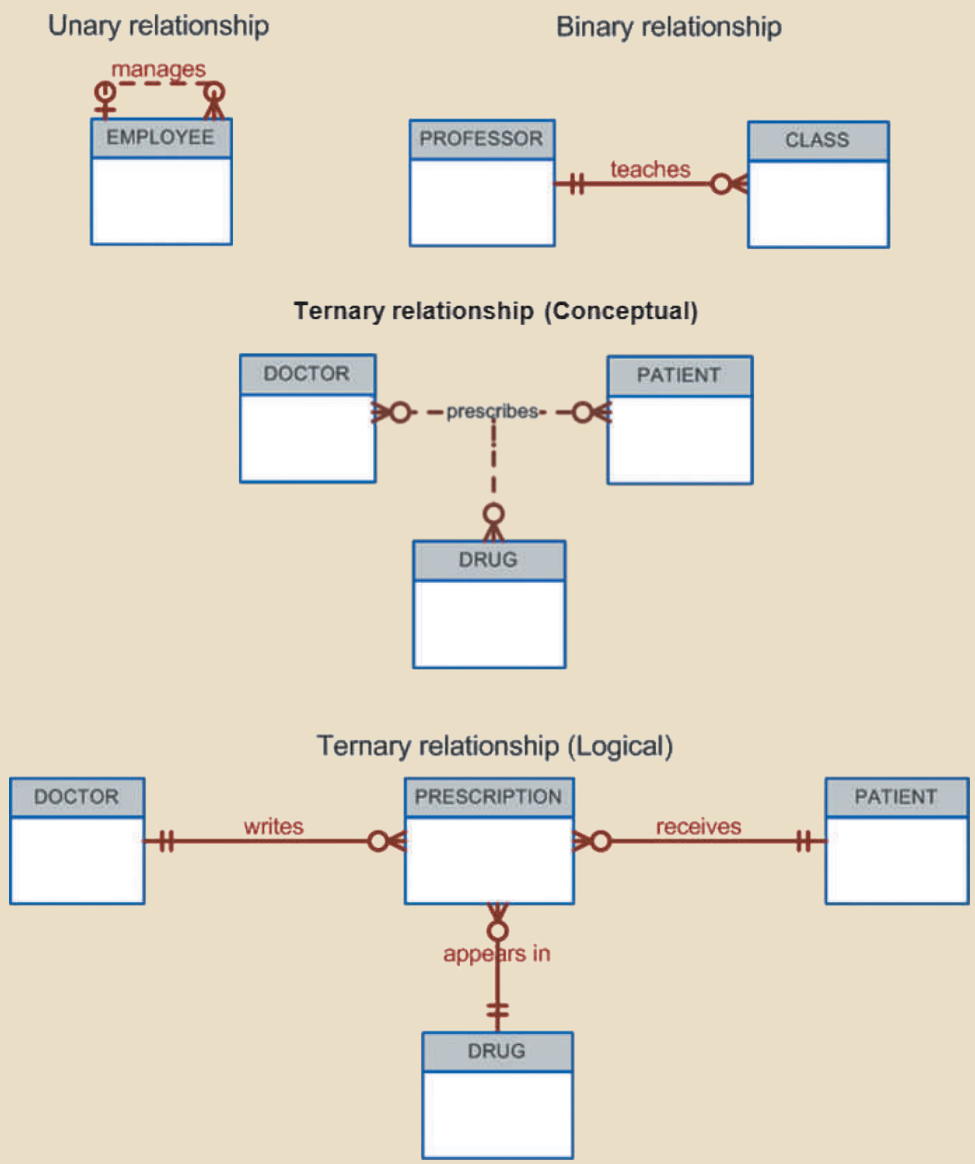
Keep in mind the practical aspects of the scenario presented in Figure 4.14. Given the semantics of this relationship, the system should not accept a course that is not associated with at least one class section. Is such a rigid environment desirable from an operational point of view? For example, when a new COURSE is created, the database first updates the COURSE table, thereby inserting a COURSE entity that does not yet have a CLASS associated with it. Naturally, the apparent problem seems to be solved when CLASS entities are inserted into the corresponding CLASS table. However, because of the mandatory relationship, the system will temporarily violate the business rule constraint. For practical purposes, it would be desirable to classify the CLASS as optional to produce a more flexible design.

Finally, as you examine the scenarios in Figures 4.13 and 4.14, keep in mind the role of the DBMS. To maintain data integrity, the DBMS must ensure that the "many" side (CLASS) is associated with a COURSE through the foreign key rules.

4-1i Relationship Degree

A **relationship degree** indicates the number of entities or participants associated with a relationship. A **unary relationship** exists when an association is maintained within a single entity. A **binary relationship** exists when two entities are associated. A **ternary relationship** exists when three entities are associated. Although higher degrees exist, they are rare and are not specifically named. (For example, an association of four entities is described simply as a *four-degree relationship*.) Figure 4.15 shows these types of relationship degrees.

FIGURE 4.15 THREE TYPES OF RELATIONSHIP DEGREE



relationship degree The number of entities or participants associated with a relationship. A relationship degree can be unary, binary, ternary, or higher.

unary relationship An ER term used to describe an association *within* an entity. For example, an EMPLOYEE might manage another EMPLOYEE.

binary relationship An ER term for an association (relationship) between two entities. For example, PROFESSOR teaches CLASS.

ternary relationship An ER term used to describe an association (relationship) between three entities. For example, a DOCTOR prescribes a DRUG for a PATIENT.

Unary Relationships

In the case of the unary relationship shown in Figure 4.15, an employee within the EMPLOYEE entity is the manager for one or more employees within that entity. In this case, the existence of the "manages" relationship means that EMPLOYEE requires another EMPLOYEE to be the manager—that is, EMPLOYEE has a relationship with itself. Such a relationship is known as a **recursive relationship**. The various cases of recursive relationships are explained in Section 4-1j.

recursive relationship
A relationship found within a single entity type. For example, an EMPLOYEE is married to an EMPLOYEE or a PART is a component of another PART.

Binary Relationships

A binary relationship exists when two entities are associated in a relationship. Binary relationships are the most common type of relationship. In fact, to simplify the conceptual design, most higher-order (ternary and higher) relationships are decomposed into appropriate equivalent binary relationships whenever possible. In Figure 4.15, "a PROFESSOR teaches one or more CLASSes" represents a binary relationship.

Ternary and Higher-Order Relationships

Although most relationships are binary, the use of ternary and higher-order relationships does allow the designer some latitude regarding the semantics of a problem. A ternary relationship implies an association among three different entities. For example, in Figure 4.16, note the relationships and their consequences, which are represented by the following business rules:

- A DOCTOR writes one or more PRESCRIPTIONs.
- A PATIENT may receive one or more PRESCRIPTIONs.
- A DRUG may appear in one or more PRESCRIPTIONs. (To simplify this example, assume that the business rule states that each prescription contains only one drug. In short, if a doctor prescribes more than one drug, a separate prescription must be written for each drug.)

FIGURE 4.16 THE IMPLEMENTATION OF A TERNARY RELATIONSHIP

Database name: Ch04_Clinic

Table name: DRUG

DRUG_CODE	DRUG_NAME	DRUG_PRICE
AF15	Afgapan-15	25.00
AF25	Afgapan-25	35.00
DRO	Droalene Chloride	111.89
DRZ	Druzacholiar Cryptolene	18.99
KO15	Kollabar Oxythexalene	65.75
OLE	Oleander-Drizapan	123.95
TRYP	Tryptoliac Heptadimetric	79.45

Table name: PATIENT

PAT_NUM	PAT_TITLE	PAT_LNAME	PAT_FNAME	PAT_INITIAL	PAT_DOB	PAT_AREACODE	PAT_PHONE
100	Mr.	Kolmycz	George	D	15-Jun-1942	615	324-5456
101	Ms.	Lewis	Rhonda	G	19-Mar-2005	615	324-4472
102	Mr.	Vandam	Rhett		14-Nov-1958	901	675-8993
103	Ms.	Jones	Anne	M	16-Oct-1974	615	898-3456
104	Mr.	Lange	John	P	08-Nov-1971	901	504-4430
105	Mr.	Williams	Robert	D	14-Mar-1975	615	890-3220
106	Mrs.	Smith	Jeanine	K	12-Feb-2003	615	324-7883
107	Mr.	Diante	Jorge	D	21-Aug-1974	615	890-4567
108	Mr.	Wesenbach	Paul	R	14-Feb-1966	615	897-4358
109	Mr.	Smith	George	K	18-Jun-1961	901	504-3339
110	Mrs.	Genkazi	Leighla	W	19-May-1970	901	569-8093
111	Mr.	Washington	Rupert	E	03-Jan-1966	615	890-4925
112	Mr.	Johnson	Edward	E	14-May-1961	615	898-4387
113	Ms.	Smythe	Melanie	P	15-Sep-1970	615	324-9006
114	Ms.	Brandon	Marie	G	02-Nov-1932	901	882-8845
115	Mrs.	Saranda	Hermine	R	25-Jul-1972	615	324-5505
116	Mr.	Smith	George	A	08-Nov-1965	615	890-2984

Table name: DOCTOR

DOC_ID	DOC_LNAME	DOC_FNAME	DOC_INITIAL	DOC_SPECIALTY
29827	Sanchez	Julio	J	Dermatology
32445	Jorgensen	Annelise	G	Neurology
33456	Korenski	Anatoly	A	Urology
33989	LeGrande	George		Pediatrics
34409	Washington	Dennis	F	Orthopaedics
36221	McPherson	Katye	H	Dermatology
36712	Dreifag	Herman	G	Psychiatry
38995	Minh	Tran		Neurology
40004	Chin	Ming	D	Orthopaedics
40028	Feinstein	Denise	L	Gynecology

Table name: PRESCRIPTION

DOC_ID	PAT_NUM	DRUG_CODE	PRES_DOSAGE	PRES_DATE
32445	102	DRZ	2 tablets every four hours -- 50 tablets total	12-Nov-18
32445	113	OLE	1 teaspoon with each meal -- 250 ml total	14-Nov-18
34409	101	KO15	1 tablet every six hours -- 30 tablets total	14-Nov-18
36221	109	DRO	2 tablets with every meal -- 60 tablets total	14-Nov-18
38995	107	KO15	1 tablet every six hours -- 30 tablets total	14-Nov-18

As you examine the table contents in Figure 4.16, note that it is possible to track all transactions. For instance, you can tell that the first prescription was written by doctor 32445 for patient 102, using the drug DRZ.

4-1j Recursive Relationships

As you just learned, a *recursive relationship* is one in which a relationship can exist between occurrences of the same entity set. (Naturally, such a condition is found within a unary relationship.) For example, a 1:M unary relationship can be expressed by "an EMPLOYEE may manage many EMPLOYEEs, and each EMPLOYEE is managed by one EMPLOYEE." Also, as long as polygamy is not legal, a 1:1 unary relationship may be expressed by "an EMPLOYEE may be married to one and only one other EMPLOYEE." Finally, the M:N unary relationship may be expressed by "a COURSE may be a prerequisite to many other COURSEs, and each COURSE may have many other COURSEs as prerequisites." Those relationships are shown in Figure 4.17.

FIGURE 4.17 AN ER REPRESENTATION OF RECURSIVE RELATIONSHIPS

The 1:1 relationship shown in Figure 4.17 can be implemented in the single table shown in Figure 4.18. Note that you can determine that James Ramirez is married to Louise Ramirez, who is married to James Ramirez. Also, Anne Jones is married to Anton Shapiro, who is married to Anne Jones.

FIGURE 4.18 THE 1:1 RECURSIVE RELATIONSHIP "EMPLOYEE IS MARRIED TO EMPLOYEE"

Database name: Ch04_PartCo
Table name: EMPLOYEE_V1

EMP_NUM	EMP_LNAME	EMP_FNAME	EMP_SPOUSE
345	Ramirez	James	347
346	Jones	Anne	349
347	Ramirez	Louise	345
348	Delaney	Robert	
349	Shapiro	Anton	346

Unary relationships are common in manufacturing industries. For example, Figure 4.19 illustrates that a rotor assembly (C-130) is composed of many parts, but each part is used to create only one rotor assembly. Figure 4.19 indicates that a rotor assembly is composed of four 2.5-cm washers, two cotter pins, one 2.5-cm steel shank, four 10.25-cm rotor blades, and two 2.5-cm hex nuts. The relationship implemented in Figure 4.19 thus enables you to track each part within each rotor assembly.

FIGURE 4.19 ANOTHER UNARY RELATIONSHIP: "PART CONTAINS PART"

Table name: PART_V1 **Database name: Ch04_PartCo**

PART_CODE	PART_DESCRIPTION	PART_IN_STOCK	PART_UNITS_NEEDED	PART_OF_PART
AA21-6	2.5 cm. washer, 1.0 mm. rim	432	4	C-130
AB-121	Cotter pin, copper	1034	2	C-130
C-130	Rotor assembly	36		
E129	2.5 cm. steel shank	128	1	C-130
X10	10.25 cm. rotor blade	345	4	C-130
X34AW	2.5 cm. hex nut	879	2	C-130

If a part can be used to assemble several different kinds of other parts and is itself composed of many parts, two tables are required to implement the "PART contains PART" relationship. Figure 4.20 illustrates such an environment. Parts tracking is increasingly important as managers become more aware of the legal ramifications of producing more complex output. In many industries, especially those involving aviation, full parts tracking is required by law.

FIGURE 4.20 THE IMPLEMENTATION OF THE M:N RECURSIVE RELATIONSHIP "PART CONTAINS PART"

Table name: COMPONENT **Database name: Ch04_PartCo**

COMP_CODE	PART_CODE	COMP_PARTS_NEEDED
C-130	AA21-6	4
C-130	AB-121	2
C-130	E129	1
C-131A2	E129	1
C-130	X10	4
C-131A2	X10	1
C-130	X34AW	2
C-131A2	X34AW	2

Table name: PART

PART_CODE	PART_DESCRIPTION	PART_IN_STOCK
AA21-6	2.5 cm. washer, 1.0 mm. rim	432
AB-121	Cotter pin, copper	1034
C-130	Rotor assembly	36
E129	2.5 cm. steel shank	128
X10	10.25 cm. rotor blade	345
X34AW	2.5 cm. hex nut	879

The M:N recursive relationship might be more familiar in a school environment. For instance, note how the M:N "COURSE requires COURSE" relationship illustrated in Figure 4.17 is implemented in Figure 4.21. In this example, MATH-243 is a prerequisite to QM-261 and QM-362, while both MATH-243 and QM-261 are prerequisites to QM-362.

Finally, the 1:M recursive relationship "EMPLOYEE manages EMPLOYEE," shown in Figure 4.17, is implemented in Figure 4.22.

One common pitfall when working with unary relationships is to confuse participation with referential integrity. In theory, participation and referential integrity are very different concepts and are normally easy to distinguish in binary relationships. In practical terms, conversely, participation and referential integrity are very similar because they are both implemented through constraints on the same set of attributes. This similarity

FIGURE 4.21 IMPLEMENTATION OF THE M:N RECURSIVE RELATIONSHIP "COURSE REQUIRES COURSE"

Table name: COURSE **Database name: Ch04_TinyCollege**

CRS_CODE	DEPT_CODE	CRS_DESCRIPTION	CRS_CREDIT
ACCT-211	ACCT	Accounting I	3
ACCT-212	ACCT	Accounting II	3
CIS-220	CIS	Intro. to Microcomputing	3
CIS-420	CIS	Database Design and Implementation	4
MATH-243	MATH	Mathematics for Managers	3
QM-261	CIS	Intro. to Statistics	3
QM-362	CIS	Statistical Applications	4

Table name: PREREQ

CRS_CODE	PRE_TAKE
CIS-420	CIS-220
QM-261	MATH-243
QM-362	MATH-243
QM-362	QM-261

FIGURE 4.22 IMPLEMENTATION OF THE 1:M RECURSIVE RELATIONSHIP "EMPLOYEE MANAGES EMPLOYEE"

Database name: Ch04_PartCo

Table name: EMPLOYEE_V2

EMP_CODE	EMP_LNAME	EMP_MANAGER
101	Waddell	102
102	Orincona	
103	Jones	102
104	Reballoh	102
105	Robertson	102
106	Deltona	102

often leads to confusion when the concepts are applied within the limited structure of a unary relationship. Consider the unary 1:1 spousal relationship between employees, which is described in Figure 4.18. Participation, as described previously, is bidirectional, meaning that it must be addressed in both directions along the relationship. Participation in Figure 4.18 addresses the following questions:

- Must every employee have a spouse who is an employee?
- Must every employee be a spouse to another employee?

For the data shown in Figure 4.18, the correct answer to both questions is "No." It is possible to be an employee and not have another employee as a spouse. Also, it is possible to be an employee and not be the spouse of another employee.

Referential integrity deals with the correspondence of values in the foreign key with values in the related primary key. Referential integrity is not bidirectional, and therefore answers only one question:

- Must every employee spouse be a valid employee?

For the data shown in Figure 4.18, the correct answer is "Yes." Another way to frame this question is to consider whether every value provided for the EMP_SPOUSE attribute must match some value in the EMP_NUM attribute.

In practical terms, both participation and referential integrity involve the values used as primary keys and foreign keys to implement the relationship. Referential integrity

requires that the values in the foreign key correspond to values in the primary key. In one direction, participation considers whether the foreign key can contain a null. In Figure 4.18, for example, employee Robert Delaney is not required to have a value in EMP_SPOUSE. In the other direction, participation considers whether every value in the primary key must appear as a value in the foreign key. In Figure 4.18, for example, employee Robert Delaney's value for EMP_NUM (348) is not required to appear as a value in EMP_SPOUSE for any other employee.

4-1k Associative (Composite) Entities

M:N relationships are a valid construct at the conceptual level, and therefore are found frequently during the ER modeling process. However, implementing the M:N relationship, particularly in the relational model, requires the use of an additional entity, as you learned in Chapter 3. The ER model uses the associative entity to represent an M:N relationship between two or more entities. This associative entity, also called a *composite* or *bridge entity*, is in a 1:M relationship with the parent entities and is composed of the primary key attributes of each parent entity. Furthermore, the associative entity can have additional attributes of its own, as shown by the ENROLL associative entity in Figure 4.23. When using the Crow's Foot notation, the associative entity is identified as a strong (identifying) relationship, as indicated by the solid relationship lines between the parents and the associative entity.

FIGURE 4.23 CONVERTING THE M:N RELATIONSHIP INTO TWO 1:M RELATIONSHIPS

Table name: STUDENT **Database name: Ch04_CollegeTry**

STU_NUM	STU_LNAME
321452	Bowser
324257	Smithson

Table name: ENROLL

CLASS_CODE	STU_NUM	ENROLL_GRADE
10014	321452	C
10014	324257	B
10018	321452	A
10018	324257	B
10021	321452	C
10021	324257	C

Table name: CLASS

CLASS_CODE	CRS_CODE	CLASS_SECTION	CLASS_TIME	ROOM_CODE	PROF_NUM
10014	ACCT-211	3	TTh 2:30-3:45 p.m.	BUS252	342
10018	CIS-220	2	MWF 9:00-9:50 a.m.	KLR211	114
10021	QM-261	1	MWF 8:00-8:50 a.m.	KLR200	114

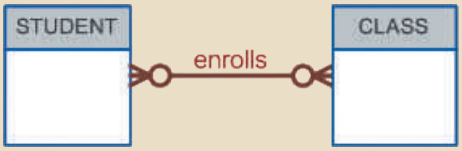
Note that the composite ENROLL entity in Figure 4.23 is existence-dependent on the other two entities; the composition of the ENROLL entity is based on the primary keys of the entities that are connected by the composite entity. The composite entity may also contain additional attributes that play no role in the connective process. For example, although the entity must be composed of at least the STUDENT and CLASS primary keys, it may also include such additional attributes as grades, absences, and other data uniquely identified by the student's performance in a specific class.

Finally, keep in mind that the ENROLL table's key (CLASS_CODE and STU_NUM) is composed entirely of the primary keys of the CLASS and STUDENT tables. Therefore, no null entries are possible in the ENROLL table's key attributes.

Implementing the small database shown in Figure 4.23 requires that you define the relationships clearly. Specifically, you must know the "1" and the "M" sides of each relationship, and you must know whether the relationships are mandatory or optional. For example, note the following points:

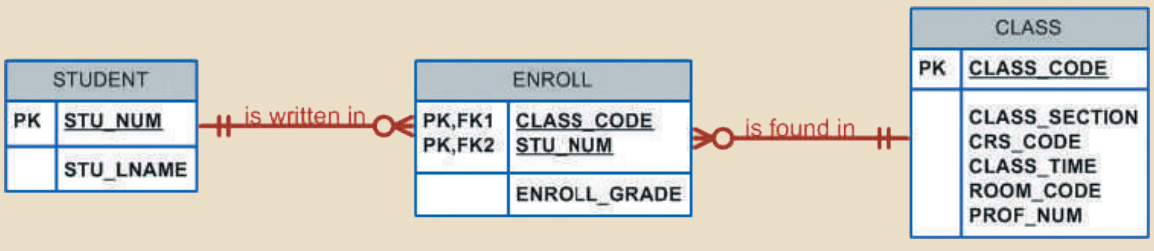
- A class may exist (at least at the start of registration) even though it contains no students. Therefore, in Figure 4.24, an optional symbol should appear on the STUDENT side of the M:N relationship between STUDENT and CLASS.

FIGURE 4.24 THE M:N RELATIONSHIP BETWEEN STUDENT AND CLASS



- You might argue that to be classified as a STUDENT, a person must be enrolled in at least one CLASS. Therefore, CLASS is mandatory to STUDENT from a purely conceptual point of view. However, when a student is admitted to college, that student has not yet signed up for any classes. Therefore, *at least initially*, CLASS is optional to STUDENT. Note that the practical considerations in the data environment help dictate the use of optionalities. If CLASS is *not* optional to STUDENT from a database point of view, a class assignment must be made when the student is admitted. However, that's *not* how the process actually works, and the database design must reflect this. In short, the optionality reflects practice.
- Because the M:N relationship between STUDENT and CLASS is decomposed into two 1:M relationships through ENROLL, the optionalities must be transferred to ENROLL. (See Figure 4.25.) In other words, it now becomes possible for a class not to occur in ENROLL if no student has signed up for that class. Because a class need not occur in ENROLL, the ENROLL entity becomes optional to CLASS. Also, because the ENROLL entity is created before any students have signed up for a class, the ENROLL entity is also optional to STUDENT, at least initially.

FIGURE 4.25 A COMPOSITE ENTITY IN AN ERD



- As students begin to sign up for their classes, they will be entered into the ENROLL entity. Naturally, if a student takes more than one class, that student will occur more than once in ENROLL. For example, note that in the ENROLL table in Figure 4.23, STU_NUM = 321452 occurs three times. On the other hand, each student occurs only once in the STUDENT entity. (Note that the STUDENT table in Figure 4.23 has only one STU_NUM = 321452 entry.) Therefore, in Figure 4.25, the relationship between STUDENT and ENROLL is shown to be 1:M, with the "M" on the ENROLL side.

- As you can see in Figure 4.23, a class can occur more than once in the ENROLL table. For example, CLASS_CODE = 10014 occurs twice. However, CLASS_CODE = 10014 occurs only once in the CLASS table to reflect that the relationship between CLASS and ENROLL is 1:M. Note that in Figure 4.25, the "M" is located on the ENROLL side, while the "1" is located on the CLASS side.

4-2 Developing an ER Diagram

The process of database design is iterative rather than a linear or sequential process. The verb *iterate* means "to do again or repeatedly." Thus, an **iterative process** is based on repetition of processes and procedures. Building an ERD usually involves the following activities:

- Create a detailed narrative of the organization's description of operations.
- Identify the business rules based on the description of operations.
- Identify the main entities and relationships from the business rules.
- Develop the initial ERD.
- Identify the attributes and primary keys that adequately describe the entities.
- Revise and review the ERD.

During the review process, additional objects, attributes, and relationships probably will be uncovered. Therefore, the basic ERM will be modified to incorporate the newly discovered ER components. Subsequently, another round of reviews might yield additional components or clarification of the existing diagram. The process is repeated until the end users and designers agree that the ERD is a fair representation of the organization's activities and functions.

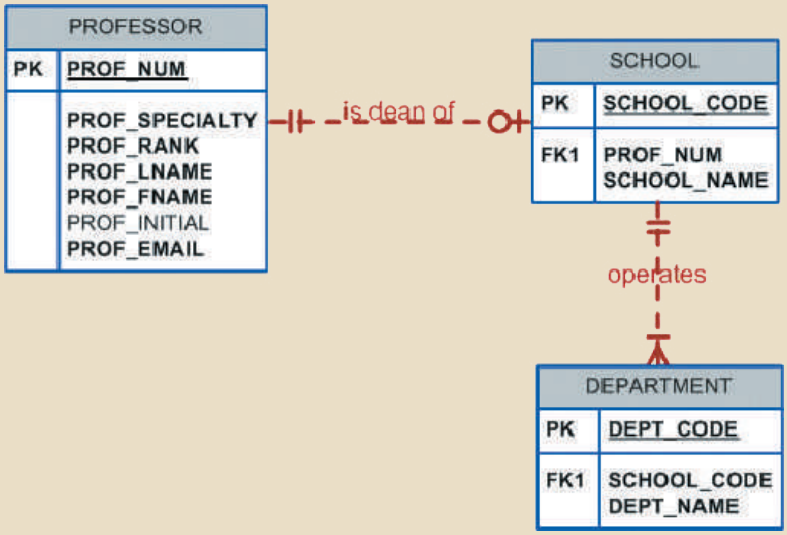
During the design process, the database designer does not depend simply on interviews to help define entities, attributes, and relationships. A surprising amount of information can be gathered by examining the business forms and reports that an organization uses in its daily operations.

To illustrate the use of the iterative process that ultimately yields a workable ERD, start with an initial interview with the Tiny College administrators. The interview process yields the following business rules:

1. Tiny College (TC) is divided into several schools: business, arts and sciences, education, and applied sciences. Each school is administered by a dean who is a professor. Each professor can be the dean of only one school, and a professor is not required to be the dean of any school. Therefore, a 1:1 relationship exists between PROFESSOR and SCHOOL. Note that the cardinality can be expressed by writing (1,1) next to the entity PROFESSOR and (0,1) next to the entity SCHOOL.
2. Each school comprises several departments. For example, the school of business has an accounting department, a management/marketing department, an economics/finance department, and a computer information systems department. Note again the cardinality rules: The smallest number of departments operated by a school is one, and the largest number of departments is indeterminate (N). On the other hand, each department belongs to only a single school; thus, the cardinality is expressed by (1,1). That is, the minimum number of schools to which a department belongs is one, as is the maximum number. Figure 4.26 illustrates these first two business rules.
3. Each department may offer courses. For example, the management/marketing department offers courses such as Introduction to Management, Principles of Marketing, and Production Management. The ERD segment for this condition is

iterative process
A process based on repetition of steps and procedures.

FIGURE 4.26 THE FIRST TINY COLLEGE ERD SEGMENT



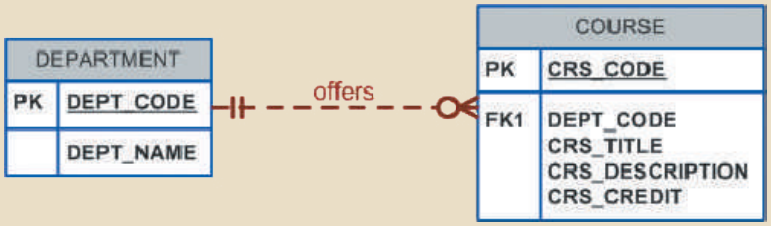
Note

It is again appropriate to evaluate the reason for maintaining the 1:1 relationship between PROFESSOR and SCHOOL in the "PROFESSOR is dean of SCHOOL" relationship. It is worth repeating that the existence of 1:1 relationships often indicates a misidentification of attributes as entities. In this case, the 1:1 relationship could easily be eliminated by storing the dean's attributes in the SCHOOL entity. This solution would also make it easier to answer the queries "Who is the dean?" and "What are the dean's credentials?" The downside of this solution is that it requires the duplication of data that is already stored in the PROFESSOR table, thus setting the stage for anomalies. However, because each school is run by a single dean, the problem of data duplication is rather minor. The selection of one approach over another often depends on information requirements, transaction speed, and the database designer's professional judgment. In short, do not use 1:1 relationships lightly, and make sure that each 1:1 relationship within the database design is defensible.

shown in Figure 4.27. Note that this relationship is based on the way Tiny College operates. For example, if Tiny College had some departments that were classified as "research only," they would not offer courses; therefore, the COURSE entity would be optional to the DEPARTMENT entity.

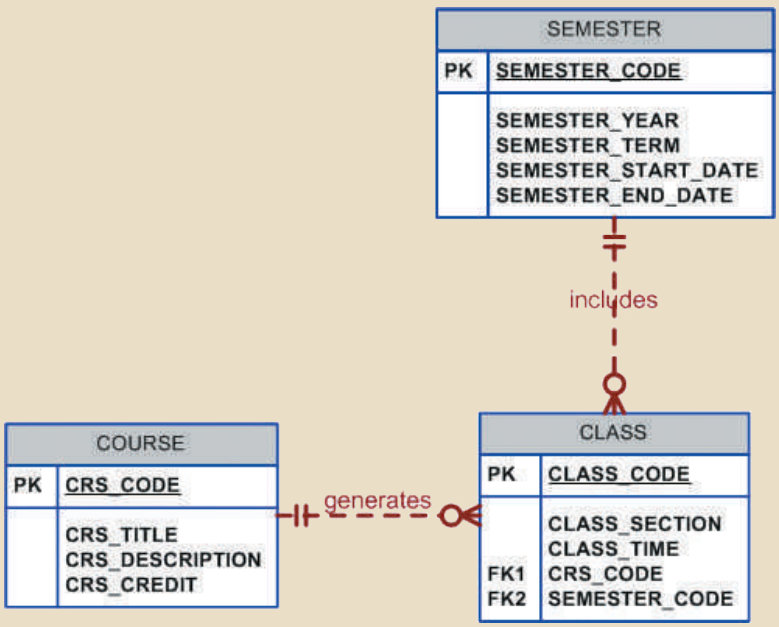
4. The relationship between COURSE and CLASS was illustrated in Figure 4.9. Nevertheless, it is worth repeating that a CLASS is a section of a COURSE. That is, a department may offer several sections (classes) of the same database course. Each of those classes is taught by a professor at a given time in a given place. In short, a 1:M relationship exists between COURSE and CLASS. Additionally, each class is offered during a given semester. SEMESTER defines the year and the term that the class will be offered. Note that this is different from the date when the student actually enrolls in a class. For example, students are able to enroll in summer and fall term classes near the end of the spring term. It is possible that the Tiny College calendar is set with semester beginning and ending dates prior to the creation of the semester class

FIGURE 4.27 THE SECOND TINY COLLEGE ERD SEGMENT



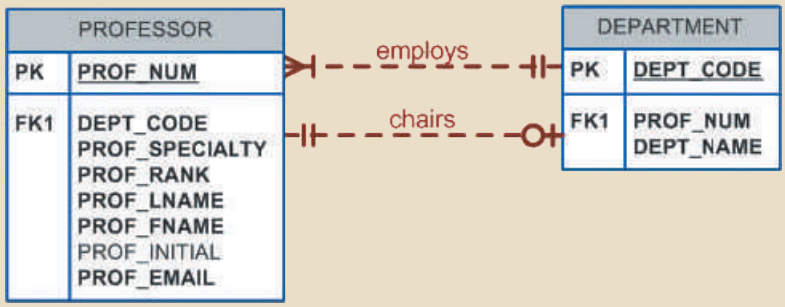
schedule so CLASS is optional to SEMESTER. This design will also help for reporting purposes, for example, you could answer questions such as: what classes were offered X semester? Or, what classes did student Y take on semester X? Because a course may exist in Tiny College's course catalog even when it is not offered as a class in a given semester, CLASS is optional to COURSE. Therefore, the relationships between SEMESTER, COURSE, and CLASS look like Figure 4.28.

FIGURE 4.28 THE THIRD TINY COLLEGE ERD SEGMENT



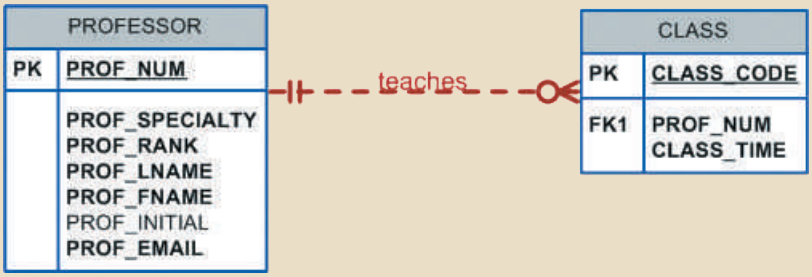
5. Each department should have one or more professors assigned to it. One and only one of those professors chairs the department, and no professor is required to accept the chair position. Therefore, DEPARTMENT is optional to PROFESSOR in the "chairs" relationship. Those relationships are summarized in the ER segment shown in Figure 4.29.

FIGURE 4.29 THE FOURTH TINY COLLEGE ERD SEGMENT



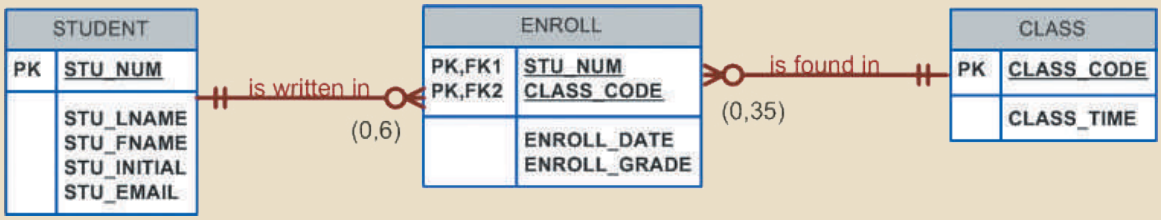
6. Each professor may teach up to four classes; each class is a section of a course. A professor may also be on a research contract and teach no classes at all. The ERD segment in Figure 4.30 depicts those conditions.

FIGURE 4.30 THE FIFTH TINY COLLEGE ERD SEGMENT



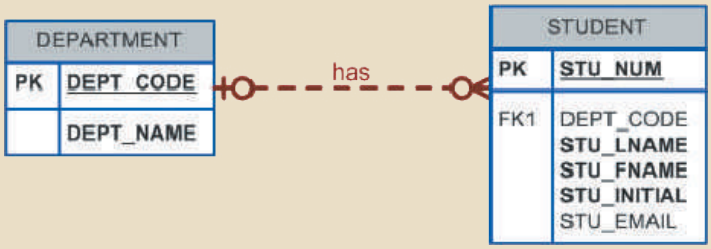
7. A student may enroll in several classes but take each class only once during any given enrollment period. For example, during the current enrollment period, a student may decide to take five classes—Statistics, Accounting, English, Database, and History—but that student would not be enrolled in the same Statistics class five times during the enrollment period! Each student may enroll in up to six classes, and each class may have up to 35 students, thus creating an M:N relationship between STUDENT and CLASS. Because a CLASS can initially exist at the start of the enrollment period even though no students have enrolled in it, STUDENT is optional to CLASS in the M:N relationship. This M:N relationship must be divided into two 1:M relationships through the use of the ENROLL entity, shown in the ERD segment in Figure 4.31. However, note that the optional symbol is shown next to ENROLL. If a class exists but has no students enrolled in it, that class does not occur in the ENROLL table. Note also that the ENROLL entity is weak: it is existence-dependent, and its (composite) PK is composed of the PKs of the STUDENT and CLASS entities. You can add the cardinalities (0,6) and (0,35) next to the ENROLL entity to reflect the business rule constraints, as shown in Figure 4.31. (Visio Professional does not automatically generate such cardinalities, but you can use a text box to accomplish that task.)

FIGURE 4.31 THE SIXTH TINY COLLEGE ERD SEGMENT



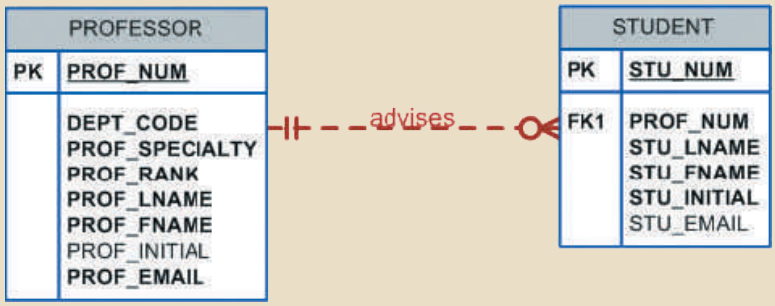
8. Each department has several (or many) students whose major is offered by that department. However, each student has only a single major and is therefore associated with a single department. (See Figure 4.32.) However, in the Tiny College environment, it is possible—at least for a while—for a student not to declare a major field of study. Such a student would not be associated with a department; therefore, DEPARTMENT is optional to STUDENT. It is worth repeating that the relationships between entities and the entities themselves reflect the organization's operating environment. That is, the business rules define the ERD components.

FIGURE 4.32 THE SEVENTH TINY COLLEGE ERD SEGMENT



9. Each student has an advisor in his or her department; each advisor counsels several students. An advisor is also a professor, but not all professors advise students. Therefore, STUDENT is optional to PROFESSOR in the "PROFESSOR advises STUDENT" relationship. (See Figure 4.33.)

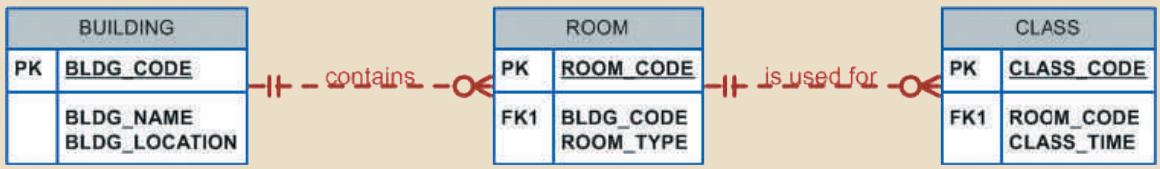
FIGURE 4.33 THE EIGHTH TINY COLLEGE ERD SEGMENT



10. As you can see in Figure 4.34, the CLASS entity contains a ROOM_CODE attribute. Given the naming conventions, it is clear that ROOM_CODE is an FK to another entity. Clearly, because a class is taught in a room, it is reasonable to assume that

the ROOM_CODE in CLASS is the FK to an entity named ROOM. In turn, each room is located in a building. So, the last Tiny College ERD is created by observing that a BUILDING can contain many ROOMs, but each ROOM is found in a single BUILDING. In this ERD segment, it is clear that some buildings do not contain (class) rooms. For example, a storage building might not contain any named rooms at all.

FIGURE 4.34 THE NINTH TINY COLLEGE ERD SEGMENT



Using the preceding summary, you can identify the following entities:

PROFESSOR	SCHOOL	DEPARTMENT
COURSE	CLASS	SEMESTER
STUDENT	BUILDING	ROOM

ENROLL (the associative entity between STUDENT and CLASS)

Once you have discovered the relevant entities, you can define the initial set of relationships among them. Next, you describe the entity attributes. Identifying the attributes of the entities helps you to better understand the relationships among entities. Table 4.4 summarizes the ERM's components, and names the entities and their relations.

TABLE 4.4

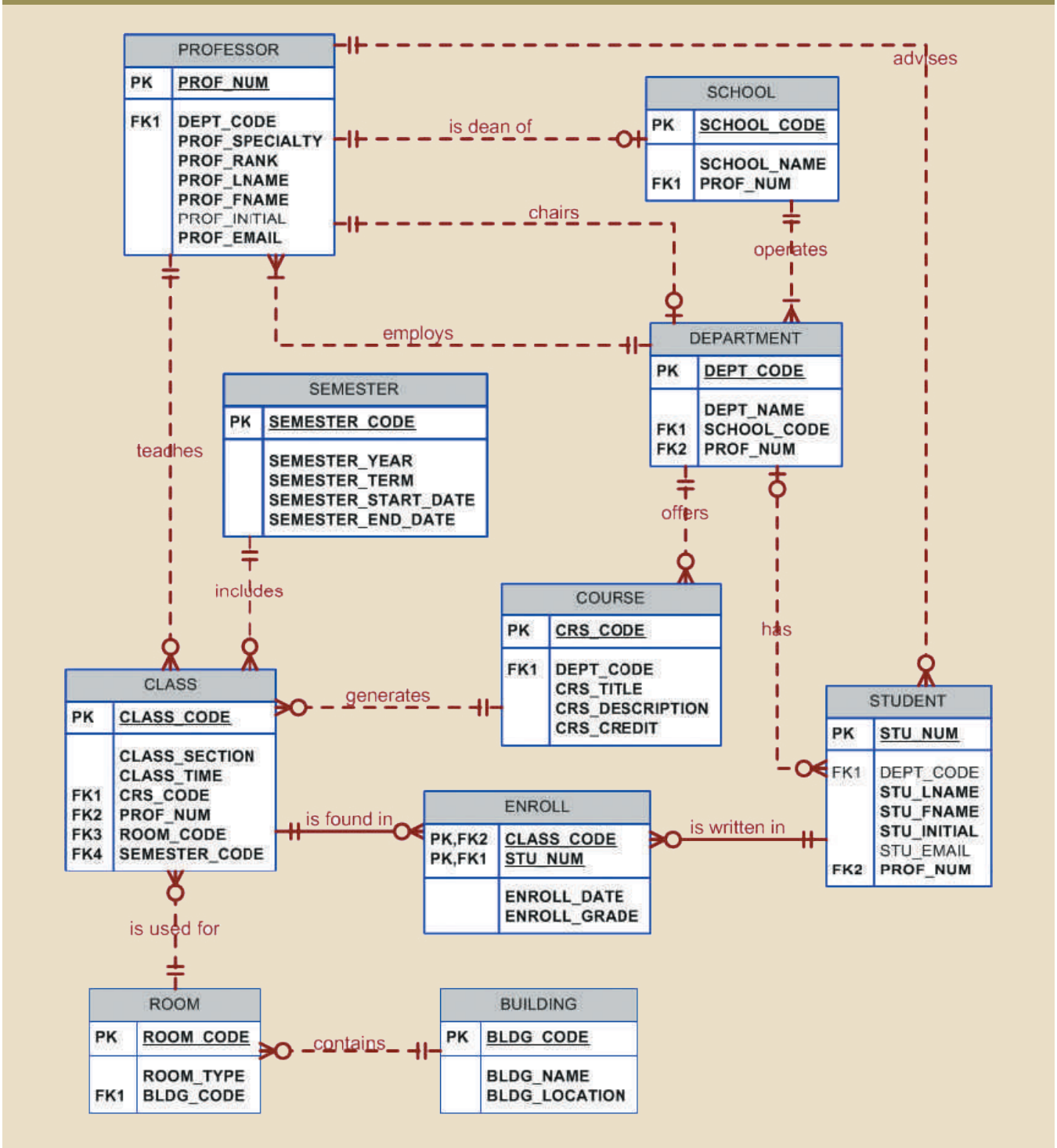
COMPONENTS OF THE ERM

ENTITY	RELATIONSHIP	CONNECTIVITY	ENTITY
SCHOOL	operates	1:M	DEPARTMENT
DEPARTMENT	has	1:M	STUDENT
DEPARTMENT	employs	1:M	PROFESSOR
DEPARTMENT	offers	1:M	COURSE
COURSE	generates	1:M	CLASS
SEMESTER	includes	1:M	CLASS
PROFESSOR	is dean of	1:1	SCHOOL
PROFESSOR	chairs	1:1	DEPARTMENT
PROFESSOR	teaches	1:M	CLASS
PROFESSOR	advises	1:M	STUDENT
STUDENT	enrolls in	M:N	CLASS
BUILDING	contains	1:M	ROOM
ROOM	is used for	1:M	CLASS

Note: ENROLL is the composite entity that implements the M:N relationship "STUDENT enrolls in CLASS."

You must also define the connectivity and cardinality for the just-discovered relations based on the business rules. However, to avoid crowding the diagram, the cardinalities are not shown. Figure 4.35 shows the Crow's Foot ERD for Tiny College. Note that this is an implementation-ready model, so it shows the ENROLL composite entity.

FIGURE 4.35 THE COMPLETED TINY COLLEGE ERD



Although we focus on Crow's Foot notation to develop our diagram, as mentioned at the beginning of this chapter, UML notation is also popular for conceptual and implementation modeling. Figure 4.36 shows the conceptual UML class diagram for Tiny College. Note that this class diagram depicts the M:N relationship between STUDENT and CLASS. Figure 4.37 shows the implementation-ready UML class diagram for Tiny College (note that the ENROLL composite entity is shown in this class diagram). If you are a good observer, you will also notice that the UML class diagrams in Figures 4.36 and 4.37 show the entity and attribute names but do not identify the primary key attributes. The reason goes back to UML's roots. UML class diagrams are an object-oriented modeling language, and therefore do not support the notion of "primary or foreign keys" found mainly in the relational world. Rather, in the object-oriented world, objects inherit a unique object identifier at creation time. For more information, see Appendix G, Object-Oriented Databases.

FIGURE 4.36 THE CONCEPTUAL UML CLASS DIAGRAM FOR TINY COLLEGE

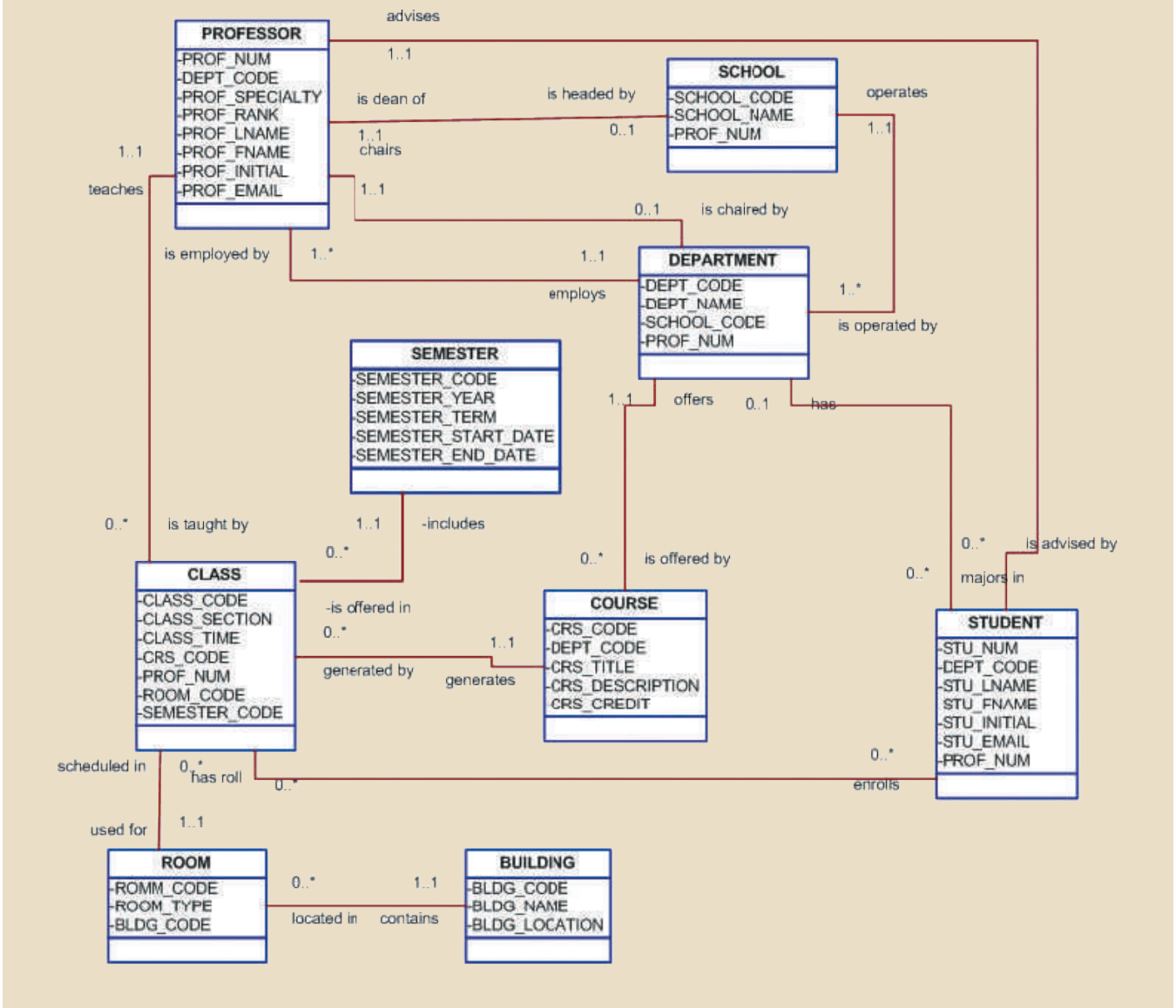
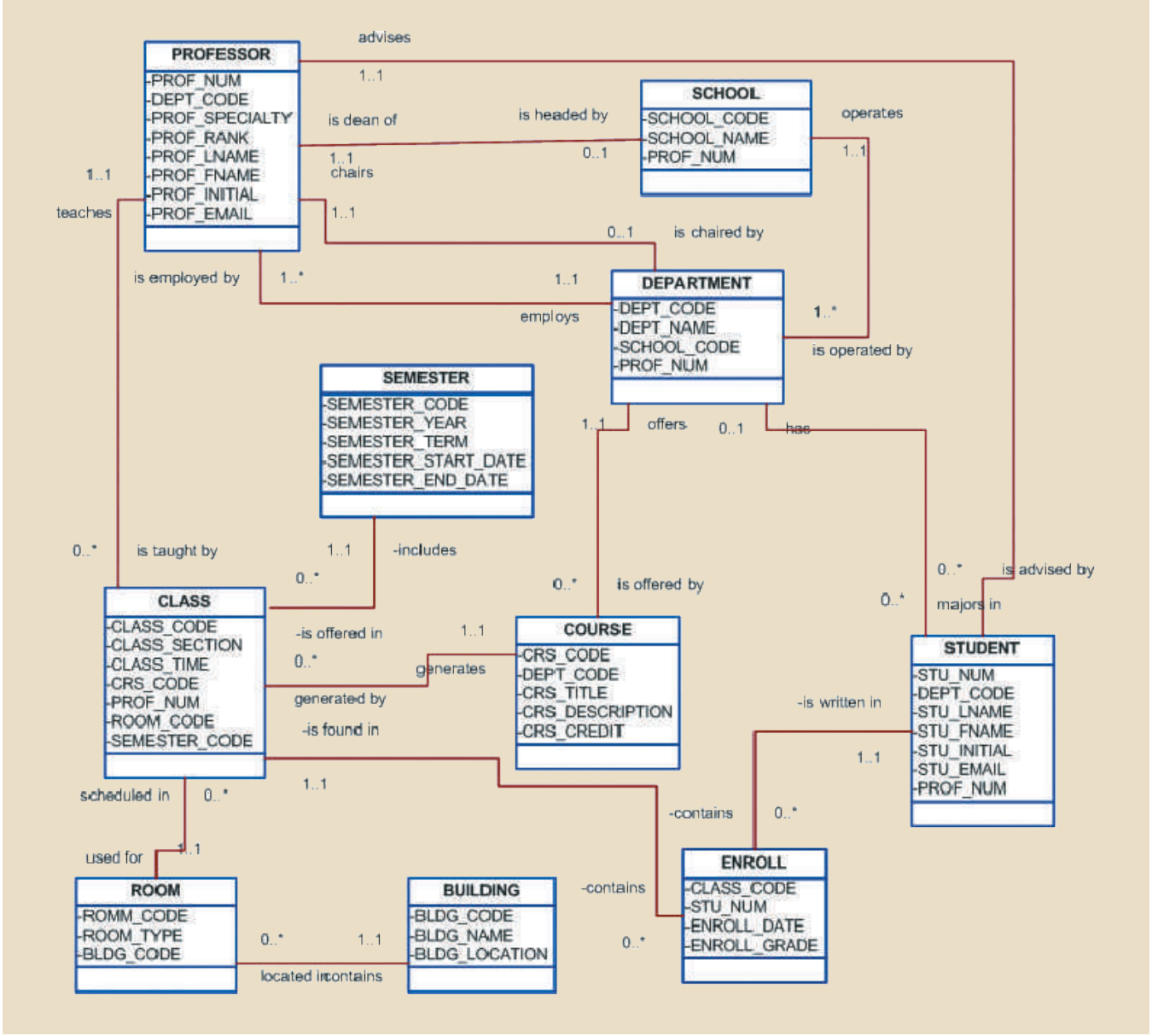


FIGURE 4.37 THE IMPLEMENTATION-READY UML CLASS DIAGRAM FOR TINY COLLEGE



4-3 Database Design Challenges: Conflicting Goals

Database designers must often make design compromises that are triggered by conflicting goals, such as adherence to design standards (design elegance), processing speed, and information requirements.

- *Design standards*. The database design must conform to design standards. Such standards guide you in developing logical structures that minimize data redundancies, thereby minimizing the likelihood that destructive data anomalies will occur. You have also learned how standards prescribe avoiding nulls to the greatest extent possible. In fact, you have learned that design standards govern the presentation of all

components within the database design. In short, design standards allow you to work with well-defined components and to evaluate the interaction of those components with some precision. Without design standards, it is nearly impossible to formulate a proper design process, to evaluate an existing design, or to trace the likely logical impact of changes in design.

- *Processing speed*. In many organizations, particularly those that generate large numbers of transactions, high processing speeds are often a top priority in database design. High processing speed means minimal access time, which may be achieved by minimizing the number and complexity of logically desirable relationships. For example, a "perfect" design might use a 1:1 relationship to avoid nulls, while a design that emphasizes higher transaction speed might combine the two tables to avoid the use of an additional relationship, using dummy entries to avoid the nulls. If the focus is on data-retrieval speed, you might also be forced to include derived attributes in the design.
- *Information requirements*. The quest for timely information might be the focus of database design. Complex information requirements may dictate data transformations, and they may expand the number of entities and attributes within the design. Therefore, the database may have to sacrifice some of its "clean" design structures and high transaction speed to ensure maximum information generation. For example, suppose that a detailed sales report must be generated periodically. The sales report includes all invoice subtotals, taxes, and totals; even the invoice lines include subtotals. If the sales report includes hundreds of thousands (or even millions) of invoices, computing the totals, taxes, and subtotals is likely to take some time. If those computations had been made and the results had been stored as derived attributes in the INVOICE and LINE tables at the time of the transaction, the real-time transaction speed might have declined. However, that loss of speed would only be noticeable if there were many simultaneous transactions. The cost of a slight loss of transaction speed at the front end and the addition of multiple derived attributes is likely to pay off when the sales reports are generated (not to mention that it will be simpler to generate the queries).

A design that meets all logical requirements and design conventions is an important goal. However, if this perfect design fails to meet the customer's transaction speed and information requirements, the designer will not have done a proper job from the end user's point of view. Compromises are a fact of life in the real world of database design.

Even while focusing on the entities, attributes, relationships, and constraints, the designer should begin thinking about end-user requirements such as performance, security, shared access, and data integrity. The designer must consider processing requirements and verify that all update, retrieval, and deletion options are available. Finally, a design is of little value unless the end product can deliver all specified query and reporting requirements.

You will probably discover that even the best design process produces an ERD that requires further changes mandated by operational requirements. Such changes should not discourage you from using the process. ER modeling is essential in the development of a sound design that can meet the demands of adjustment and growth. Using ERDs yields perhaps the richest bonus of all: a thorough understanding of how an organization really functions.

Occasionally, design and implementation problems do not yield "clean" implementation solutions. To get a sense of the design and implementation choices a database designer faces, you will revisit the 1:1 recursive relationship "EMPLOYEE is married to EMPLOYEE," first examined in Figure 4.18. Figure 4.38 shows three different ways of implementing such a relationship.

FIGURE 4.38 VARIOUS IMPLEMENTATIONS OF THE 1:1 RECURSIVE RELATIONSHIP

Table name: EMPLOYEE_V1 — **Database name: Ch04_PartCo**

EMP_NUM	EMP_LNAME	EMP_FNAME	EMP_SPOUSE
345	Ramirez	James	347
346	Jones	Anne	349
347	Ramirez	Louise	345
348	Delaney	Robert	
349	Shapiro	Anton	346

First implementation

Table name: EMPLOYEE

EMP_NUM	EMP_LNAME	EMP_FNAME
345	Ramirez	James
346	Jones	Anne
347	Ramirez	Louise
348	Delaney	Robert
349	Shapiro	Anton

Table name: MARRIED_V1

EMP_NUM	EMP_SPOUSE
345	347
346	349
347	345
349	346

Second implementation

Table name: MARRIAGE

MAR_NUM	MAR_DATE
1	04-Mar-03
2	02-Feb-99

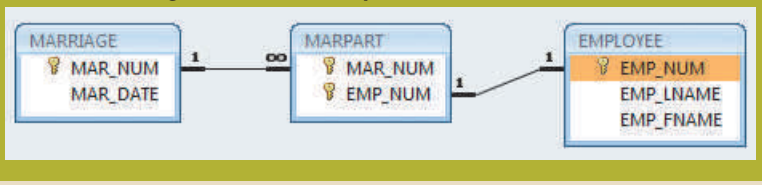
Table name: MARPART

MAR_NUM	EMP_NUM
1	345
1	347
2	346
2	349

Table name: EMPLOYEE

EMP_NUM	EMP_LNAME	EMP_FNAME
345	Ramirez	James
346	Jones	Anne
347	Ramirez	Louise
348	Delaney	Robert
349	Shapiro	Anton

The relational diagram for the third implementation



Third implementation

Note that the EMPLOYEE_V1 table in Figure 4.38 is likely to yield data anomalies. For example, if Anne Jones divorces Anton Shapiro, two records must be updated—by setting the respective EMP_SPOUSE values to null—to properly reflect that change. If only one record is updated, inconsistent data occurs. The problem becomes even worse if several of the divorced employees then marry each other. In addition, that implementation also produces undesirable nulls for employees who are *not* married to other employees in the company.

Another approach would be to create a new entity shown as MARRIED_V1 in a 1:M relationship with EMPLOYEE. (See Figure 4.38.) This second implementation does eliminate the nulls for employees who are not married to other employees in the same company. (Such employees would not be entered in the MARRIED_V1 table.) However, this approach still yields possible duplicate values. For example, the marriage between employees 345 and 347 may still appear twice, once as 345,347 and once as 347,345. (Because each of those permutations is unique the first time it appears, the creation of a unique index will not solve the problem.)

As you can see, the first two implementations yield several problems:

- Both solutions use synonyms. The EMPLOYEE_V1 table uses EMP_NUM and EMP_SPOUSE to refer to an employee. The MARRIED_V1 table uses the same synonyms.

- Both solutions are likely to produce redundant data. For example, it is possible to enter employee 345 as married to employee 347 and to enter employee 347 as married to employee 345.
- Both solutions are likely to produce inconsistent data. For example, it is possible to have data pairs such as 345,347 and 348,345 and 347,349, none of which will violate entity integrity requirements because they are all unique. However, this solution would allow any one employee to be married to multiple employees.

A third approach would be to have two new entities, MARRIAGE and MARPART, in a 1:M relationship. MARPART contains the EMP_NUM foreign key to EMPLOYEE. (See the relational diagram in Figure 4.38.) However, even this approach has issues. It requires the collection of additional data regarding the employees' marriage—the marriage date. If the business users do not need this data, then requiring them to collect it would be inappropriate. To ensure that an employee occurs only once in any given marriage, you would have to create a unique index on the EMP_NUM attribute in the MARPART table. Another potential problem with this solution is that the database implementation would theoretically allow more than two employees to "participate" in the same marriage.

As you can see, a recursive 1:1 relationship yields many different solutions with varying degrees of effectiveness and adherence to basic design principles. Any of the preceding solutions would likely involve the creation of program code to help ensure the integrity and consistency of the data. In a later chapter, you will examine the creation of database triggers that can do exactly that. Your job as a database designer is to use your professional judgment to yield a solution that meets the requirements imposed by business rules, processing requirements, and basic design principles.

Finally, document, document, and document! Put all design activities in writing, and then review what you have written. Documentation not only helps you stay on track during the design process, it also enables you and your coworkers to pick up the design thread when the time comes to modify the design. Although the need for documentation should be obvious, one of the most vexing problems in database and systems analysis work is that this need is often ignored in the design and implementation stages. The development of organizational documentation standards is an important aspect of ensuring data compatibility and coherence.

Summary

- The ERM uses ERDs to represent the conceptual database as viewed by the end user. The ERM's main components are entities, relationships, and attributes. The ERD includes connectivity and cardinality notations, and can also show relationship strength, relationship participation (optional or mandatory), and degree of relationship (such as unary, binary, or ternary).
- Connectivity describes the relationship classification (1:1, 1:M, or M:N). Cardinality expresses the specific number of entity occurrences associated with an occurrence of a related entity. Connectivities and cardinalities are usually based on business rules.
- In the ERM, an M:N relationship is valid at the conceptual level. However, when implementing the ERM in a relational database, the M:N relationship must be mapped to a set of 1:M relationships through a composite entity.
- ERDs may be based on many different ERMs. However, regardless of which model is selected, the modeling logic remains the same. Because no ERM can accurately portray all real-world data and action constraints, application software must be used to augment the implementation of at least some of the business rules.
- Unified Modeling Language (UML) class diagrams are used to represent the static data structures in a data model. The symbols used in the UML class and ER diagrams are very similar. The UML class diagrams can be used to depict data models at the conceptual or implementation abstraction levels.
- Database designers, no matter how well they can produce designs that conform to all applicable modeling conventions, are often forced to make design compromises. Those compromises are required when end users have vital transaction-speed and information requirements that prevent the use of "perfect" modeling logic and adherence to all modeling conventions. Therefore, database designers must use their professional judgment to determine how and to what extent the modeling conventions are subject to modification. To ensure that their professional judgments are sound, database designers must have detailed and in-depth knowledge of data-modeling conventions. It is also important to document the design process from beginning to end, which helps keep the design process on track and allows for easy modifications in the future.

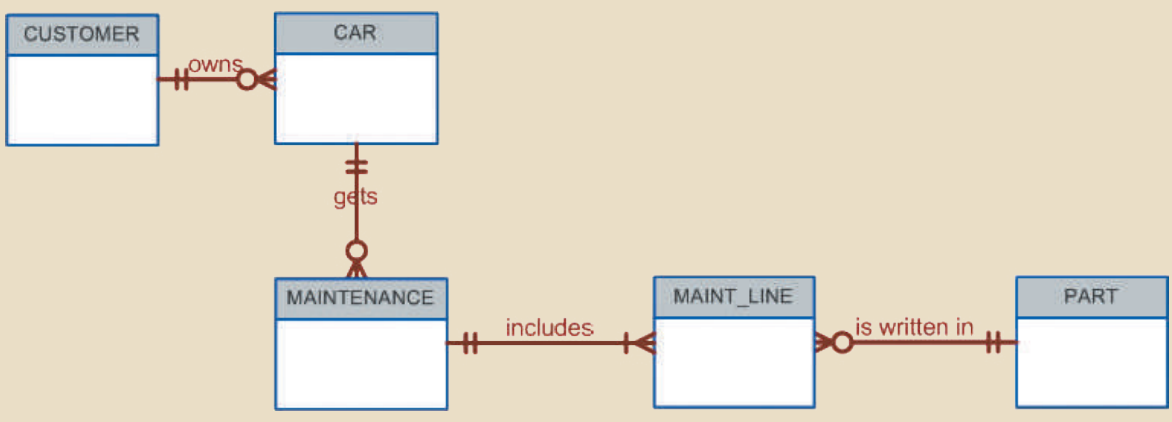
Key Terms

binary relationship
cardinality
composite attribute
composite identifier
connectivity
derived attribute
existence-dependent
existence-independent
identifier
iterative process
mandatory participation
multivalued attribute
optional attribute
optional participation
participants
recursive relationship
regular entity
relational schema
relationship degree
required attribute
simple attribute
single-valued attribute
strong entity
strong (identifying) relationship
ternary relationship
unary relationship
weak entity
weak (non-identifying) relationship

Review Questions

1. What two conditions must be met before an entity can be classified as a weak entity? Give an example of a weak entity.
2. What is a strong (or identifying) relationship, and how is it depicted in a Crow's Foot ERD?
3. Given the business rule "an employee may have many degrees," discuss its effect on attributes, entities, and relationships. (*Hint:* Remember what a multivalued attribute is and how it might be implemented.)
4. What is a composite entity, and when is it used?
5. Suppose you are working within the framework of the conceptual model in Figure Q4.5.

FIGURE Q4.5 THE CONCEPTUAL MODEL FOR QUESTION 5



Given the conceptual model in Figure Q4.5:

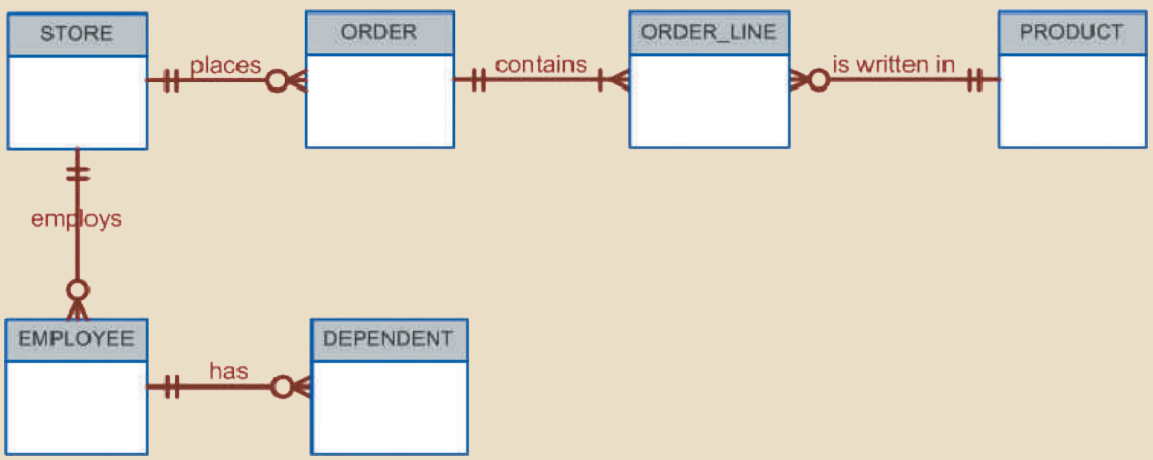
a. Write the business rules that are reflected in it.

b. Identify all of the cardinalities.

6. What is a recursive relationship? Give an example.
7. How would you (graphically) identify each of the following ERM components in a Crow's Foot notation?
   a. an entity
   b. the cardinality (0,N)
   c. a weak relationship
   d. a strong relationship
8. Discuss the difference between a composite key and a composite attribute. How would each be indicated in an ERD?
9. What two courses of action are available to a designer who encounters a multivalued attribute?
10. What is a derived attribute? Give an example. What are the advantages or disadvantages of storing or not storing a derived attribute?
11. How is a relationship between entities indicated in an ERD? Give an example using the Crow's Foot notation.
12. Discuss two ways in which the 1:M relationship between COURSE and CLASS can be implemented. (*Hint:* Think about relationship strength.)
13. How is a composite entity represented in an ERD, and what is its function? Illustrate the Crow's Foot notation.
14. What three (often conflicting) database requirements must be addressed in database design?
15. Briefly, but precisely, explain the difference between single-valued attributes and simple attributes. Give an example of each.
16. What are multivalued attributes, and how can they be handled within the database design?

Questions 17–20 are based on the ERD in Figure Q4.17.

FIGURE Q4.17 THE ERD FOR QUESTIONS 17–20



17. Write the 10 cardinalities that are appropriate for this ERD.
18. Write the business rules reflected in this ERD.
19. What two attributes must be contained in the composite entity between STORE and PRODUCT? Use proper terminology in your answer.
20. Describe precisely the composition of the DEPENDENT weak entity's primary key. Use proper terminology in your answer.
21. The local city youth league needs a database system to help track children who sign up to play soccer. Data needs to be kept on each team, the children who will play on each team, and their parents. Also, data needs to be kept on the coaches for each team.

    Draw a data model with the entities and attributes described here.

    Entities required: Team, Player, Coach, and Parent

    Attributes required:

    Team: Team ID number, Team name, and Team colors

    Player: Player ID number, Player first name, Player last name, and Player age

    Coach: Coach ID number, Coach first name, Coach last name, and Coach home phone number

    Parent: Parent ID number, Parent last name, Parent first name, Home phone number, and Home address (Street, City, State, and Zip code)

    The following relationships must be defined:

    - Team is related to Player.
    - Team is related to Coach.
    - Player is related to Parent.

    Connectivities and participations are defined as follows:

    - A Team may or may not have a Player.
    - A Player must have a Team.
    - A Team may have many Players.
    - A Player has only one Team.
    - A Team may or may not have a Coach.
    - A Coach must have a Team.
    - A Team may have many Coaches.
    - A Coach has only one Team.
    - A Player must have a Parent.
    - A Parent must have a Player.
    - A Player may have many Parents.
    - A Parent may have many Players.

Problems

1. Use the following business rules to create a Crow's Foot ERD. Write all appropriate connectivities and cardinalities in the ERD.
   - A department employs many employees, but each employee is employed by only one department.
   - Some employees, known as "rovers," are not assigned to any department.
   - A division operates many departments, but each department is operated by only one division.
   - An employee may be assigned many projects, and a project may have many employees assigned to it.
   - A project must have at least one employee assigned to it.
   - One of the employees manages each department, and each department is managed by only one employee.
   - One of the employees runs each division, and each division is run by only one employee.
2. Create a complete ERD in Crow's Foot notation that can be implemented in the relational model using the following description of operations. Hot Water (HW) is a small start-up company that sells spas. HW does not carry any stock. A few spas are set up in a simple warehouse so customers can see some of the models available, but any products sold must be ordered at the time of the sale.
   - HW can get spas from several different manufacturers.
   - Each manufacturer produces one or more different brands of spas.
   - Each and every brand is produced by only one manufacturer.
   - Every brand has one or more models.
   - Every model is produced as part of a brand. For example, Iguana Bay Spas is a manufacturer that produces Big Blue Iguana spas, a premium-level brand, and Lazy Lizard spas, an entry-level brand. The Big Blue Iguana brand offers several models, including the BBI-6, an 81-jet spa with two 6-hp motors, and the BBI-10, a 102-jet spa with three 6-hp motors.
   - Every manufacturer is identified by a manufacturer code. The company name, address, area code, phone number, and account number are kept in the system for every manufacturer.
   - For each brand, the brand name and brand level (premium, mid-level, or entry-level) are kept in the system.
   - For each model, the model number, number of jets, number of motors, number of horsepower per motor, suggested retail price, HW retail price, dry weight, water capacity, and seating capacity must be kept in the system.
3. The Jonesburgh County Basketball Conference (JCBC) is an amateur basketball association. Each city in the county has one team as its representative. Each team has a maximum of 12 players and a minimum of 9 players. Each team also has up to 3 coaches (offensive, defensive, and physical training coaches). During the season, each team plays 2 games (home and visitor) against each of the other teams. Given those conditions, do the following:

- Identify the connectivity of each relationship.
- Identify the type of dependency that exists between CITY and TEAM.
- Identify the cardinality between teams and players and between teams and city.
- Identify the dependency between COACH and TEAM and between TEAM and PLAYER.
- Draw the Chen and Crow's Foot ERDs to represent the JCBC database.
- Draw the UML class diagram to depict the JCBC database.

4. Create an ERD based on the Crow's Foot notation using the following requirements:

   - An INVOICE is written by a SALESREP. Each sales representative can write many invoices, but each invoice is written by a single sales representative.
   - The INVOICE is written for a single CUSTOMER. However, each customer can have many invoices.
   - An INVOICE can include many detail lines (LINE), each of which describes one product bought by the customer.
   - The product information is stored in a PRODUCT entity.
   - The product's vendor information is found in a VENDOR entity.

5. The Hudson Engineering Group (HEG) has contacted you to create a conceptual model whose application will meet the expected database requirements for the company's training program. The HEG administrator gives you the following description of the training group's operating environment. (*Hint:* Some of the following sentences identify the volume of data rather than cardinalities. Can you tell which ones?)

   The HEG has 12 instructors and can handle up to 30 trainees per class. HEG offers 5 Advanced Technology courses, each of which may generate several classes. If a class has fewer than 10 trainees, it will be canceled. Therefore, it is possible for a course not to generate any classes. Each class is taught by one instructor. Each instructor may teach up to 2 classes or may be assigned to do research only. Each trainee may take up to 2 classes per year.

   Given that information, do the following:

   a. Define all of the entities and relationships. (Use Table 4.4 as your guide.)
   b. Describe the relationship between instructor and class in terms of connectivity, cardinality, and existence dependence.

6. Automata, Inc., produces specialty vehicles by contract. The company operates several departments, each of which builds a particular vehicle, such as a limousine, truck, van, or RV.

   - Before a new vehicle is built, the department places an order with the purchasing department to request specific components. Automata's purchasing department is interested in creating a database to keep track of orders and to accelerate the process of delivering materials.
   - The order received by the purchasing department may contain several different items. An inventory is maintained so the most frequently requested items are delivered almost immediately. When an order comes in, it is checked to determine whether the requested item is in inventory. If an item is not in inventory, it must be ordered from a supplier. Each item may have several suppliers.

Given that functional description of the processes at Automata's purchasing department, do the following:

a. Identify all of the main entities.

b. Identify all of the relations and connectivities among entities.

c. Identify the type of existence dependence in all the relationships.

d. Give at least two examples of the types of reports that can be obtained from the database.

7. United Helpers is a nonprofit organization that provides aid to people after natural disasters. Based on the following brief description of operations, create the appropriate fully labeled Crow's Foot ERD.

- Volunteers carry out the tasks of the organization. The name, address, and telephone number are tracked for each volunteer. Each volunteer may be assigned to several tasks, and some tasks require many volunteers. A volunteer might be in the system without having been assigned a task yet. It is possible to have tasks that no one has been assigned. When a volunteer is assigned to a task, the system should track the start time and end time of that assignment.
- Each task has a task code, task description, task type, and task status. For example, there may be a task with task code "101," a description of "answer the telephone," a type of "recurring," and a status of "ongoing." Another task might have a code of "102," a description of "prepare 5,000 packages of basic medical supplies," a type of "packing," and a status of "open."
- For all tasks of type "packing," there is a packing list that specifies the contents of the packages. There are many packing lists to produce different packages, such as basic medical packages, child-care packages, and food packages. Each packing list has an ID number, a packing list name, and a packing list description, which describes the items that should make up the package. Every packing task is associated with only one packing list. A packing list may not be associated with any tasks, or it may be associated with many tasks. Tasks that are not packing tasks are not associated with any packing list.
- Packing tasks result in the creation of packages. Each individual package of supplies produced by the organization is tracked, and each package is assigned an ID number. The date the package was created and its total weight are recorded. A given package is associated with only one task. Some tasks (such as "answer the phones") will not produce any packages, while other tasks (such as "prepare 5,000 packages of basic medical supplies") will be associated with many packages.
- The packing list describes the *ideal* contents of each package, but it is not always possible to include the ideal number of each item. Therefore, the actual items included in each package should be tracked. A package can contain many different items, and a given item can be used in many different packages.
- Each item that the organization provides has an item ID number, item description, item value, and item quantity on hand stored in the system. Along with tracking the actual items that are placed in each package, the quantity of each item placed in the package must be tracked as well. For example, a packing list may state that basic medical packages should include 100 bandages, 4 bottles of iodine, and 4 bottles of hydrogen peroxide. However, because of the limited supply of items, a given package may include only 10 bandages, 1 bottle of iodine, and no hydrogen peroxide. The fact that the package includes bandages and iodine needs to be recorded along with the quantity of each item included. It is

possible for the organization to have items that have not been included in any package yet, but every package will contain at least one item.

8. Using the Crow's Foot notation, create an ERD that can be implemented for a medical clinic using the following business rules:
   - A patient can make many appointments with one or more doctors in the clinic, and a doctor can accept appointments with many patients. However, each appointment is made with only one doctor and one patient.
   - Emergency cases do not require an appointment. However, for appointment management purposes, an emergency is entered in the appointment book as "unscheduled."
   - If kept, an appointment yields a visit with the doctor specified in the appointment. The visit yields a diagnosis and, when appropriate, treatment.
   - With each visit, the patient's records are updated to provide a medical history.
   - Each patient visit creates a bill. Each patient visit is billed by one doctor, and each doctor can bill many patients.
   - Each bill must be paid. However, a bill may be paid in many installments, and a payment may cover more than one bill.
   - A patient may pay the bill directly, or the bill may be the basis for a claim submitted to an insurance company.
   - If the bill is paid by an insurance company, the deductible is submitted to the patient for payment.
9. Create a Crow's Foot notation ERD to support the following business operations:
   - A friend of yours has opened Professional Electronics and Repairs (PEAR) to repair smartphones, laptops, tablets, and MP3 players. She wants you to create a database to help her run her business.
   - When a customer brings a device to PEAR for repair, data must be recorded about the customer, the device, and the repair. The customer's name, address, and a contact phone number must be recorded (if the customer has used the shop before, the information already in the system for the customer is verified as being current). For the device to be repaired, the type of device, model, and serial number are recorded (or verified if the device is already in the system). Only customers who have brought devices into PEAR for repair will be included in this system.
   - Since a customer might sell an older device to someone else who then brings the device to PEAR for repair, it is possible for a device to be brought in for repair by more than one customer. However, each repair is associated with only one customer. When a customer brings in a device to be fixed, it is referred to as a repair request, or just "repair," for short. Each repair request is given a reference number, which is recorded in the system along with the date of the request, and a description of the problem(s) that the customer wants fixed. It is possible for a device to be brought to the shop for repair many different times, and only devices that are brought in for repair are recorded in the system. Each repair request is for the repair of one and only one device. If a customer needs multiple devices fixed, then each device will require its own repair request.
   - There are a limited number of repair services that PEAR can perform. For each repair service, there is a service ID number, description, and charge. "Charge" is how much the customer is charged for the shop to perform the service, including

any parts used. The actual repair of a device is the performance of the services necessary to address the problems described by the customer. Completing a repair request may require the performance of many services. Each service can be performed many different times during the repair of different devices, but each service will be performed only once during a given repair request.

- All repairs eventually require the performance of at least one service, but which services will be required may not be known at the time the repair request is made. It is possible for services to be available at PEAR but that have never been required in performing any repair.
- Some services involve only labor activities and no parts are required, but most services require the replacement of one or more parts. The quantity of each part required in the performance of each service should also be recorded. For each part, the part number, part description, quantity in stock, and cost is recorded in the system. The cost indicated is the amount that PEAR pays for the part. Some parts may be used in more than one service, but each part is required for at least one service.

10. Luxury-Oriented Scenic Tours (LOST) provides guided tours to groups of visitors to the Washington, D.C. area. In recent years, LOST has grown quickly and is having difficulty keeping up with all of the various information needs of the company. The company's operations are as follows:
    - LOST offers many different tours. For each tour, the tour name, approximate length (in hours), and fee charged is needed. Guides are identified by an employee ID, but the system should also record a guide's name, home address, and date of hire. Guides take a test to be qualified to lead specific tours. It is important to know which guides are qualified to lead which tours and the date that they completed the qualification test for each tour. A guide may be qualified to lead many different tours. A tour can have many different qualified guides. New guides may or may not be qualified to lead any tours, just as a new tour may or may not have any qualified guides.
    - Every tour must be designed to visit at least three locations. For each location, a name, type, and official description are kept. Some locations (such as the White House) are visited by more than one tour, while others (such as Arlington Cemetery) are visited by a single tour. All locations are visited by at least one tour. The order in which the tour visits each location should be tracked as well.
    - When a tour is actually given, that is referred to as an "outing." LOST schedules outings well in advance so they can be advertised and so employees can understand their upcoming work schedules. A tour can have many scheduled outings, although newly designed tours may not have any outings scheduled. Each outing is for a single tour and is scheduled for a particular date and time. All outings must be associated with a tour. All tours at LOST are guided tours, so a guide must be assigned to each outing. Each outing has one and only one guide. Guides are occasionally asked to lead an outing of a tour even if they are not officially qualified to lead that tour. Newly hired guides may not have ever been scheduled to lead any outings. Tourists, called "clients" by LOST, pay to join a scheduled outing. For each client, the name and telephone number are recorded. Clients may sign up to join many different outings, and each outing can have many clients. Information is kept only on clients who have signed up for at least one outing, although newly scheduled outings may not have any clients signed up yet.

a. Create a Crow's Foot notation ERD to support LOST operations.

b. The operations provided state that it is possible for a guide to lead an outing of a tour even if the guide is not officially qualified to lead outings of that tour. Imagine that the business rules instead specified that a guide is never, under any circumstance, allowed to lead an outing unless he or she is qualified to lead outings of that tour. How could the data model in Part a. be modified to enforce this new constraint?

Note

You can use the following cases and additional problems from the Instructor Online Companion as the basis for class projects. These problems illustrate the challenge of translating a description of operations into a set of business rules that will define the components for an ERD you can implement successfully. These problems can also be used as the basis for discussions about the components and contents of a proper description of operations. If you want to create databases that can be successfully implemented, you must learn to separate the generic background material from the details that directly affect database design. You must also keep in mind that many constraints cannot be incorporated into the database design; instead, such constraints are handled by the application software.

Cases

11. The administrators of Tiny College are so pleased with your design and implementation of their student registration and tracking system that they want you to expand the design to include the database for their motor vehicle pool. A brief description of operations follows:

- Faculty members may use the vehicles owned by Tiny College for officially sanctioned travel. For example, the vehicles may be used by faculty members to travel to off-campus learning centers, to travel to locations at which research papers are presented, to transport students to officially sanctioned locations, and to travel for public service purposes. The vehicles used for such purposes are managed by Tiny College's Travel Far But Slowly (TFBS) Center.
- Using reservation forms, each department can reserve vehicles for its faculty, who are responsible for filling out the appropriate trip completion form at the end of a trip. The reservation form includes the expected departure date, vehicle type required, destination, and name of the authorized faculty member. The faculty member who picks up a vehicle must sign a checkout form to log out the vehicle and pick up a trip completion form. (The TFBS employee who releases the vehicle for use also signs the checkout form.) The faculty member's trip completion form includes the faculty member's identification code, the vehicle's identification, the odometer readings at the start and end of the trip, maintenance complaints (if any), gallons of fuel purchased (if any), and the Tiny College credit card number used to pay for the fuel. If fuel is purchased, the credit card receipt must be stapled to the trip completion form. Upon receipt of the trip completion form, the faculty member's department is billed at a mileage rate based on the vehicle type used: sedan, station wagon, panel truck, minivan, or minibus. (*Hint:* Do *not* use more entities than are necessary. Remember the difference between attributes and entities!)

- All vehicle maintenance is performed by TFBS. Each time a vehicle requires maintenance, a maintenance log entry is completed on a prenumbered maintenance log form. The maintenance log form includes the vehicle identification, brief description of the type of maintenance required, initial log entry date, date the maintenance was completed, and name of the mechanic who released the vehicle back into service. (Only mechanics who have an inspection authorization may release a vehicle back into service.)
- As soon as the log form has been initiated, the log form's number is transferred to a maintenance detail form; the log form's number is also forwarded to the parts department manager, who fills out a parts usage form on which the maintenance log number is recorded. The maintenance detail form contains separate lines for each maintenance item performed, for the parts used, and for identification of the mechanic who performed the maintenance. When all maintenance items have been completed, the maintenance detail form is stapled to the maintenance log form, the maintenance log form's completion date is filled out, and the mechanic who releases the vehicle back into service signs the form. The stapled forms are then filed, to be used later as the source for various maintenance reports.
- TFBS maintains a parts inventory, including oil, oil filters, air filters, and belts of various types. The parts inventory is checked daily to monitor parts usage and to reorder parts that reach the "minimum quantity on hand" level. To track parts usage, the parts manager requires each mechanic to sign out the parts that are used to perform each vehicle's maintenance; the parts manager records the maintenance log number under which the part is used.
- Each month TFBS issues a set of reports. The reports include the mileage driven by vehicle, by department, and by faculty members within a department. In addition, various revenue reports are generated by vehicle and department. A detailed parts usage report is also filed each month. Finally, a vehicle maintenance summary is created each month.

Given that brief summary of operations, draw the appropriate (and fully labeled) ERD. Use the Crow's foot methodology to indicate entities, relationships, connectivities, and participations.

12. During peak periods, Temporary Employment Corporation (TEC) places temporary workers in companies. TEC's manager gives you the following description of the business:

    - TEC has a file of candidates who are willing to work.
    - Any candidate who has worked before has a specific job history. (Naturally, no job history exists if the candidate has never worked.) Each time the candidate works, one additional job history record is created.
    - Each candidate has earned several qualifications. Each qualification may be earned by more than one candidate. (For example, more than one candidate may have earned a Bachelor of Business Administration degree or a Microsoft Network Certification, and clearly a candidate may have earned both a BBA and a Microsoft Network Certification.)
    - TEC offers courses to help candidates improve their qualifications.
    - Every course develops one specific qualification; however, TEC does not offer a course for every qualification. Some qualifications are developed through multiple courses.
    - Some courses cover advanced topics that require specific qualifications as prerequisites. Some courses cover basic topics that do not require any prerequisite

qualifications. A course can have several prerequisites. A qualification can be a prerequisite for more than one course.

- Courses are taught during training sessions. A training session is the presentation of a single course. Over time, TEC will offer many training sessions for each course; however, new courses may not have any training sessions scheduled right away.
- Candidates can pay a fee to attend a training session. A training session can accommodate several candidates, although new training sessions will not have any candidates registered at first.
- TEC also has a list of companies that request temporaries.
- Each time a company requests a temporary employee, TEC makes an entry in the Openings folder. That folder contains an opening number, a company name, required qualifications, a starting date, an anticipated ending date, and hourly pay.
- Each opening requires only one specific or main qualification.
- When a candidate matches the qualification, the job is assigned, and an entry is made in the Placement Record folder. The folder contains such information as an opening number, candidate number, and total hours worked. In addition, an entry is made in the job history for the candidate.
- An opening can be filled by many candidates, and a candidate can fill many openings.
- TEC uses special codes to describe a candidate's qualifications for an opening. The list of codes is shown in Table P4.12.

TABLE P4.12

CODES FOR PROBLEM 12

CODE	DESCRIPTION
SEC-45	Secretarial work; candidate must type at least 45 words per minute
SEC-60	Secretarial work; candidate must type at least 60 words per minute
CLERK	General clerking work
PRG-VB	Programmer, Visual Basic
PRG-C++	Programmer, C++
DBA-ORA	Database Administrator, Oracle
DBA-DB2	Database Administrator, IBM DB2
DBA-SQLSERV	Database Administrator, MS SQL Server
SYS-1	Systems Analyst, level 1
SYS-2	Systems Analyst, level 2
NW-NOV	Network Administrator, Novell experience
WD-CF	Web Developer, ColdFusion

TEC's management wants to keep track of the following entities:

COMPANY, OPENING, QUALIFICATION, CANDIDATE, JOB_HISTORY, PLACEMENT, COURSE, and SESSION. Given that information, do the following:

a. Draw the Crow's Foot ERDs for this enterprise.

b. Identify all necessary relationships.

c. Identify the connectivity for each relationship.

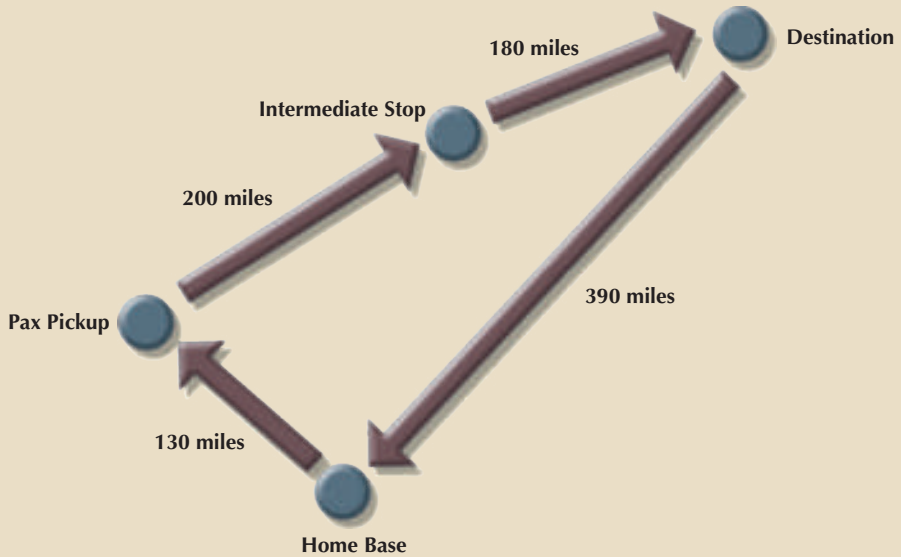
d. Identify the mandatory and optional dependencies for the relationships.

e. Resolve all M:N relationships.

13. Use the following description of the operations of the RC_Charter2 Company to complete this exercise:

- The RC_Charter2 Company operates a fleet of aircraft under the Federal Air Regulations (FAR) Part 135 (air taxi or charter) certificate, enforced by the FAA. The aircraft are available for air taxi (charter) operations within the United States and Canada.
- Charter companies provide so-called unscheduled operations—that is, charter flights take place only after a customer reserves the use of an aircraft at a designated date and time to fly to one or more designated destinations; the aircraft transports passengers, cargo, or some combination of passengers and cargo. Of course, a customer can reserve many different charter trips during any time frame. However, for billing purposes, each charter trip is reserved by one and only one customer. Some of RC_Charter2's customers do not use the company's charter operations; instead, they purchase fuel, use maintenance services, or use other RC_Charter2 services. However, this database design will focus on the charter operations only.
- Each charter trip yields revenue for the RC_Charter2 Company. This revenue is generated by the charges a customer pays upon the completion of a flight. The charter flight charges are a function of aircraft model used, distance flown, waiting time, special customer requirements, and crew expenses. The distance flown charges are computed by multiplying the round-trip miles by the model's charge per mile. Round-trip miles are based on the actual navigational path flown. The sample route traced in Figure P4.13 illustrates the procedure. Note that the number of round-trip miles is calculated to be 130 + 200 + 180 + 390 = 900.

FIGURE P4.13 ROUND-TRIP MILE DETERMINATION



- Depending on whether a customer has RC_Charter2 credit authorization, the customer may do the following:
  a. Pay the entire charter bill upon the completion of the charter flight.
  b. Pay a part of the charter bill and charge the remainder to the account. The charge amount may not exceed the available credit.

c. Charge the entire charter bill to the account. The charge amount may not exceed the available credit.

d. Customers may pay all or part of the existing balance for previous charter trips. Such payments may be made at any time and are not necessarily tied to a specific charter trip. The charter mileage charge includes the expense of the pilot(s) and other crew required by FAR 135. However, if customers request *additional* crew *not* required by FAR 135, those customers are charged for the crew members on an hourly basis. The hourly crew-member charge is based on each crew member's qualifications.

e. The database must be able to handle crew assignments. Each charter trip requires the use of an aircraft, and a crew flies each aircraft. The smaller, piston-engine charter aircraft require a crew consisting of only a single pilot. All jets and other aircraft that have a gross takeoff weight of at least 12,500 pounds require a pilot and a copilot, while some of the larger aircraft used to transport passengers may require flight attendants as part of the crew. Some of the older aircraft require the assignment of a flight engineer, and larger cargo-carrying aircraft require the assignment of a loadmaster. In short, a crew can consist of more than one person, and not all crew members are pilots.

f. The charter flight's aircraft waiting charges are computed by multiplying the hours waited by the model's hourly waiting charge. Crew expenses are limited to meals, lodging, and ground transportation.

The RC_Charter2 database must be designed to generate a monthly summary of all charter trips, expenses, and revenues derived from the charter records. Such records are based on the data that each pilot in command is required to record for each charter trip: trip date(s) and time(s), destination(s), aircraft number, pilot data and other crew data, distance flown, fuel usage, and other data pertinent to the charter flight. Such charter data is then used to generate monthly reports that detail revenue and operating cost information for customers, aircraft, and pilots. All pilots and other crew members are RC_Charter2 Company employees; that is, the company does not use contract pilots and crew.

FAR Part 135 operations are conducted under a strict set of requirements that govern the licensing and training of crew members. For example, pilots must have earned either a commercial license or an Airline Transport Pilot (ATP) license. Both licenses require appropriate ratings, which are specific competency requirements. For example, consider the following:

- To operate a multiengine aircraft designed for takeoffs and landings on land only, the appropriate rating is MEL, or Multiengine Landplane. When a multiengine aircraft can take off and land on water, the appropriate rating is MES, or Multiengine Seaplane.
- The instrument rating is based on a demonstrated ability to conduct all flight operations with sole reference to cockpit instrumentation. The instrument rating is required to operate an aircraft under Instrument Meteorological Conditions (IMC), and all such operations are governed under FAR-specified Instrument Flight Rules (IFR). In contrast, operations conducted under "good weather" or *visual* flight conditions are based on the FAR Visual Flight Rules (VFR).
- The type rating is required for all aircraft with a takeoff weight of more than 12,500 pounds or for aircraft that are purely jet-powered. If an aircraft uses jet engines to drive propellers, that aircraft is said to be turboprop-powered. A turboprop—that is, a turbo-propeller-powered aircraft—does not require a type rating unless it meets the 12,500-pound weight limitation.

- Although pilot licenses and ratings are not time limited, exercising the privilege of the license and ratings under Part 135 requires both *a current medical certificate and a current Part 135 checkride*. The following distinctions are important:
  a. The medical certificate may be Class I or Class II. The Class I medical is more stringent than the Class II, and it must be renewed every six months. The Class II medical must be renewed yearly. If the Class I medical is not renewed during the six-month period, it automatically reverts to a Class II certificate. If the Class II medical is not renewed within the specified period, it automatically reverts to a Class III medical, which is not valid for commercial flight operations.
  b. A Part 135 checkride is a practical flight examination that must be successfully completed every six months. The checkride includes all flight maneuvers and procedures specified in Part 135.

Nonpilot crew members must also have the proper certificates to meet specific job requirements. For example, loadmasters need an appropriate certificate, as do flight attendants. Crew members such as loadmasters and flight attendants may be required in operations that involve large aircraft with a takeoff weight of more than 12,500 pounds and more than 19 passengers; these crew members are also required to pass a written and practical exam periodically. The RC_Charter2 Company is required to keep a complete record of all test types, dates, and results for each crew member, as well as examination dates for pilot medical certificates.

In addition, all flight crew members are required to submit to periodic drug testing; the results must be tracked as well. Note that nonpilot crew members are not required to take pilot-specific tests such as Part 135 checkrides, nor are pilots required to take crew tests such as loadmaster and flight attendant practical exams. However, many crew members have licenses and certifications in several areas. For example, a pilot may have an ATP and a loadmaster certificate. If that pilot is assigned to be a loadmaster on a given charter flight, the loadmaster certificate is required. Similarly, a flight attendant may have earned a commercial pilot's license. Sample data formats are shown in Table P4.13.

Pilots and other crew members must receive recurrency training appropriate to their work assignments. Recurrency training is based on an FAA-approved curriculum that is job specific. For example, pilot recurrency training includes a review of all applicable Part 135 flight rules and regulations, weather data interpretation, company flight operations requirements, and specified flight procedures. The RC_Charter2 Company is required to keep a complete record of all recurrency training for each crew member subject to the training.

The RC_Charter2 Company is required to maintain a detailed record of all crew credentials and all training mandated by Part 135. The company must keep a complete record of each requirement and of all compliance data.

To conduct a charter flight, the company must have a properly maintained aircraft available. A pilot who meets all of the FAA's licensing and currency requirements must fly the aircraft as Pilot in Command (PIC). For aircraft that are powered by piston engines or turboprops and have a gross takeoff weight under 12,500 pounds, single-pilot operations are permitted under Part 135 as long as a properly maintained autopilot is available. However, even if FAR Part 135 permits single-pilot operations, many customers require the presence of a copilot who is capable of conducting the flight operations under Part 135.

TABLE P4.13

PART A TESTS			
TEST CODE	TEST DESCRIPTION		TEST FREQUENCY
1	Part 135 Flight Check		6 months
2	Medical, Class I		6 months
3	Medical, Class II		12 months
4	Loadmaster Practical		12 months
5	Flight Attendant Practical		12 months
6	Drug test		Random
7	Operations, written exam		6 months
PART B RESULTS			
EMPLOYEE	TEST CODE	TEST DATE	TEST RESULT
101	1	12-Nov-17	Pass-1
103	6	23-Dec-17	Pass-1
112	4	23-Dec-17	Pass-2
103	7	11-Jan-18	Pass-1
112	7	16-Jan-18	Pass-1
101	7	16-Jan-18	Pass-1
101	6	11-Feb-18	Pass-2
125	2	15-Feb-18	Pass-1
PART C LICENSES AND CERTIFICATIONS			
LICENSE OR CERTIFICATE		LICENSE OR CERTIFICATE DESCRIPTION	
ATP		Airline Transport Pilot	
Comm		Commercial license	
Med-1		Medical certificate, Class I	
Med-2		Medical certificate, Class II	
Instr		Instrument rating	
MEL		Multiengine Land aircraft rating	
LM		Loadmaster	
FA		Flight Attendant	
EMPLOYEE	LICENSE OR CERTIFICATE	DATE EARNED	
101	Comm	12-Nov-93	
101	Instr	28-Jun-94	
101	MEL	9-Aug-94	
103	Comm	21-Dec-95	
112	FA	23-Jun-02	
103	Instr	18-Jan-96	
112	LM	27-Nov-05	

The RC_Charter2 operations manager anticipates the lease of turbojet-powered aircraft, which are required to have a crew consisting of a pilot and copilot. Both the pilot and copilot must meet the same Part 135 licensing, ratings, and training requirements.

The company also leases larger aircraft that exceed the 12,500-pound gross takeoff weight. Those aircraft might carry enough passengers to require the presence of one or more flight attendants. If those aircraft carry cargo that weighs more than 12,500 pounds, a loadmaster must be assigned as a crew member to supervise the loading and securing of the cargo. *The database must be designed to meet the anticipated capability for additional charter crew assignments.*

a. Given this incomplete description of operations, write all applicable business rules to establish entities, relationships, optionalities, connectivities, and cardinalities. (*Hint:* Use the following five business rules as examples, and write the remaining business rules in the same format.) A customer may request many charter trips.
   - Each charter trip is requested by only one customer.
   - Some customers have not yet requested a charter trip.
   - An employee may be assigned to serve as a crew member on many charter trips.
   - Each charter trip may have many employees assigned to serve as crew members.

b. Draw the fully labeled and implementable Crow's Foot ERD based on the business rules you wrote in Part a. of this problem. Include all entities, relationships, optionalities, connectivities, and cardinalities.

Chapter 5

Advanced Data Modeling

After completing this chapter, you will be able to:

- Describe the main extended entity relationship (EER) model constructs and how they are represented in ERDs and EERDs
- Use entity clusters to represent multiple entities and relationships in an entity relationship diagram (ERD)
- Describe the characteristics of good primary keys and how to select them
- Apply flexible solutions for special data-modeling cases

Preview

In the previous two chapters, you learned how to use entity relationship diagrams (ERDs) to properly create a data model. In this chapter, you will learn about the extended entity relationship (EER) model. The EER model builds on entity relationship (ER) concepts and adds support for entity supertypes, subtypes, and entity clustering.

Most current database implementations are based on relational databases. Because the relational model uses keys to create associations among tables, it is essential to learn the characteristics of good primary keys and how to select them. Selecting a good primary key is too important to be left to chance, so this chapter covers the critical aspects of primary key identification and placement.

Focusing on practical database design, this chapter also illustrates some special design cases that highlight the importance of flexible designs, which can be adapted to meet the demands of changing data and information requirements. Data modeling is a vital step in the development of databases that in turn provides a good foundation for successful application development. Remember that good database applications cannot be based on bad database designs, and no amount of outstanding coding can overcome the limitations of poor database design.

Data Files and Available Formats

	MS Access	Oracle	MS SQL	My SQL		MS Access	Oracle	MS SQL	My SQL
CH05_AirCo	✓	✓	✓	✓	CH05_GCSdata	✓	✓	✓	✓
CH05_TinyCollege	✓	✓	✓	✓					

Data Files Available on cengagebrain.com

Note

The extended entity relationship model discussed in this chapter includes advanced data modeling constructs such as specialization hierarchies. Although Microsoft Visio 2010 and earlier versions handled these constructs neatly, newer versions of Visio starting with Microsoft Visio 2013 removed support for many database modeling activities, including specialization hierarchies.

5-1 The Extended Entity Relationship Model

As the complexity of the data structures being modeled has increased and as application software requirements have become more stringent, the need to capture more information in the data model has increased. The **extended entity relationship model (EERM)**, sometimes referred to as the enhanced entity relationship model, is the result of adding more semantic constructs to the original ER model. As you might expect, a diagram that uses the EERM is called an **EER diagram (EERD)**. In the following sections, you will learn about the main EER model constructs—entity supertypes, entity subtypes, and entity clustering—and see how they are represented in ERDs or EERDs.

extended entity relationship model (EERM)
Sometimes referred to as the enhanced entity relationship model; the result of adding more semantic constructs, such as entity supertypes, entity subtypes, and entity clustering, to the original entity relationship (ER) model.

EER diagram (EERD)
The entity relationship diagram resulting from the application of extended entity relationship concepts that provide additional semantic content in the ER model.

entity supertype
In a generalization or specialization hierarchy, a generic entity type that contains the common characteristics of entity subtypes.

entity subtype
In a generalization or specialization hierarchy, a subset of an entity supertype. The entity supertype contains the common characteristics and the subtypes contain the unique characteristics of each entity.

5-1a Entity Supertypes and Subtypes

Because most employees possess a wide range of skills and special qualifications, data modelers must find a variety of ways to group employees based on their characteristics. For instance, a retail company could group employees as salaried and hourly, while a university could group employees as faculty, staff, and administrators.

The grouping of employees into various *types* provides two important benefits:

- It avoids unnecessary nulls in attributes when some employees have characteristics that are not shared by other employees.
- It enables a particular employee type to participate in relationships that are unique to that employee type.

To illustrate those benefits, you will explore the case of an aviation business that employs pilots, mechanics, secretaries, accountants, database managers, and many other types of employees. Figure 5.1 illustrates how pilots share certain characteristics with other employees, such as a last name (EMP_LNAME) and hire date (EMP_HIRE_DATE). On the other hand, many pilot characteristics are not shared by other employees. For example, unlike other employees, pilots must meet special requirements such as flight hour restrictions, flight checks, and periodic training. Therefore, if all employee characteristics and special qualifications were stored in a single EMPLOYEE entity, you would have a lot of nulls or you would have to create a lot of needless dummy entries. In this case, special pilot characteristics such as EMP_LICENSE, EMP_RATINGS, and EMP_MED_TYPE will generate nulls for employees who are not pilots. In addition, pilots participate in some relationships that are unique to their qualifications. For example, not all employees can fly airplanes; only employees who are pilots can participate in the "employee flies airplane" relationship.

Based on the preceding discussion, you would correctly deduce that the PILOT entity stores only attributes that are unique to pilots and that the EMPLOYEE entity stores attributes that are common to all employees. Based on that hierarchy, you can conclude that PILOT is a *subtype* of EMPLOYEE and that EMPLOYEE is the *supertype* of PILOT. In modeling terms, an **entity supertype** is a generic entity type that is related to one or more **entity subtypes**. The entity supertype contains common characteristics, and the entity subtypes each contain their own unique characteristics.

FIGURE 5.1 NULLS CREATED BY UNIQUE ATTRIBUTES

Database name: Ch05_AirCo

EMP_NUM	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_LICENSE	EMP_RATINGS	EMP_MED_TYPE	EMP_HIRE_DATE
100	Kolmycz	Xavier	T				15-Mar-88
101	Lewis	Marcos		ATP	SEL/MEL/Instr/CFII	1	25-Apr-89
102	Vandam	Jean					20-Dec-93
103	Jones	Victoria	R				28-Aug-03
104	Lange	Edith		ATP	SEL/MEL/Instr	1	20-Oct-97
105	Williams	Gabriel	U	COM	SEL/MEL/Instr/CFI	2	08-Nov-97
106	Duzak	Mario		COM	SEL/MEL/Instr	2	05-Jan-04
107	Diante	Venite	L				02-Jul-97
108	Wiesenbach	Joni					18-Nov-95
109	Travis	Brett	T	COM	SEL/MEL/SES/Instr/CFII	1	14-Apr-01
110	Genkazi	Stan					01-Dec-03

Two criteria help the designer determine when to use subtypes and supertypes:

- There must be different, identifiable kinds or types of the entity in the user's environment.
- The different kinds or types of instances should each have one or more attributes that are unique to that kind or type of instance.

In the preceding example, because pilots meet both criteria of being an identifiable kind of employee and having unique attributes that other employees do not possess, it is appropriate to create PILOT as a subtype of EMPLOYEE. Assume that mechanics and accountants also each have attributes that are unique to them, respectively, and that clerks do not. In that case, MECHANIC and ACCOUNTANT would also be legitimate subtypes of EMPLOYEE because they are identifiable kinds of employees and have unique attributes. CLERK would *not* be an acceptable subtype of EMPLOYEE because it only satisfies one of the criteria—it is an identifiable kind of employee—but none of the attributes are unique to just clerks. In the next section, you will learn how entity supertypes and subtypes are related in a specialization hierarchy.

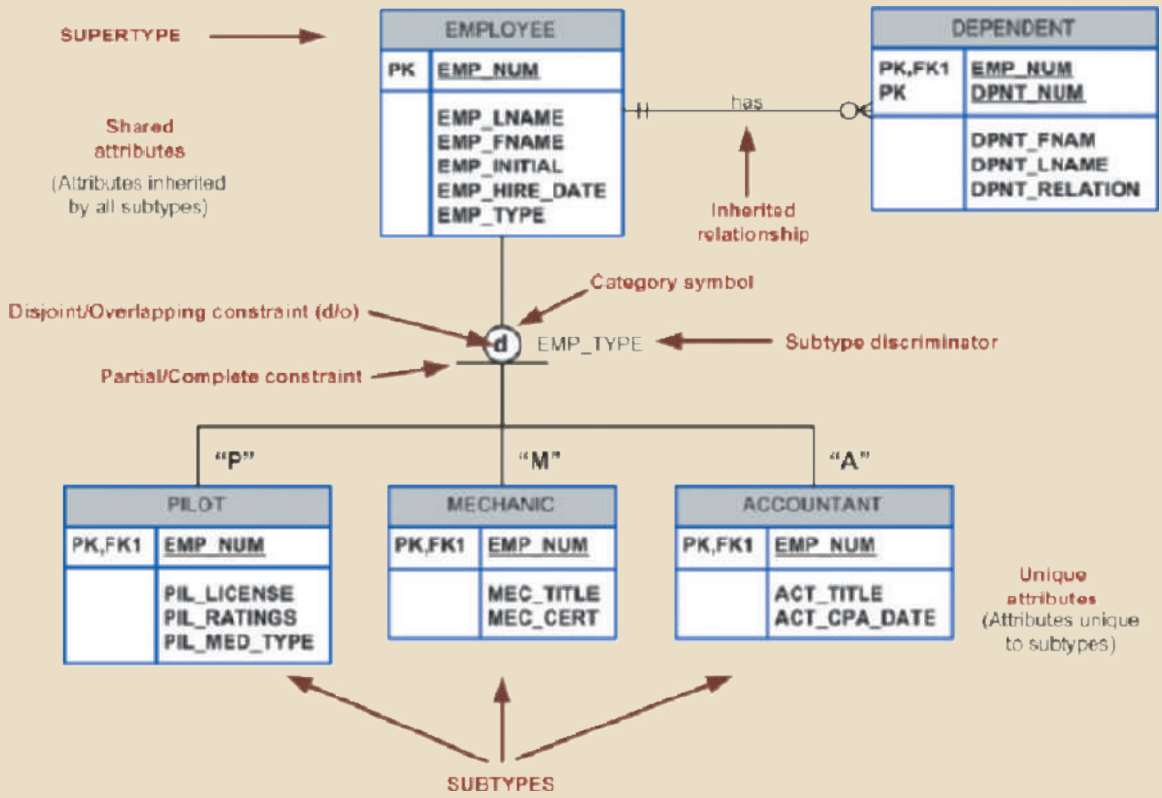
5-1b Specialization Hierarchy

Entity supertypes and subtypes are organized in a **specialization hierarchy**, which depicts the arrangement of higher-level entity supertypes (parent entities) and lower-level entity subtypes (child entities). Figure 5.2 shows the specialization hierarchy formed by an EMPLOYEE supertype and three entity subtypes—PILOT, MECHANIC, and ACCOUNTANT. The specialization hierarchy reflects the 1:1 relationship between EMPLOYEE and its subtypes. For example, a PILOT subtype occurrence is related to one instance of the EMPLOYEE supertype, and a MECHANIC subtype occurrence is related to one instance of the EMPLOYEE supertype. The terminology and symbols in Figure 5.2 are explained throughout this chapter.

The relationships depicted within the specialization hierarchy are sometimes described in terms of "is-a" relationships. For example, a pilot *is an* employee, a mechanic *is an* employee, and an accountant *is an* employee. It is important to understand that within a specialization hierarchy, a subtype can exist only within the context of a supertype, and every subtype can have only one supertype to which it is directly related. However, a specialization hierarchy can have many levels of supertype or subtype relationships—that is, you can have a specialization hierarchy in which a supertype has many subtypes. In turn, one of the subtypes is the supertype to other lower-level subtypes.

specialization hierarchy
A hierarchy based on the top-down process of identifying lower-level, more specific entity subtypes from a higher-level entity supertype. Specialization is based on grouping unique characteristics and relationships of the subtypes.

FIGURE 5.2 A SPECIALIZATION HIERARCHY



Online Content

This chapter covers only specialization hierarchies. The EER model also supports specialization *lattices*, in which a subtype can have multiple parents (supertypes). However, those concepts are better covered under the object-oriented model in Appendix G, Object-Oriented Databases. The appendix is available at *www.cengagebrain.com*.

As you can see in Figure 5.2, the arrangement of entity supertypes and subtypes in a specialization hierarchy is more than a cosmetic convenience. Specialization hierarchies enable the data model to capture additional semantic content (meaning) into the ERD. A specialization hierarchy provides the means to:

- Support attribute inheritance.
- Define a special supertype attribute known as the subtype discriminator.
- Define disjoint or overlapping constraints and complete or partial constraints.

The following sections cover such characteristics and constraints in more detail.

5-1c Inheritance

The property of **inheritance** enables an entity subtype to inherit the attributes and relationships of the supertype. As discussed earlier, a supertype contains attributes that are common to all of its subtypes. In contrast, subtypes contain only the attributes that are unique to the subtype. For example, Figure 5.2 illustrates that pilots, mechanics, and accountants all inherit the employee number, last name, first name, middle initial, and hire date from the EMPLOYEE entity. However, Figure 5.2 also illustrates that pilots have unique attributes; the same is true for mechanics and accountants. *One important inheritance characteristic is that all entity subtypes inherit their primary key attribute from their supertype.* Note in Figure 5.2 that the EMP_NUM attribute is the primary key for each of the subtypes.

inheritance
In the EERD, the property that enables an entity subtype to inherit the attributes and relationships of the entity supertype.

At the implementation level, the supertype and its subtype(s) depicted in the specialization hierarchy maintain a 1:1 relationship. For example, the specialization hierarchy lets you replace the undesirable EMPLOYEE table structure in Figure 5.1 with two tables—one representing the supertype EMPLOYEE and the other representing the subtype PILOT. (See Figure 5.3.)

FIGURE 5.3 THE EMPLOYEE-PILOT SUPERTYPE-SUBTYPE RELATIONSHIP

Database name: Ch05_AirCo

Table name: EMPLOYEE

EMP_NUM	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_HIRE_DATE	EMP_TYPE
100	Kolmycz	Xavier	T	15-Mar-88	
101	Lewis	Marcos		25-Apr-09	P
102	Vandam	Jean		20-Dec-93	A
103	Jones	Victoria	R	28-Aug-03	
104	Lange	Edith		20-Oct-97	P
105	Williams	Gabriel	U	08-Nov-97	P
106	Duzak	Mario		05-Jan-04	P
107	Diante	Venite	L	02-Jul-97	M
108	Wesenbach	Joni		18-Nov-95	M
109	Travis	Brett	T	14-Apr-01	P
110	Genkazi	Stan		01-Dec-03	A

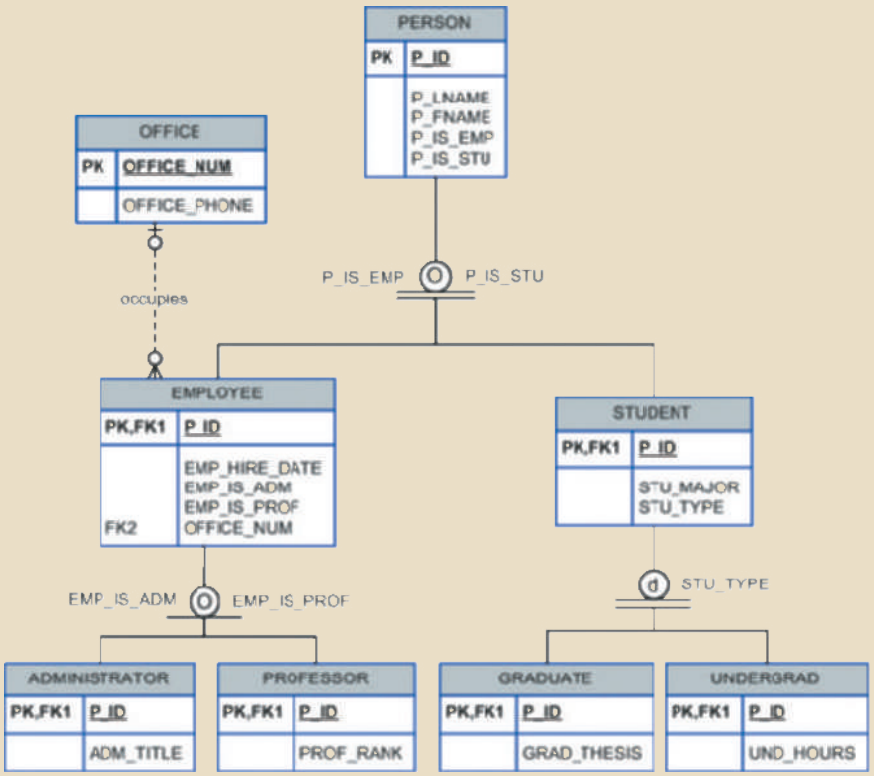
Table name: PILOT

EMP_NUM	PIL_LICENSE	PIL_RATINGS	PIL_MED_TYPE
101	ATP	SEL/MEL/Instr/CFII	1
104	ATP	SEL/MEL/Instr	1
105	COM	SEL/MEL/Instr/CFI	2
106	COM	SEL/MEL/Instr	2
109	COM	SEL/MEL/SES/Instr/CFII	1

Entity subtypes inherit all relationships in which the supertype entity participates. For example, Figure 5.2 shows the EMPLOYEE entity supertype participating in a 1:M relationship with a DEPENDENT entity. Through inheritance, all subtypes also participate in that relationship. In specialization hierarchies with multiple levels of supertype and subtypes, a lower-level subtype inherits all of the attributes and relationships from all of its upper-level supertypes.

Inheriting the relationships of their supertypes does not mean that subtypes cannot have relationships of their own. Figure 5.4 illustrates a 1:M relationship between EMPLOYEE, a subtype of PERSON, and OFFICE. Because only employees and no other type of person will ever have an office within this system, the relationship is modeled with the subtype directly.

FIGURE 5.4 SPECIALIZATION HIERARCHY WITH OVERLAPPING SUBTYPES



Online Content

For a tutorial on using Visio 2010 to create a specialization hierarchy, see Appendix A, Designing Databases with Visio Professional: A Tutorial, at *www. cengagebrain.com.*

5-1d Subtype Discriminator

A **subtype discriminator** is the attribute in the supertype entity that determines to which subtype the supertype occurrence is related. In Figure 5.2, the subtype discriminator is the employee type (EMP_TYPE).

subtype discriminator
The attribute in the supertype entity that determines to which entity subtype each supertype occurrence is related.

It is common practice to show the subtype discriminator and its value for each subtype in the ERD, as shown in Figure 5.2. However, not all ER modeling tools follow that practice. For example, Microsoft Visio shows the subtype discriminator but not its value. In Figure 5.2, a text tool was used to manually add the discriminator value above the entity subtype, close to the connector line. Using Figure 5.2 as your guide, note that the supertype is related to a PILOT subtype if the EMP_TYPE has a value of "P." If the EMP_TYPE value is "M," the supertype is related to a MECHANIC subtype. If the EMP_TYPE value is "A," the supertype is related to the ACCOUNTANT subtype.

Note that the default comparison condition for the subtype discriminator attribute is the equality comparison. However, in some situations the subtype discriminator is not necessarily based on an equality comparison. For example, based on business requirements, you might create two new pilot subtypes: pilot-in-command (PIC)-qualified and copilot-qualified only. A PIC-qualified pilot must have more than 1,500 PIC flight hours. In this case, the subtype discriminator would be "Flight_Hours," and the criteria would be > 1,500 or <= 1,500, respectively.

Note

In Visio 2010, you select the subtype discriminator when creating a category by using the Category shape from the available shapes. The Category shape is a small circle with a horizontal line underneath that connects the supertype to its subtypes. Newer versions of Visio do not support specialization hierarchy. Professional data modeling tools, such as E/R Studio Data Architect by Idera and Power Designer by Sybase, support specialization hierarchies, but these tools typically cost thousands of dollars for a single user license.

5-1e Disjoint and Overlapping Constraints

An entity supertype can have disjoint or overlapping entity subtypes. In the aviation example, an employee can be a pilot, a mechanic, or an accountant. Assume that one of the business rules dictates that an employee cannot belong to more than one subtype at a time; that is, an employee cannot be a pilot and a mechanic at the same time. **Disjoint subtypes**, also known as **nonoverlapping subtypes**, are subtypes that contain a *unique* subset of the supertype entity set; in other words, each entity instance of the supertype can appear in only one of the subtypes. For example, in Figure 5.2, an employee (supertype) who is a pilot (subtype) can appear only in the PILOT subtype, not in any of the other subtypes. In an ERD, such disjoint subtypes are indicated by the letter *d* inside the category shape.

disjoint subtypes
In a specialization hierarchy, these are unique and nonoverlapping subtype entity set.

nonoverlapping subtypes
See *disjoint subtype.*

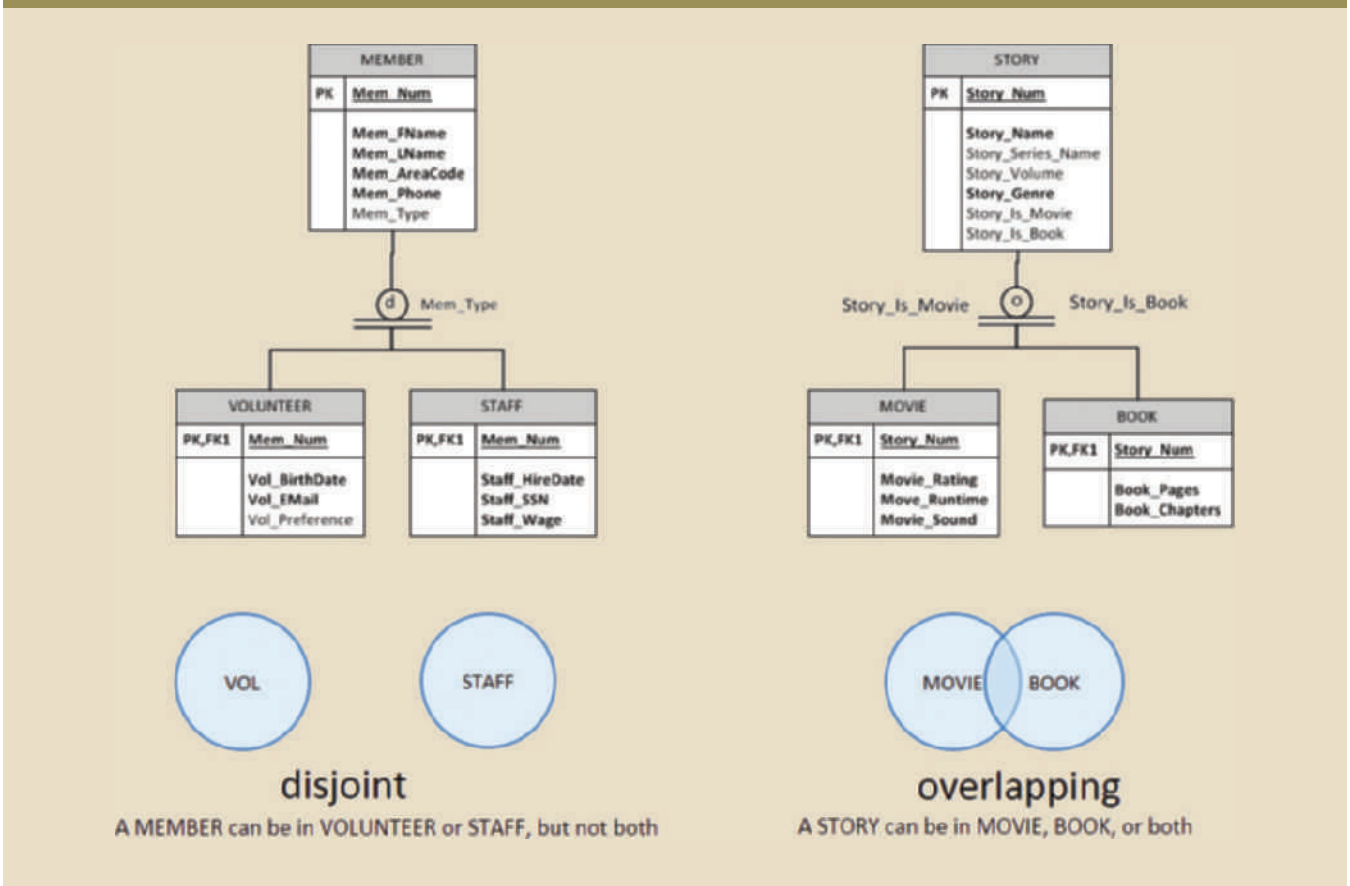
overlapping subtype
In a specialization hierarchy, a condition in which each entity instance (row) of the supertype can appear in more than one subtype.

On the other hand, if the business rule specifies that employees can have multiple classifications, the EMPLOYEE supertype may contain *overlapping* job classification subtypes. **Overlapping subtypes** are subtypes that contain nonunique subsets of the supertype entity set; that is, each entity instance of the supertype may appear in more than one subtype. For example, in a university environment, a person may be an employee, a student, or both. In turn, an employee may be a professor as well as an administrator. Because an employee may also be a student, STUDENT and EMPLOYEE

are overlapping subtypes of the supertype PERSON, just as PROFESSOR and ADMINISTRATOR are overlapping subtypes of the supertype EMPLOYEE. Figure 5.4 illustrates overlapping subtypes with the letter *o* inside the category shape.

Figure 5.5 illustrates more examples of disjoint and overlapping subtypes. At a charity organization, members are either volunteers or they are paid staff. Recall that a subtype must be an identifiable kind or type of instance of the supertype (in this example, identifiable kinds of members), and the subtype must have attributes that are unique to that kind or type of instance. There are attributes that apply to volunteers that do not apply to paid staff, so VOLUNTEER is a valid subtype. There are attributes that apply to paid staff that do not apply to volunteers, so STAFF is also a valid subtype. A single member of the organization cannot be both a volunteer and paid staff, so those subtypes are disjoint. The second example in Figure 5.5 is a website that provides data about popular children's stories and tracks stories that appear in movies and books. Stories in movies have attributes that stories in books do not have, such as a rating and sound type. Stories in books have attributes that do not apply to movies, such as the number of chapters and pages. MOVIE and BOOK are valid subtypes because each is an identifiable kind of story, and each has attributes that do not apply to the other kind of story. However, these subtypes are overlapping because it is possible for a story to appear as both a book and a movie.

FIGURE 5.5 DISJOINT AND OVERLAPPING SUBTYPES



It is common practice to show disjoint and overlapping symbols in the ERD. (See Figures 5.2, 5.4, and 5.5.) However, not all ER modeling tools follow that practice. For example, by default, Visio shows only the subtype discriminator (using the Category shape), but not the disjoint and overlapping symbols. The Visio text tool was used to manually add the *d* and *o* symbols in the previous figures.

Note

Alternative notations exist for representing disjoint and overlapping subtypes. For example, Toby J. Teorey popularized the use of G and Gs to indicate disjoint and overlapping subtypes.

As you learned earlier in this section, the implementation of disjoint subtypes is based on the value of the subtype discriminator attribute in the supertype. However, *implementing* overlapping subtypes requires the use of one discriminator attribute for each subtype. For example, in the case of the Tiny College database design in Chapter 4, Entity Relationship (ER) Modeling, a professor can also be an administrator. Therefore, the EMPLOYEE supertype would have the subtype discriminator attributes and values shown in Table 5.1.

TABLE 5.1

DISCRIMINATOR ATTRIBUTES WITH OVERLAPPING SUBTYPES

DISCRIMINATOR ATTRIBUTES		COMMENT
PROFESSOR	ADMINISTRATOR	
Y	N	The Employee is a member of the Professor subtype.
N	Y	The Employee is a member of the Administrator subtype.
Y	Y	The Employee is both a Professor and an Administrator.

5-1f Completeness Constraint

completeness constraint
A constraint that specifies whether each entity supertype occurrence must also be a member of at least one subtype. The completeness constraint can be partial or total.

partial completeness
In a generalization or specialization hierarchy, a condition in which some supertype occurrences might not be members of any subtype.

total completeness
In a generalization or specialization hierarchy, a condition in which every supertype occurrence must be a member of at least one subtype.

The **completeness constraint** specifies whether each entity supertype occurrence must also be a member of at least one subtype. The completeness constraint can be partial or total. **Partial completeness** means that not every supertype occurrence is a member of a subtype; some supertype occurrences may not be members of any subtype. **Total completeness** means that every supertype occurrence must be a member of at least one subtype.

The ERDs in the previous figures represent the completeness constraint based on the Visio Category shape. A single horizontal line under the circle represents a partial completeness constraint; a double horizontal line under the circle represents a total completeness constraint.

Note

Alternative notations exist to represent the completeness constraint. For example, some notations use a single line (partial) or double line (total) to connect the supertype to the Category shape.

Given the disjoint and overlapping subtypes and completeness constraints, it is possible to have the specialization hierarchy constraint scenarios shown in Table 5.2.

TABLE 5.2

SPECIALIZATION HIERARCHY CONSTRAINT SCENARIOS

TYPE	DISJOINT CONSTRAINT	OVERLAPPING CONSTRAINT
Partial 	Supertype has optional subtypes. Subtype discriminator can be null. Subtype sets are unique.	Supertype has optional subtypes. Subtype discriminators can be null. Subtype sets are not unique.
Total 	Every supertype occurrence is a member of only one subtype. Subtype discriminator cannot be null. Subtype sets are unique.	Every supertype occurrence is a member of at least one subtype. Subtype discriminators cannot be null. Subtype sets are not unique.

5-1g Specialization and Generalization

You can use various approaches to develop entity supertypes and subtypes. For example, you can first identify a regular entity and then identify all entity subtypes based on their distinguishing characteristics. You can also start by identifying multiple entity types and then later extract the common characteristics of those entities to create a higher-level supertype entity.

Specialization is the top-down process of identifying lower-level, more specific entity subtypes from a higher-level entity supertype. Specialization is based on grouping the unique characteristics and relationships of the subtypes. In the aviation example, you used specialization to identify multiple entity subtypes from the original employee supertype. **Generalization** is the bottom-up process of identifying a higher-level, more generic entity supertype from lower-level entity subtypes. Generalization is based on grouping the common characteristics and relationships of the subtypes. For example, you might identify multiple types of musical instruments: piano, violin, and guitar. Using the generalization approach, you could identify a "string instrument" entity supertype to hold the common characteristics of the multiple subtypes.

specialization
In a specialization hierarchy, the grouping of unique attributes into a subtype entity.

generalization
In a specialization hierarchy, the grouping of common attributes into a supertype entity.

5-2 Entity Clustering

Developing an ER diagram entails the discovery of possibly hundreds of entity types and their respective relationships. Generally, the data modeler will develop an initial ERD that contains a few entities. As the design approaches completion, the ERD will contain hundreds of entities and relationships that crowd the diagram to the point of making it unreadable and inefficient as a communication tool. In those cases, you can use entity clusters to minimize the number of entities shown in the ERD.

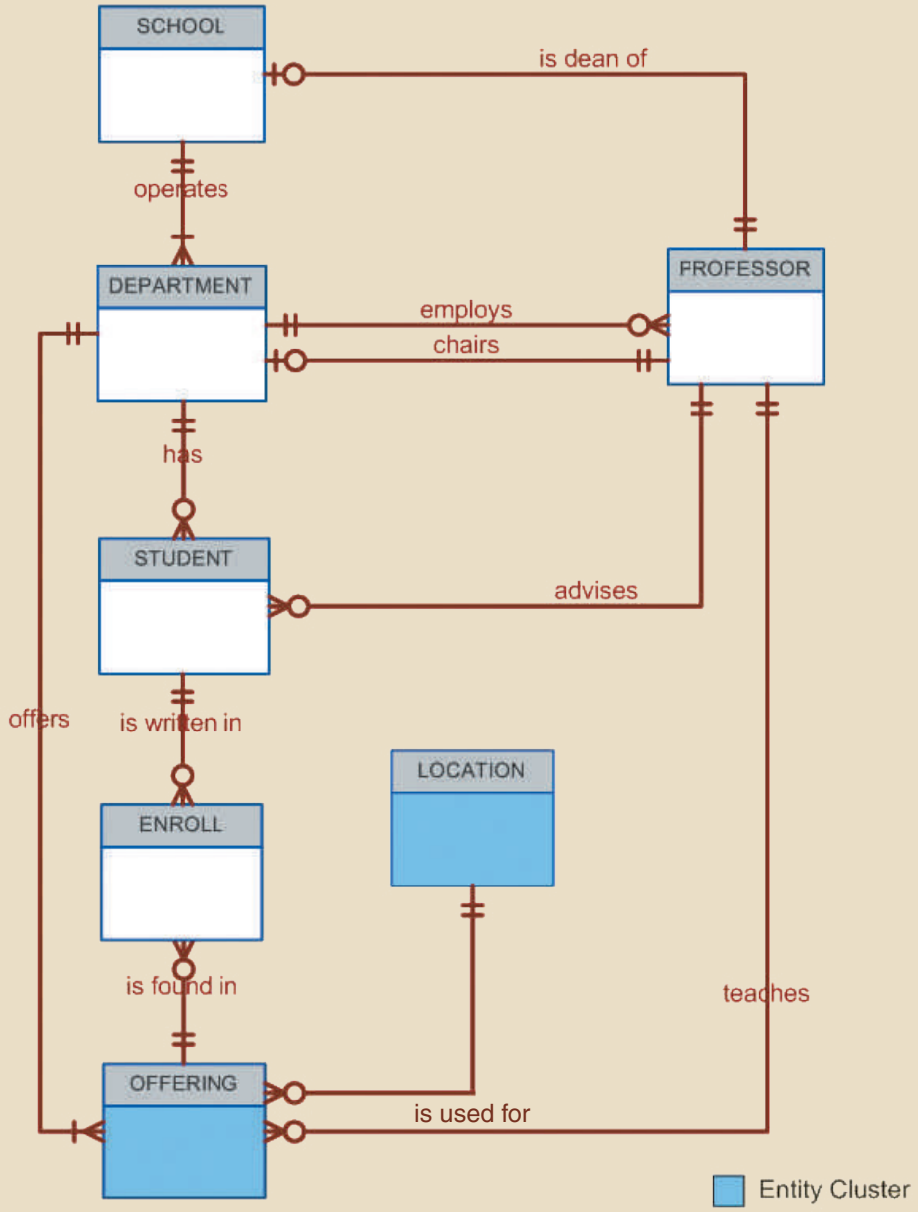
An **entity cluster** is a "virtual" entity type used to represent multiple entities and relationships in the ERD. An entity cluster is formed by combining multiple interrelated entities into a single, abstract entity object. An entity cluster is considered "virtual" or "abstract" in the sense that it is not actually an entity in the final ERD. Instead, it is a temporary entity used to represent multiple entities and relationships, with the purpose of simplifying the ERD and thus enhancing its readability.

Figure 5.6 illustrates the use of entity clusters based on the Tiny College example in Chapter 4. Note that the ERD contains two entity clusters:

- OFFERING, which groups the SEMESTER, COURSE, and CLASS entities and relationships
- LOCATION, which groups the ROOM and BUILDING entities and relationships

entity cluster
A "virtual" entity type used to represent multiple entities and relationships in the ERD. An entity cluster is formed by combining multiple interrelated entities into a single abstract entity object. An entity cluster is considered "virtual" or "abstract" because it is not actually an entity in the final ERD.

FIGURE 5.6 TINY COLLEGE ERD USING ENTITY CLUSTERS



Note also that the ERD in Figure 5.6 does not show attributes for the entities. When using entity clusters, the key attributes of the combined entities are no longer available. Without the key attributes, primary key inheritance rules change. In turn, the change in the inheritance rules can have undesirable consequences, such as changes in relationships—from identifying to nonidentifying or vice versa—and the loss of foreign key attributes from some entities. To eliminate those problems, the general rule is to *avoid the display of attributes when entity clusters are used.*

5-3 Entity Integrity: Selecting Primary Keys

Arguably, the most important characteristic of an entity is its primary key (a single attribute or some combination of attributes), which uniquely identifies each entity instance. The primary key's function is to guarantee entity integrity. Furthermore, primary keys

and foreign keys work together to implement relationships in the relational model. Therefore, the importance of properly selecting the primary key has a direct bearing on the efficiency and effectiveness of database implementation.

5-3a Natural Keys and Primary Keys

The concept of a unique identifier is commonly encountered in the real world. For example, you use class or section numbers to register for classes, invoice numbers to identify specific invoices, and account numbers to identify credit cards. Those examples illustrate natural identifiers or keys. A **natural key** or **natural identifier** is a real-world, generally accepted identifier used to distinguish—that is, uniquely identify—real-world objects. As its name implies, a natural key is familiar to end users and forms part of their day-to-day business vocabulary.

Usually, if an entity *has* a natural identifier, a data modeler uses it as the primary key of the entity being modeled. Generally, most natural keys make acceptable primary key identifiers. The next section presents some basic guidelines for selecting primary keys.

5-3b Primary Key Guidelines

A primary key is the attribute or combination of attributes that uniquely identifies entity instances in an entity set. However, can the primary key be based on, for example, 12 attributes? And just how long can a primary key be? In previous examples, why was EMP_NUM selected as a primary key of EMPLOYEE and not a combination of EMP_LNAME, EMP_FNAME, EMP_INITIAL, and EMP_DOB? Can a single, 256-byte text attribute be a good primary key? There is no single answer to those questions, but database experts have built a body of practice over the years. This section examines that body of documented practices.

First, you should understand the function of a primary key. Its main function is to uniquely identify an entity instance or row within a table. In particular, given a primary key value—that is, the determinant—the relational model can determine the value of all dependent attributes that "describe" the entity. Note that identification and description are separate semantic constructs in the model. *The function of the primary key is to guarantee entity integrity, not to "describe" the entity*.

Second, primary keys and foreign keys are used to implement relationships among entities. However, the implementation of such relationships is done mostly behind the scenes, hidden from end users. In the real world, end users identify objects based on the characteristics they know about the objects. For example, when shopping at a grocery store, you select products by taking them from a display shelf and reading the labels, not by looking at the stock number. It is wise for database applications to mimic the human selection process as much as possible. Therefore, database applications should let the end user choose among multiple descriptive narratives of different objects, while using primary key values behind the scenes. Keeping those concepts in mind, look at Table 5.3, which summarizes desirable primary key characteristics.

5-3c When to Use Composite Primary Keys

In the previous section, you learned about the desirable characteristics of primary keys. For example, you learned that the primary key should use the minimum number of attributes possible. However, that does *not* mean that composite primary keys are not permitted in a model. In fact, composite primary keys are particularly useful in two cases:

- As identifiers of composite entities, in which each primary key combination is allowed only once in the M:N relationship
- As identifiers of weak entities, in which the weak entity has a strong identifying relationship with the parent entity

natural key (natural identifier)
A generally accepted identifier for real-world objects. As its name implies, a natural key is familiar to end users and forms part of their day-to-day business vocabulary.

TABLE 5.3

DESIRABLE PRIMARY KEY CHARACTERISTICS

PK CHARACTERISTIC	RATIONALE
Unique values	The PK must uniquely identify each entity instance. A primary key must be able to guarantee unique values. It cannot contain nulls.
Nonintelligent	The PK should not have embedded semantic meaning other than to uniquely identify each entity instance. An attribute with embedded semantic meaning is probably better used as a descriptive characteristic of the entity than as an identifier. For example, a student ID of 650973 would be preferred over Smith, Martha L. as a primary key identifier.
No change over time	If an attribute has semantic meaning, it might be subject to updates, which is why names do not make good primary keys. If Vickie Smith is the primary key, what happens if she changes her name when she gets married? If a primary key is subject to change, the foreign key values must be updated, thus adding to the database work load. Furthermore, changing a primary key value means that you are basically changing the identity of an entity. In short, the PK should be permanent and unchangeable.
Preferably single-attribute	A primary key should have the minimum number of attributes possible (irreducible). Single-attribute primary keys are desirable but not required. Single-attribute primary keys simplify the implementation of foreign keys. Having multiple-attribute primary keys can cause primary keys of related entities to grow through the possible addition of many attributes, thus adding to the database workload and making (application) coding more cumbersome.
Preferably numeric	Unique values can be better managed when they are numeric, because the database can use internal routines to implement a counter-style attribute that automatically increments values with the addition of each new row. In fact, most database systems include the ability to use special constructs, such as Autonumber in Microsoft Access, sequence in Oracle, or uniqueidentifier in MS SQL Server to support self-incrementing primary key attributes.
Security-compliant	The selected primary key must not be composed of any attribute(s) that might be considered a security risk or violation. For example, using a Social Security number as a PK in an EMPLOYEE table is not a good idea.

To illustrate the first case, assume that you have a STUDENT entity set and a CLASS entity set. In addition, assume that those two sets are related in an M:N relationship via an ENROLL entity set, in which each student or class combination may appear only once in the composite entity. Figure 5.7 shows the ERD to represent such a relationship.

As shown in Figure 5.7, the composite primary key automatically provides the benefit of ensuring that there cannot be duplicate values—that is, it ensures that the same student cannot enroll more than once in the same class.

In the second case, a weak entity in a strong identifying relationship with a parent entity is normally used to represent one of two situations:

1. *A real-world object that is existence-dependent on another real-world object*. Such objects are distinguishable in the real world. A dependent and an employee are two separate people who exist independently of each other. However, such objects can exist in the model only when they relate to each other in a strong identifying relationship. For example, the relationship between EMPLOYEE and DEPENDENT is one of existence dependency, in which the primary key of the dependent entity is a composite key that contains the key of the parent entity.
2. *A real-world object that is represented in the data model as two separate entities in a strong identifying relationship*. For example, the real-world invoice object is represented by two entities in a data model: INVOICE and LINE. Clearly, the LINE entity does not exist in the real world as an independent object but as part of an INVOICE.

FIGURE 5.7 THE M:N RELATIONSHIP BETWEEN STUDENT AND CLASS

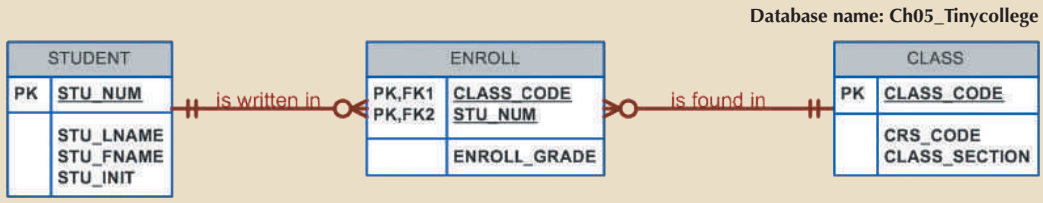


Table name: STUDENT
(first four fields)

STU_NUM	STU_LNAME	STU_FNAME	STU_INIT
321452	Bowser	William	C
324257	Smithson	Anne	K
324258	Brewer	Juliette	
324269	Oblonski	Walter	H
324273	Smith	John	D
324274	Katinga	Raphael	P
324291	Robertson	Gerald	T
324299	Smith	John	B

Table name: ENROLL

CLASS_CODE	STU_NUM	ENROLL_GRADE
10014	321452	C
10014	324257	B
10018	321452	A
10018	324257	B
10021	321452	C
10021	324257	C

Table name: CLASS
(first three fields)

CLASS_CODE	CRS_CODE	CLASS_SECTION
10012	ACCT-211	1
10013	ACCT-211	2
10014	ACCT-211	3
10015	ACCT-212	1
10016	ACCT-212	2
10017	CIS-220	1
10018	CIS-220	2
10019	CIS-220	3
10020	CIS-420	1
10021	QM-261	1
10022	QM-261	2
10023	QM-362	1
10024	QM-362	2
10025	MATH-243	1

In both situations, having a strong identifying relationship ensures that the dependent entity can exist only when it is related to the parent entity. In summary, the selection of a composite primary key for composite and weak entity types provides benefits that enhance the integrity and consistency of the model.

5-3d When to Use Surrogate Primary Keys

In some instances a primary key doesn't exist in the real world or the existing natural key might not be a suitable primary key. In these cases, it is standard practice to create a surrogate key. A **surrogate key** is a primary key created by the database designer to simplify the identification of entity instances. The surrogate key has no meaning in the user's environment—it exists only to distinguish one entity instance from another (just like any other primary key). One practical advantage of a surrogate key is that because it has no intrinsic meaning, values for it can be generated by the DBMS to ensure that unique values are always provided.

surrogate key
A system-assigned primary key, generally numeric and auto-incremented.

For example, consider the case of a park recreation facility that rents rooms for small parties. The manager of the facility keeps track of all events, using a folder with the format shown in Table 5.4.

TABLE 5.4

DATA USED TO KEEP TRACK OF EVENTS

DATE	TIME_START	TIME_END	ROOM	EVENT_NAME	PARTY_OF
6/17/2018	11:00 a.m.	2:00 p.m.	Allure	Burton Wedding	60
6/17/2018	11:00 a.m.	2:00 p.m.	Bonanza	Adams Office	12
6/17/2018	3:00 p.m.	5:30 p.m.	Allure	Smith Family	15
6/17/2018	3:30 p.m.	5:30 p.m.	Bonanza	Adams Office	12
6/18/2018	1:00 p.m.	3:00 p.m.	Bonanza	Boy Scouts	33
6/18/2018	11:00 a.m.	2:00 p.m.	Allure	March of Dimes	25
6/18/2018	11:00 a.m.	12:30 p.m.	Bonanza	Smith Family	12

Given the data shown in Table 5.4, you would model the EVENT entity as follows:

EVENT (DATE, TIME_START, TIME_END, ROOM, EVENT_NAME, PARTY_OF)

What primary key would you suggest? In this case, there is no simple natural key that could be used as a primary key in the model. Based on the primary key concepts you learned in previous chapters, you might suggest one of these options:

(**DATE**, **TIME_START**, **ROOM**) or (**DATE**, **TIME_END**, **ROOM**)

Assume that you select the composite primary key (**DATE**, **TIME_START**, **ROOM**) for the EVENT entity. Next, you determine that one EVENT may use many RESOURCEs (such as tables, projectors, PCs, and stands) and that the same RESOURCE may be used for many EVENTs. The RESOURCE entity would be represented by the following attributes:

RESOURCE (**RSC_ID**, RSC_DESCRIPTION, RSC_TYPE, RSC_QTY, RSC_PRICE)

Given the business rules, the M:N relationship between RESOURCE and EVENT would be represented via the EVNTRSC composite entity with a composite primary key as follows:

EVNTRSC (**DATE**, **TIME_START**, **ROOM**, **RSC_ID**, QTY_USED)

You now have a lengthy, four-attribute composite primary key. What would happen if the EVNTRSC entity's primary key were inherited by another existence-dependent entity? At this point, you can see that the composite primary key could make the database implementation and program coding unnecessarily complex.

As a data modeler, you probably noticed that the EVENT entity's selected primary key might not fare well, given the primary key guidelines in Table 5.3. In this case, the EVENT entity's selected primary key contains embedded semantic information and is formed by a combination of date, time, and text data columns. In addition, the selected primary key would cause lengthy primary keys for existence-dependent entities. The preferred alternative is to use a numeric, single-attribute surrogate primary key.

Surrogate primary keys are accepted practice in today's complex data environments. They are especially helpful when there is no natural key, when the selected candidate key has embedded semantic contents, or when the selected candidate key is too long or cumbersome. However, there is a trade-off: if you use a surrogate key, you must ensure that the candidate key of the entity in question performs properly through the use of "unique index" and "not null" constraints.

Note

This example shows a case in which entity integrity is maintained but semantic correctness of business rules is not. For example, you could have two events that overlap and whose primary keys are perfectly compliant. The only way to ensure adherence to this type of business rule (two events cannot overlap—occur on the same room at the same time) would be via application programming code.

5-4 Design Cases: Learning Flexible Database Design

Data modeling and database design require skills that are acquired through experience. In turn, experience is acquired through practice—regular and frequent repetition, applying the concepts learned to specific and different design problems. This section presents four special design cases that highlight the importance of flexible designs, proper identification of primary keys, and placement of foreign keys.

Note

In describing the various modeling concepts throughout this book, the focus is on relational models. Also, given the focus on the practical nature of database design, all design issues are addressed with the implementation goal in mind. Therefore, there is no sharp line of demarcation between design and implementation.

At the pure conceptual stage of the design, foreign keys are not part of an ERD. The ERD displays only entities and relationships. Entity instances are distinguished by identifiers that may become primary keys. During design, the modeler attempts to understand and define the entities and relationships. Foreign keys are the mechanism through which the relationship designed in an ERD is implemented in a relational model.

5-4a Design Case 1: Implementing 1:1 Relationships

Foreign keys work with primary keys to properly implement relationships in the relational model. The basic rule is very simple: put the primary key of the "one" side (the parent entity) on the "many" side (the dependent entity) as a foreign key. However, where do you place the foreign key when you are working with a 1:1 relationship? For example, take the case of a 1:1 relationship between EMPLOYEE and DEPARTMENT based on the business rule "one EMPLOYEE is the manager of one DEPARTMENT, and one DEPARTMENT is managed by one EMPLOYEE." In that case, there are two options for selecting and placing the foreign key:

1. *Place a foreign key in both entities*. This option is derived from the basic rule you learned in Chapter 4. Place EMP_NUM as a foreign key in DEPARTMENT, and place DEPT_ID as a foreign key in EMPLOYEE. However, this solution is not recommended because it duplicates work, and it could conflict with other existing relationships. (Remember that DEPARTMENT and EMPLOYEE also participate in a 1:M relationship—one department employs many employees.)
2. *Place a foreign key in one of the entities*. In that case, the primary key of one of the two entities appears as a foreign key in the other entity. That is the preferred solution, but a question remains: *which* primary key should be used as a foreign key? The answer is found in Table 5.5, which shows the rationale for selecting the foreign key in a 1:1 relationship based on the relationship properties in the ERD.

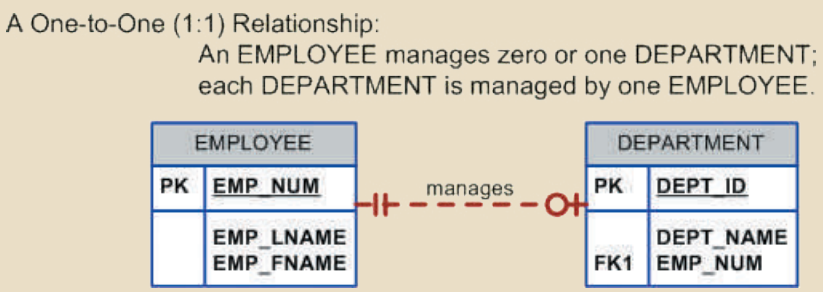
TABLE 5.5

SELECTION OF FOREIGN KEY IN A 1:1 RELATIONSHIP

CASE	ER RELATIONSHIP CONSTRAINTS	ACTION
I	One side is mandatory and the other side	Place the PK of the entity on the mandatory side in the entity on the optional side as a FK, and make the FK mandatory.
II	Both sides are optional.	Select the FK that causes the fewest nulls, or place the FK in the entity in which the (relationship) role is played.
III	Both sides are mandatory.	See Case II, or consider revising your model to ensure that the two entities do not belong together in a single entity.

Figure 5.8 illustrates the "EMPLOYEE manages DEPARTMENT" relationship. Note that in this case, EMPLOYEE is mandatory to DEPARTMENT. Therefore, EMP_NUM is placed as the foreign key in DEPARTMENT. Alternatively, you might also argue that the "manager" role is played by the EMPLOYEE in the DEPARTMENT.

FIGURE 5.8 THE 1:1 RELATIONSHIP BETWEEN DEPARTMENT AND EMPLOYEE



As a designer, you must recognize that 1:1 relationships exist in the real world; therefore, they should be supported in the data model. In fact, a 1:1 relationship is used to ensure that two entity sets are not placed in the same table. In other words, EMPLOYEE and DEPARTMENT are clearly separate and unique entity types that do not belong together in a single entity. If you grouped them together in one entity, what would you name that entity?

5-4b Design Case 2: Maintaining History of Time-Variant Data

Company managers generally realize that good decision making is based on the information generated through the data stored in databases. Such data reflects both current and past events. Company managers use the data stored in databases to answer questions such as "How do the current company profits compare to those of previous years?" and "What are XYZ product's sales trends?" In other words, the data stored in databases reflects not only current data but also historic data.

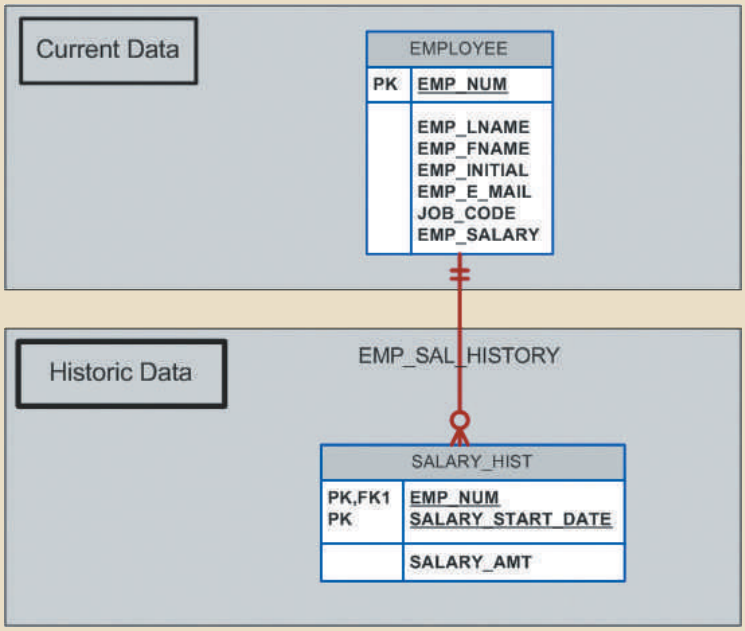
Normally, data changes are managed by replacing the existing attribute value with the new value, without regard to the previous value. However, in some situations the history of values for a given attribute must be preserved. From a data-modeling point of view, **time-variant data** refers to data whose values change over time and for which you *must* keep a history of the data changes. You could argue that all data in a database is subject to change over time and is therefore time variant. However, some attribute values, such as your date of birth or your Social Security number, are not time variant. On the other hand, attributes such as your student GPA or your bank account balance are subject to change over time. Sometimes the data changes are externally originated and event driven, such as a product price change. On other occasions, changes are based on well-defined schedules, such as the daily stock quote "open" and "close" values.

The storage of time-variant data requires changes in the data model; the type of change depends on the nature of the data. Some time-variant data is equivalent to having a multivalued attribute in your entity. To model this type of time-variant data, you must create a new entity in a 1:M relationship with the original entity. This new entity will contain the new value, the date of the change, and any other attribute that is pertinent to the event being modeled. For example, if you want to track salary histories for each employee, then the EMP_SALARY attribute becomes multivalued, as shown in Figure 5.9. In this case, for each employee, there will be one or more records in the SALARY_HIST entity, which stores the salary amount and the date when the new salary goes into effect.

time-variant data
Data whose values are a function of time. For example, time-variant data can be seen at work when a company's history of all administrative appointments is tracked.

Other time-variant data can turn a 1:M relationship into an M:N relationship. Assume that in addition to employee data, your data model includes data about the different departments in the organization and which employee manages each department. Assuming that each department is managed by only one employee and each employee can manage one department at most, then a 1:1 relationship would exist between EMPLOYEE and DEPARTMENT. This relationship would record the current manager

FIGURE 5.9 MAINTAINING SALARY HISTORY



of each department. However, if you want to keep track of the history of all department managers as well as the current manager, you can create the model shown in Figure 5.10.

Note that in Figure 5.10, the MGR_HIST entity has a 1:M relationship with EMPLOYEE and a 1:M relationship with DEPARTMENT to reflect the fact that an employee could be the manager of many different departments over time, and a department could have many different employee managers. Because you are recording time-variant data, you must store the DATE_ASSIGN attribute in the MGR_HIST entity to provide the date that the employee (EMP_NUM) became the department manager. The primary key of MGR_HIST permits the same employee to be the manager of the same department but on different dates. If that scenario is not the case in your environment—if, for example, an employee is the manager of a department only once—you could make DATE_ASSIGN a nonprime attribute in the MGR_HIST entity.

Note in Figure 5.10 that the "manages" relationship is optional in theory and redundant in practice. At any time, you could identify the manager of a department by retrieving the most recent DATE_ASSIGN date from MGR_HIST for a given department. On the other hand, the ERD in Figure 5.10 differentiates between current data and historic data. The *current* manager relationship is implemented by the "manages" relationship between EMPLOYEE and DEPARTMENT. Additionally, the historic data is managed through EMP_MGR_HIST and DEPT_MGR_HIST. The trade-off with that model is that each time a new manager is assigned to a department, there will be two data modifications: one update in the DEPARTMENT entity and one insert in the MGR_HIST entity.

The flexibility of the model proposed in Figure 5.10 becomes more apparent when you add the 1:M "one department employs many employees" relationship. In that case, the PK of the "1" side (DEPT_ID) appears in the "many" side (EMPLOYEE) as a foreign key. Now suppose you would like to keep track of the job history for each of the company's employees—you'd probably want to store the department, the job code, the date assigned, and the salary. To accomplish that task, you could modify the model in Figure 5.10 by adding a JOB_HIST entity. Figure 5.11 shows the use of the new JOB_HIST entity to maintain the employee's history.

FIGURE 5.10 MAINTAINING MANAGER HISTORY

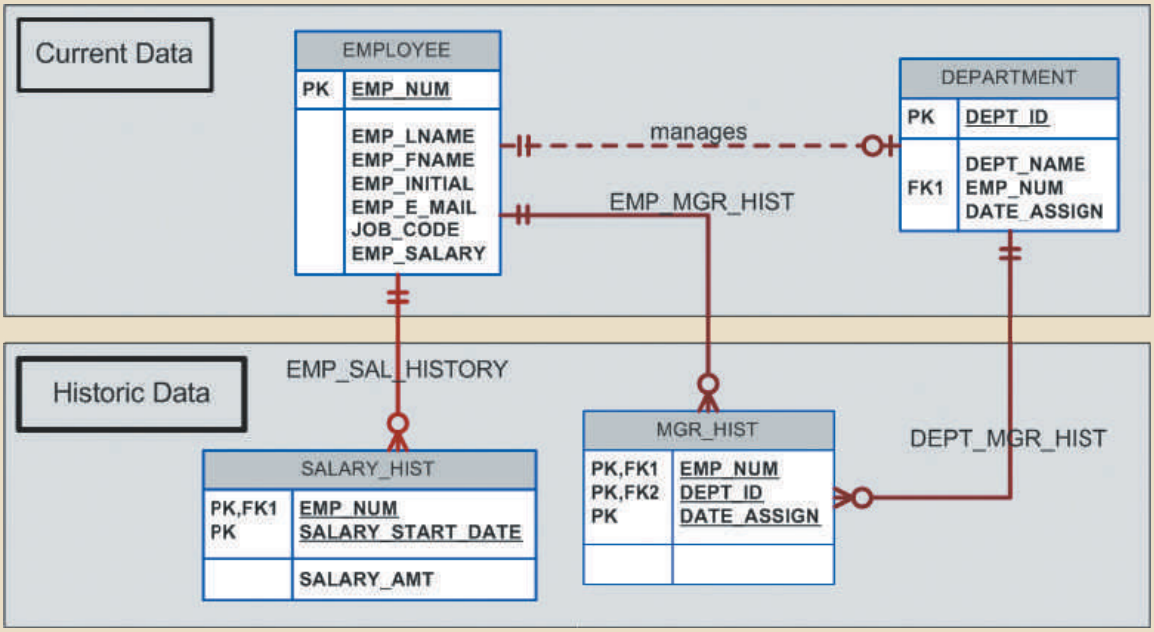
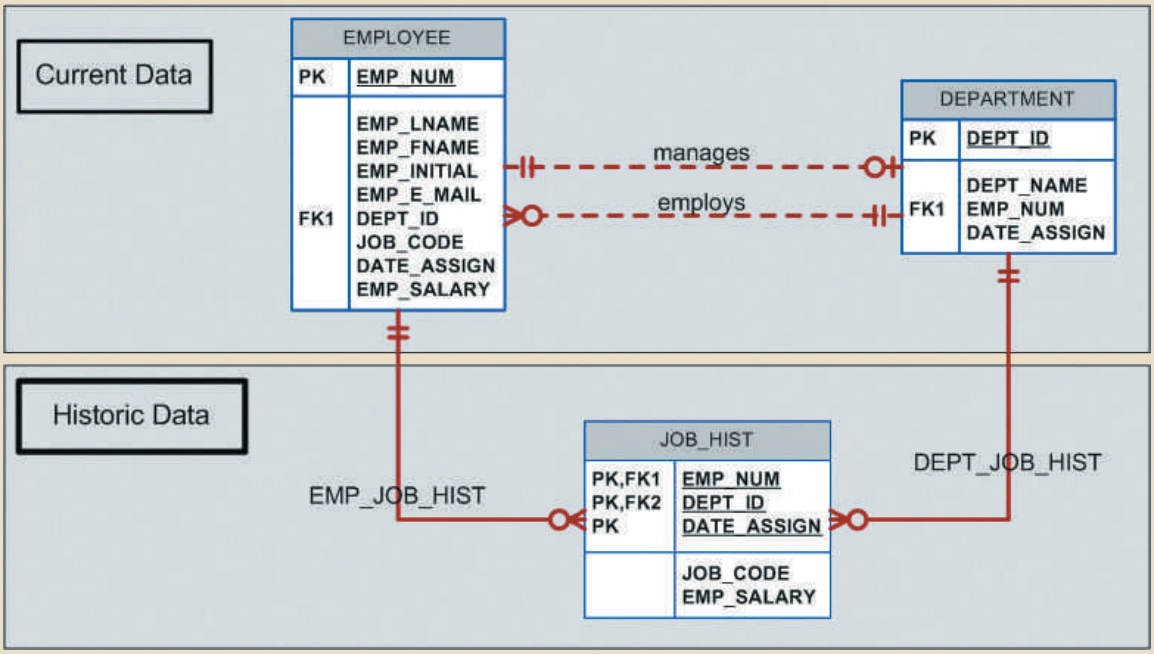


FIGURE 5.11 MAINTAINING JOB HISTORY



Again, it is worth emphasizing that the "manages" and "employs" relationships are theoretically optional and redundant in practice. You can always find out where each employee works by looking at the job history and selecting only the most current data row for each employee. However, as you will discover in Chapter 7, Introduction to Structured Query Language (SQL), and in Chapter 8, Advanced SQL, finding where each employee works is not a trivial task. Therefore, the model represented in Figure 5.11

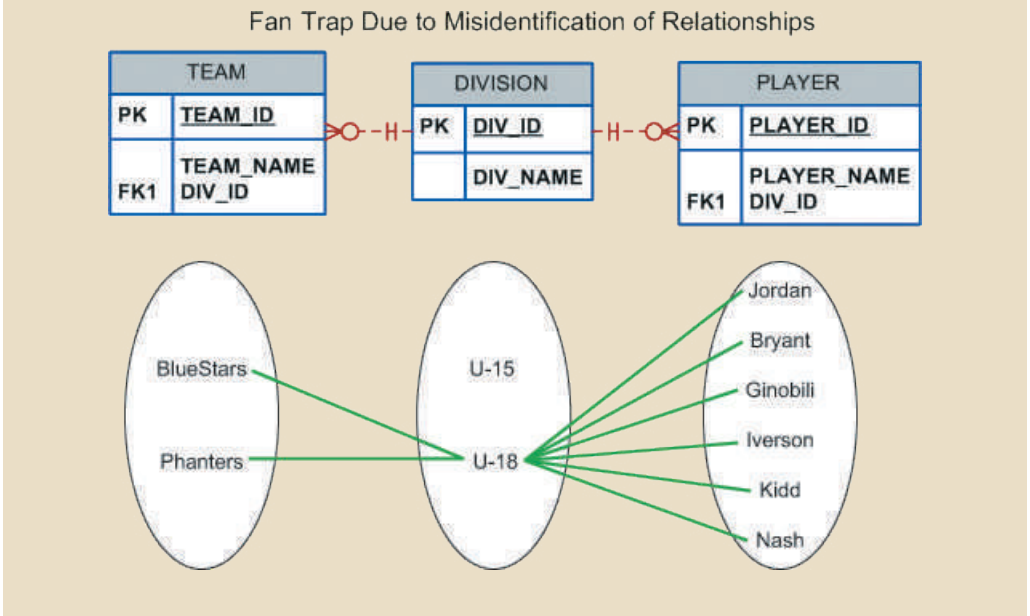
includes the admittedly redundant but unquestionably useful "manages" and "employs" relationships to separate current data from historic data.

5-4c Design Case 3: Fan Traps

Creating a data model requires proper identification of the data relationships among entities. However, due to miscommunication or incomplete understanding of the business rules or processes, it is not uncommon to misidentify relationships among entities. Under those circumstances, the ERD may contain a design trap. A **design trap** occurs when a relationship is improperly or incompletely identified and is therefore represented in a way that is not consistent with the real world. The most common design trap is known as a *fan trap*.

A **fan trap** occurs when you have one entity in two 1:M relationships to other entities, thus producing an association among the other entities that is not expressed in the model. For example, assume that the JCB basketball league has many divisions. Each division has many players, and each division has many teams. Given those "incomplete" business rules, you might create an ERD that looks like the one in Figure 5.12.

FIGURE 5.12 INCORRECT ERD WITH FAN TRAP PROBLEM



As you can see in Figure 5.12, DIVISION is in a 1:M relationship with TEAM and in a 1:M relationship with PLAYER. Although that representation is semantically correct, the relationships are not properly identified. For example, there is no way to identify which players belong to which team. Figure 5.12 also shows a sample instance relationship representation for the ERD. Note that the relationship lines for the DIVISION instances fan out to the TEAM and PLAYER entity instances—thus the "fan trap" label.

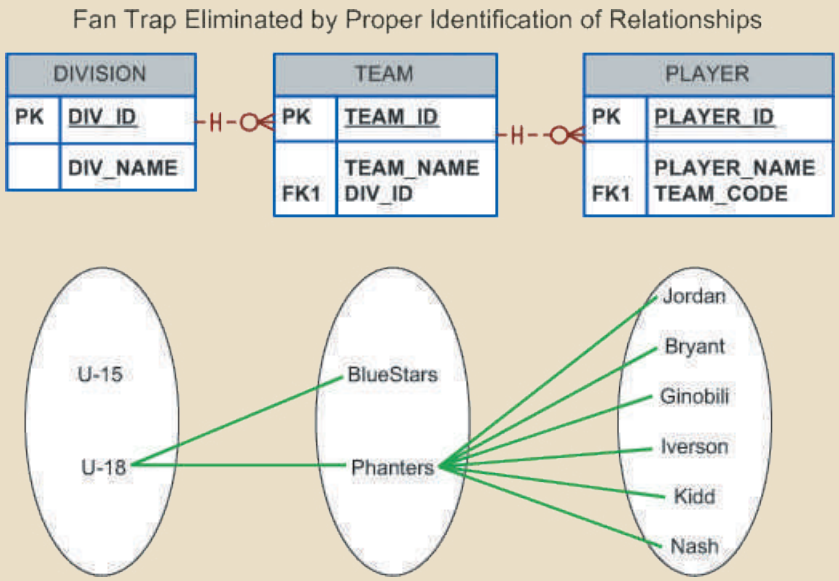
Figure 5.13 shows the correct ERD after the fan trap has been eliminated. Note that, in this case, DIVISION is in a 1:M relationship with TEAM. In turn, TEAM is in a 1:M relationship with PLAYER. Figure 5.13 also shows the instance relationship representation after eliminating the fan trap.

Given the design in Figure 5.13, note how easy it is to see which players play for which team. However, to find out which players play in which division, you first need to see what teams belong to each division, and then you need to find out which players play on each team. In other words, there is a transitive relationship between DIVISION and PLAYER via the TEAM entity.

design trap
A problem that occurs when a relationship is improperly or incompletely identified and therefore is represented in a way that is not consistent with the real world. The most common design trap is known as a *fan trap*.

fan trap
A design trap that occurs when one entity is in two 1:M relationships with other entities, thus producing an association among the other entities that is not expressed in the model.

FIGURE 5.13 CORRECTED ERD AFTER REMOVAL OF THE FAN TRAP



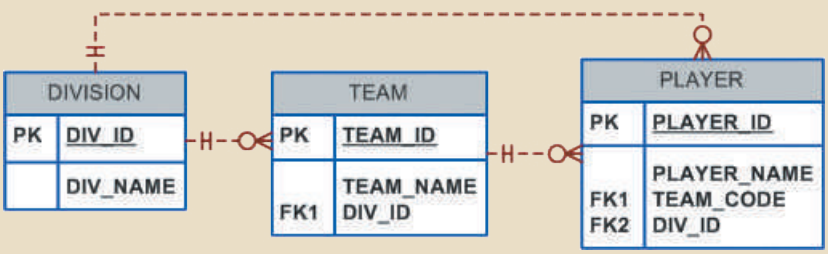
5-4d Design Case 4: Redundant Relationships

Although redundancy is often good to have in computer environments (multiple back-ups in multiple places, for example), redundancy is seldom good in the database environment. (As you learned in Chapter 3, The Relational Database Model, redundancies can cause data anomalies in a database.) Redundant relationships occur when there are multiple relationship paths between related entities. The main concern with redundant relationships is that they remain consistent across the model. However, it is important to note that some designs use redundant relationships as a way to simplify the design.

An example of redundant relationships was first introduced in Figure 5.10 during the discussion of maintaining a history of time-variant data. However, the use of the redundant "manages" and "employs" relationships was justified by the fact that such relationships dealt with current data rather than historic data. Another more specific example of a redundant relationship is represented in Figure 5.14.

In Figure 5.14, note the transitive 1:M relationship between DIVISION and PLAYER through the TEAM entity set. Therefore, the relationship that connects DIVISION and PLAYER is redundant, for all practical purposes. In that case, the relationship could be safely deleted without losing any information-generation capabilities in the model.

FIGURE 5.14 A REDUNDANT RELATIONSHIP



Summary

- The extended entity relationship (EER) model adds semantics to the ER model via entity supertypes, subtypes, and clusters. An entity supertype is a generic entity type that is related to one or more entity subtypes.
- A specialization hierarchy depicts the arrangement and relationships between entity supertypes and entity subtypes. Inheritance means that an entity subtype inherits the attributes and relationships of the supertype. Subtypes can be disjoint or overlapping. A subtype discriminator is used to determine to which entity subtype the supertype occurrence is related. The subtypes can exhibit partial or total completeness. There are basically two approaches to developing a specialization hierarchy of entity supertypes and subtypes: specialization and generalization.
- An entity cluster is a "virtual" entity type used to represent multiple entities and relationships in the ERD. An entity cluster is formed by combining multiple interrelated entities and relationships into a single, abstract entity object.
- Natural keys are identifiers that exist in the real world. Natural keys sometimes make good primary keys, but not always. Primary keys must have unique values, they should be nonintelligent, they must not change over time, and they are preferably numeric and composed of a single attribute.
- Composite keys are useful to represent M:N relationships and weak (strong identifying) entities.
- Surrogate primary keys are useful when there is no natural key that makes a suitable primary key, when the primary key is a composite primary key with multiple data types, or when the primary key is too long to be usable.
- In a 1:1 relationship, place the PK of the mandatory entity as a foreign key in the optional entity, as an FK in the entity that causes the fewest nulls, or as an FK where the role is played.
- Time-variant data refers to data whose values change over time and require that you keep a history of data changes. To maintain the history of time-variant data, you must create an entity that contains the new value, the date of change, and any other time-relevant data. This entity maintains a 1:M relationship with the entity for which the history is to be maintained.
- A fan trap occurs when you have one entity in two 1:M relationships to other entities, and there is an association among the other entities that is not expressed in the model. Redundant relationships occur when there are multiple relationship paths between related entities. The main concern with redundant relationships is that they remain consistent across the model.

Key Terms

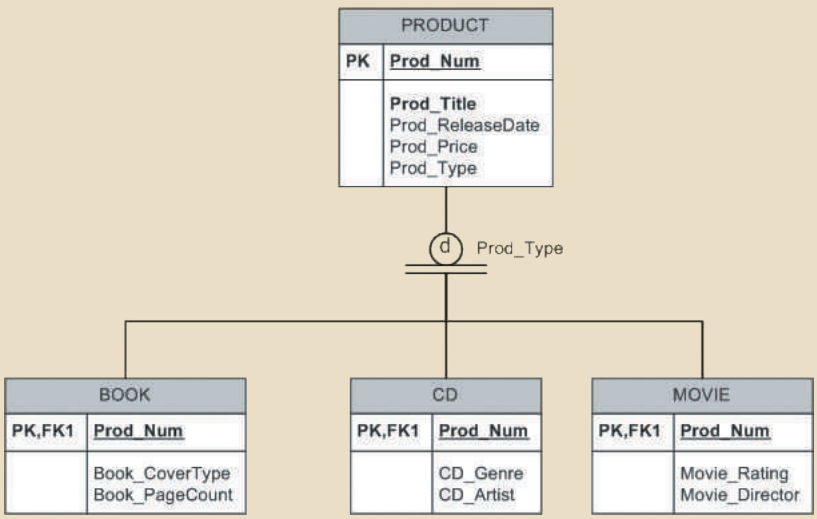
completeness constraint
design trap
disjoint subtype
EER diagram (EERD)
entity cluster
entity subtype
entity supertype
extended entity relationship model (EERM)
fan trap
generalization
inheritance
natural key (natural identifier)
nonoverlapping subtype
overlapping subtype
partial completeness
specialization
specialization hierarchy
subtype discriminator
surrogate key
time-variant data
total completeness

Review Questions

1. What is an entity supertype, and why is it used?
2. What kinds of data would you store in an entity subtype?
3. What is a specialization hierarchy?
4. What is a subtype discriminator? Give an example of its use.
5. What is an overlapping subtype? Give an example.
6. What is a disjoint subtype? Give an example.
7. What is the difference between partial completeness and total completeness?

For Questions 8–10, refer to Figure Q5.8.

FIGURE Q5.8 THE PRODUCT DATA MODEL



8. List all of the attributes of a movie.
9. According to the data model, is it required that every entity instance in the PRODUCT table be associated with an entity instance in the CD table? Why, or why not?
10. Is it possible for a book to appear in the BOOK table without appearing in the PRODUCT table? Why, or why not?
11. What is an entity cluster, and what advantages are derived from its use?
12. What primary key characteristics are considered desirable? Explain *why* each characteristic is considered desirable.
13. Under what circumstances are composite primary keys appropriate?
14. What is a surrogate primary key, and when would you use one?
15. When implementing a 1:1 relationship, where should you place the foreign key if one side is mandatory and one side is optional? Should the foreign key be mandatory or optional?

16. What is time-variant data, and how would you deal with such data from a database design point of view?
17. What is the most common design trap, and how does it occur?

Problems

1. Given the following business scenario, create a Crow's Foot ERD using a specialization hierarchy if appropriate. Two-Bit Drilling Company keeps information on employees and their insurance dependents. Each employee has an employee number, name, date of hire, and title. If an employee is an inspector, then the date of certification and certification renewal date should also be recorded in the system. For all employees, the Social Security number and dependent names should be kept. All dependents must be associated with one and only one employee. Some employees will not have dependents, while others will have many dependents.
2. Given the following business scenario, create a Crow's Foot ERD using a specialization hierarchy if appropriate. Tiny Hospital keeps information on patients and hospital rooms. The system assigns each patient a patient ID number. In addition, the patient's name and date of birth are recorded. Some patients are resident patients who spend at least one night in the hospital, and others are outpatients who are treated and released. Resident patients are assigned to a room. Each room is identified by a room number. The system also stores the room type (private or semiprivate) and room fee. Over time, each room will have many patients. Each resident patient will stay in only one room. Every room must have had a patient, and every resident patient must have a room.
3. Given the following business scenario, create a Crow's Foot ERD using a specialization hierarchy if appropriate. Granite Sales Company keeps information on employees and the departments in which they work. For each department, the department name, internal mail box number, and office phone extension are kept. A department can have many assigned employees, and each employee is assigned to only one department. Employees can be salaried, hourly, or work on contract. All employees are assigned an employee number, which is kept along with the employee's name and address. For hourly employees, hourly wages and target weekly work hours are stored; for example, the company may target 40 hours/week for some employees, 32 for others, and 20 for others. Some salaried employees are salespeople who can earn a commission in addition to their base salary. For all salaried employees, the yearly salary amount is recorded in the system. For salespeople, their commission percentage on sales and commission percentage on profit are stored in the system. For example, John is a salesperson with a base salary of $50,000 per year plus a 2 percent commission on the sales price for all sales he makes, plus another 5 percent of the profit on each of those sales. For contract employees, the beginning date and end date of their contracts are stored along with the billing rate for their hours.
4. In Chapter 4, you saw the creation of the Tiny College database design, which reflected such business rules as "a professor may advise many students" and "a professor may chair one department." Modify the design shown in Figure 4.35 to include these business rules:
   - An employee could be staff, a professor, or an administrator.
   - A professor may also be an administrator.

- Staff employees have a work-level classification, such as Level I or Level II.
- Only professors can chair a department. A department is chaired by only one professor.
- Only professors can serve as the dean of a college. Each of the university's colleges is served by one dean.
- A professor can teach many classes.
- Administrators have a position title.

Given that information, create the complete ERD that contains all primary keys, foreign keys, and main attributes.

5. Tiny College wants to keep track of the history of all its administrative appointments, including dates of appointment and dates of termination. (*Hint:* Time-variant data is at work.) The Tiny College chancellor may want to know how many deans worked in the College of Business between January 1, 1960, and January 1, 2018, or who the dean of the College of Education was in 1990. Given that information, create the complete ERD that contains all primary keys, foreign keys, and main attributes.
6. Some Tiny College staff employees are information technology (IT) personnel. Some IT personnel provide technology support for academic programs, some provide technology infrastructure support, and some provide support for both. IT personnel are not professors; they are required to take periodic training to retain their technical expertise. Tiny College tracks all IT personnel training by date, type, and results (completed versus not completed). Given that information, create the complete ERD that contains all primary keys, foreign keys, and main attributes.
7. The FlyRight Aircraft Maintenance (FRAM) division of the FlyRight Company (FRC) performs all maintenance for FRC's aircraft. Produce a data model segment that reflects the following business rules:

   - All mechanics are FRC employees. Not all employees are mechanics.
   - Some mechanics are specialized in engine (EN) maintenance. Others are specialized in airframe (AF) maintenance or avionics (AV) maintenance. (Avionics are the electronic components of an aircraft that are used in communication and navigation.) All mechanics take periodic refresher courses to stay current in their areas of expertise. FRC tracks all courses taken by each mechanic—date, course type, certification (Y/N), and performance.
   - FRC keeps an employment history of all mechanics. The history includes the date hired, date promoted, and date terminated.

   Given those requirements, create the Crow's Foot ERD segment.

Cases

8. "Martial Arts R Us" (MARU) needs a database. MARU is a martial arts school with hundreds of students. The database must keep track of all the classes that are offered, who is assigned to teach each class, and which students attend each class. Also, it is important to track the progress of each student as they advance. Create a complete Crow's Foot ERD for these requirements:

   - Students are given a student number when they join the school. The number is stored along with their name, date of birth, and the date they joined the school.

- All instructors are also students, but clearly not all students are instructors. In addition to the normal student information, for all instructors, the date that they start working as an instructor must be recorded along with their instructor status (compensated or volunteer).
- An instructor may be assigned to teach any number of classes, but each class has one and only one assigned instructor. Some instructors, especially volunteer instructors, may not be assigned to any class.
- A class is offered for a specific level at a specific time, day of the week, and location. For example, one class taught on Mondays at 5:00 p.m. in Room 1 is an intermediate-level class. Another class taught on Mondays at 6:00 p.m. in Room 1 is a beginner-level class. A third class taught on Tuesdays at 5:00 p.m. in Room 2 is an advanced-level class.
- Students may attend any class of the appropriate level during each week, so there is no expectation that any particular student will attend any particular class session. Therefore, the attendance of students at each individual class meeting must be tracked.
- A student will attend many different class meetings, and each class meeting is normally attended by many students. Some class meetings may not be attended by any students. New students may not have attended any class meetings yet.
- At any given meeting of a class, instructors other than the assigned instructor may show up to help. Therefore, a given class meeting may have a head instructor and many assistant instructors, but it will always have at least the one instructor who is assigned to that class. For each class meeting, the date of the class and the instructors' roles (head instructor or assistant instructor) need to be recorded. For example, Mr. Jones is assigned to teach the Monday, 5:00 p.m., intermediate class in Room 1. During a particular meeting of that class, Mr. Jones was the head instructor and Ms. Chen served as an assistant instructor.
- Each student holds a rank in the martial arts. The rank name, belt color, and rank requirements are stored. Most ranks have numerous rank requirements, but each requirement is associated with only one particular rank. All ranks except white belt have at least one requirement.
- A given rank may be held by many students. While it is customary to think of a student as having a single rank, it is necessary to track each student's progress through the ranks. Therefore, every rank that a student attains is kept in the system. New students joining the school are automatically given the rank of white belt. The date that a student is awarded each rank should be kept in the system. All ranks have at least one student who has achieved that rank at some time.

9. The *Journal of E-commerce Research Knowledge* is a prestigious information systems research journal. It uses a peer-review process to select manuscripts for publication. Only about 10 percent of the manuscripts submitted to the journal are accepted for publication. A new issue of the journal is published each quarter. Create a complete ERD to support the business needs described below.
   - Unsolicited manuscripts are submitted by authors. When a manuscript is received, the editor assigns it a number and records some basic information about it in the system, including the title of the manuscript, the date it was received, and a manuscript status of "received." Information about the author(s) is also recorded, including each author's name, mailing address, email address, and affiliation (the author's school or company). Every manuscript must have an author. Only authors who have submitted manuscripts are kept in the system.

It is typical for a manuscript to have several authors. A single author may have submitted many different manuscripts to the journal. Additionally, when a manuscript has multiple authors, it is important to record the order in which the authors are listed in the manuscript credits.

- At his or her earliest convenience, the editor will briefly review the topic of the manuscript to ensure that its contents fall within the scope of the journal. If the content is not appropriate for the journal, the manuscript's status is changed to "rejected," and the author is notified via email. If the content is within the scope of the journal, then the editor selects three or more reviewers to review the manuscript. Reviewers work for other companies or universities and read manuscripts to ensure their scientific validity. For each reviewer, the system records a reviewer number, name, email address, affiliation, and areas of interest. Areas of interest are predefined areas of expertise that the reviewer has specified. An area of interest is identified by an IS code and includes a description (e.g., IS2003 is the code for "database modeling"). A reviewer can have many areas of interest, and an area of interest can be associated with many reviewers. All reviewers must specify at least one area of interest. It is unusual, but possible, to have an area of interest for which the journal has no reviewers. The editor will change the status of the manuscript to "under review" and record which reviewers received the manuscript and the date it was sent to each reviewer. A reviewer will typically receive several manuscripts to review each year, although new reviewers may not have received any manuscripts yet.
- The reviewers will read the manuscript at their earliest convenience and provide feedback to the editor. The feedback from each reviewer includes rating the manuscript on a 10-point scale for appropriateness, clarity, methodology, and contribution to the field, as well as a recommendation for publication (accept or reject). The editor will record all of this information in the system for each review received, along with the date the feedback was received. Once all of the reviewers have provided their evaluations, the editor will decide whether to publish the manuscript and change its status to "accepted" or "rejected." If the manuscript will be published, the date of acceptance is recorded.
- Once a manuscript has been accepted for publication, it must be scheduled. For each issue of the journal, the publication period (fall, winter, spring, or summer), publication year, volume, and number are recorded. An issue will contain many manuscripts, although the issue may be created in the system before it is known which manuscripts will be published in that issue. An accepted manuscript appears in only one issue of the journal. Each manuscript goes through a typesetting process that formats the content, including fonts, font size, line spacing, justification, and so on. Once the manuscript has been typeset, its number of pages is recorded in the system. The editor will then decide which issue each accepted manuscript will appear in and the order of manuscripts within each issue. The order and the beginning page number for each manuscript must be stored in the system. Once the manuscript has been scheduled for an issue, the status of the manuscript is changed to "scheduled." Once an issue is published, the print date for the issue is recorded, and the status of each manuscript in that issue is changed to "published."

10. Global Unified Technology Sales (GUTS) is moving toward a "bring your own device" (BYOD) model for employee computing. Employees can use traditional

desktop computers in their offices. They can also use a variety of personal mobile computing devices such as tablets, smartphones, and laptops. The new computing model introduces some security risks that GUTS is attempting to address. The company wants to ensure that any devices connecting to their servers are properly registered and approved by the Information Technology department. Create a complete ERD to support the business needs described below:

- Every employee works for a department that has a department code, name, mail box number, and phone number. The smallest department currently has 5 employees, and the largest department has 40 employees. This system will only track in which department an employee is currently employed. Very rarely, a new department can be created within the company. At such times, the department may exist temporarily without any employees. For every employee, an employee number and name (first, last, and middle initial) are recorded in the system. It is also necessary to keep each employee's title.
- An employee can have many devices registered in the system. Each device is assigned an identification number when it is registered. Most employees have at least one device, but newly hired employees might not have any devices registered initially. For each device, the brand and model need to be recorded. Only devices that are registered to an employee will be in the system. While unlikely, it is possible that a device could transfer from one employee to another. However, if that happens, only the employee who currently owns the device is tracked in the system. When a device is registered in the system, the date of that registration needs to be recorded.
- Devices can be either desktop systems that reside in a company office or mobile devices. Desktop devices are typically provided by the company and are intended to be a permanent part of the company network. As such, each desktop device is assigned a static IP address, and the MAC address for the computer hardware is kept in the system. A desktop device is kept in a static location (building name and office number). This location should also be kept in the system so that, if the device becomes compromised, the IT department can dispatch someone to remediate the problem.
- For mobile devices, it is important to also capture the device's serial number, which operating system (OS) it is using, and the version of the OS. The IT department is also verifying that each mobile device has a screen lock enabled and has encryption enabled for data. The system should support storing information on whether or not each mobile device has these capabilities enabled.
- Once a device is registered in the system, and the appropriate capabilities are enabled if it is a mobile device, the device may be approved for connections to one or more servers. Not all devices meet the requirements to be approved at first, so the device might be in the system for a period of time before it is approved to connect to any server. GUTS has a number of servers, and a device must be approved for each server individually. Therefore, it is possible for a single device to be approved for several servers but not for all servers.
- Each server has a name, brand, and IP address. Within the IT department's facilities are a number of climate-controlled server rooms where the physical servers can be located. Which room each server is in should also be recorded. Further, it is necessary to track which operating system is being used on each server.

Some servers are virtual servers and some are physical servers. If a server is a virtual server, then the system should track which physical server it is running on. A single physical server can host many virtual servers, but each virtual server is hosted on only one physical server. Only physical servers can host a virtual server. In other words, one virtual server cannot host another virtual server. Not all physical servers host a virtual server.

- A server will normally have many devices that are approved to access the server, but it is possible for new servers to be created that do not yet have any approved devices. When a device is approved for connection to a server, the date of that approval should be recorded. It is also possible for a device that was approved for a server to lose its approval. If that happens, the date that the approval was removed should be recorded. If a device loses its approval, it may regain that approval at a later date if whatever circumstance that lead to the removal is resolved.
- A server can provide many user services, such as email, chat, homework managers, and others. Each service on a server has a unique identification number and name. The date that GUTS began offering that service should be recorded. Each service runs on only one server although new servers might not offer any services initially. Client-side services are not tracked in this system, so every service must be associated with a server.
- Employees must get permission to access a service before they can use it. Most employees have permissions to use a wide array of services, but new employees might not have permission on any service. Each service can support multiple approved employees as users, but new services might not have any approved users at first. The date on which the employee is approved to use a service is tracked by the system. The first time an employee is approved to access a service, the employee must create a username and password. This will be the same username and password that the employee will use for every service for which the employee is eventually approved.

11. Global Computer Solutions (GCS) is an information technology consulting company with many offices throughout the United States. The company's success is based on its ability to maximize its resources—that is, its ability to match highly skilled employees with projects according to region. To better manage its projects, GCS has contacted you to design a database, so GCS managers can keep track of their customers, employees, projects, project schedules, assignments, and invoices.

    The GCS database must support all of GCS's operations and information requirements. A basic description of the main entities follows:

    - The *employees* of GCS must have an employee ID, a last name, a middle initial, a first name, a region, and a date of hire recorded in the system.
    - Valid *regions* are as follows: Northwest (NW), Southwest (SW), Midwest North (MN), Midwest South (MS), Northeast (NE), and Southeast (SE).
    - Each employee has many skills, and many employees have the same skill.
    - Each *skill* has a skill ID, description, and rate of pay. Valid skills are as follows: Data Entry I, Data Entry II, Systems Analyst I, Systems Analyst II, Database Designer I, Database Designer II, Java I, Java II, C++ I, C++ II, Python I, Python II, ColdFusion I, ColdFusion II, ASP I, ASP II, Oracle DBA, MS SQL Server

TABLE P5.11A

SKILL	EMPLOYEE
Data Entry I	Seaton Amy; Williams Josh; Underwood Trish
Data Entry II	Williams Josh; Seaton Amy
Systems Analyst I	Craig Brett; Sewell Beth; Robbins Erin; Bush Emily; Zebras Steve
Systems Analyst II	Chandler Joseph; Burklow Shane; Robbins Erin
DB Designer I	Yarbrough Peter; Smith Mary
DB Designer II	Yarbrough Peter; Pascoe Jonathan
Java I	Kattan Chris; Ephanor Victor; Summers Anna; Ellis Maria
Java II	Kattan Chris; Ephanor Victor; Batts Melissa
C++ I	Smith Jose; Rogers Adam; Cope Leslie
C++ II	Rogers Adam; Bible Hanah
Python I	Zebras Steve; Ellis Maria
Python II	Zebras Steve; Newton Christopher
ColdFusion I	Duarte Miriam; Bush Emily
ColdFusion II	Bush Emily; Newton Christopher
ASP I	Duarte Miriam; Bush Emily
ASP II	Duarte Miriam; Newton Christopher
Oracle DBA	Smith Jose; Pascoe Jonathan
SQL Server DBA	Yarbrough Peter; Smith Jose
Network Engineer I	Bush Emily; Smith Mary
Network Engineer II	Bush Emily; Smith Mary
Web Administrator	Bush Emily; Smith Mary; Newton Christopher
Technical Writer	Kilby Surgena; Bender Larry
Project Manager	Paine Brad; Mudd Roger; Kenyon Tiffany; Connor Sean

DBA, Network Engineer I, Network Engineer II, Web Administrator, Technical Writer, and Project Manager. Table P5.11a shows an example of the Skills Inventory.

- GCS has many *customers*. Each customer has a customer ID, name, phone number, and region.
- GCS works by *projects*. A project is based on a contract between the customer and GCS to design, develop, and implement a computerized solution. Each project has specific characteristics such as the project ID, the customer to which the project belongs, a brief description, a project date (the date the contract was signed), an estimated project start date and end date, an estimated project budget, an actual start date, an actual end date, an actual cost, and one employee assigned as the manager of the project.
- The actual cost of the project is updated each Friday by adding that week's cost to the actual cost. The week's cost is computed by multiplying the hours each employee worked by the rate of pay for that skill.
- The employee who is the manager of the project must complete a *project schedule*, which effectively is a design and development plan. In the project schedule (or plan), the manager must determine the tasks that will be performed to take the project from beginning to end. Each task has a task ID, a brief task description, starting and ending dates, the types of skills needed, and the number of employees (with the required skills) needed to complete the task. General tasks are the initial interview, database and system design, implementation, coding, testing, and final evaluation and sign-off. For example, GCS might have the project schedule shown in Table P5.11b.

TABLE P5.11B

PROJECT ID: 1 COMPANY: SEE ROCKS START DATE: 3/1/2018		DESCRIPTION: SALES MANAGEMENT SYSTEM CONTRACT DATE: 2/12/2018 END DATE: 7/1/2018		REGION: NW BUDGET: $15,500
START DATE	END DATE	TASK DESCRIPTION	SKILL(S) REQUIRED	QUANTITY REQUIRED
3/1/18	3/6/18	Initial interview	Project Manager Systems Analyst II DB Designer I	1 1 1
3/11/18	3/15/18	Database design	DB Designer I	1
3/11/18	4/12/18	System design	Systems Analyst II Systems Analyst I	1 2
3/18/18	3/22/18	Database implementation	Oracle DBA	1
3/25/18	5/20/18	System coding and testing	Java I Java II Oracle DBA	2 1 1
3/25/18	6/7/18	System documentation	Technical Writer	1
6/10/18	6/14/18	Final evaluation	Project Manager Systems Analyst II DB Designer I Java II	1 1 1 1
6/17/18	6/21/18	On-site system online and data loading	Project Manager Systems Analyst II DB Designer I Java II	1 1 1 1
7/1/18	7/1/18	Sign-off	Project Manager	1

- GCS pools all of its employees by region; from this pool, employees are assigned to a specific task scheduled by the project manager. For example, in the first project's schedule, you know that a Systems Analyst II, Database Designer I, and Project Manager are needed for the period from 3/1/18 to 3/6/18. The project manager is assigned when the project is created and remains for the duration of the project. Using that information, GCS searches the employees who are located in the same region as the customer, matches the skills required, and assigns the employees to the project task.
- Each project schedule task can have many employees assigned to it, and a given employee can work on multiple project tasks. However, an employee can work on only one project task at a time. For example, if an employee is already assigned to work on a project task from 2/20/18 to 3/3/18, the employee cannot work on another task until the current assignment is closed (ends). The date that an assignment is closed does not necessarily match the ending date of the project schedule task because a task can be completed ahead of or behind schedule.
- Given all of the preceding information, you can see that the assignment associates an employee with a project task, using the project schedule. Therefore, to keep track of the *assignment*, you require at least the following information: assignment ID, employee, project schedule task, assignment start date, and assignment end date. The end date could be any date, as some projects run ahead of or behind schedule. Table P5.11c shows a sample assignment form.

TABLE P5.11C

PROJECT ID: 1		DESCRIPTION: SALES MANAGEMENT SYSTEM				
COMPANY: SEE ROCKS		CONTRACT DATE: 2/12/2018			AS OF: 03/29/18	
SCHEDULED				ACTUAL ASSIGNMENTS		
PROJECT TASK	START DATE	END DATE	SKILL	EMPLOYEE	START DATE	END DATE
Initial interview	3/1/18	3/6/18	Project Mgr. Sys. Analyst II DB Designer I	101-Connor S. 102-Burklow S. 103-Smith M.	3/1/18 3/1/18 3/1/18	3/6/18 3/6/18 3/6/18
Database design	3/11/18	3/15/18	DB Designer I	104-Smith M.	3/11/18	3/14/18
System design	3/11/18	4/12/18	Sys. Analyst II Sys. Analyst I Sys. Analyst I	105-Burklow S. 106-Bush E. 107-Zebras S.	3/11/18 3/11/18 3/11/18	
Database implementation	3/18/18	3/22/18	Oracle DBA	108-Smith J.	3/15/18	3/19/18
System coding and testing	3/25/18	5/20/18	Java I Java I Java II Oracle DBA	109-Summers A. 110-Ellis M. 111-Ephanor V. 112-Smith J.	3/21/18 3/21/18 3/21/18 3/21/18	
System documentation	3/25/18	6/7/18	Tech. Writer	113-Kilby S.	3/25/18	
Final evaluation	6/10/18	6/14/18	Project Mgr. Sys. Analyst II DB Designer I Java II			
On-site system online and data loading	6/17/18	6/21/18	Project Mgr. Sys. Analyst II DB Designer I Java II			
Sign-off	7/1/18	7/1/18	Project Mgr.			

(*Note:* The assignment number is shown as a prefix of the employee name—for example, 101 or 102.) Assume that the assignments shown previously are the only ones as of the date of this design. The assignment number can be any number that matches your database design.

- Employee work hours are kept in a *work log*, which contains a record of the actual hours worked by employees on a given assignment. The work log is a form that the employee fills out at the end of each week (Friday) or at the end of each month. The form contains the date, which is either the current Friday of the month or the last workday of the month if it does not fall on a Friday. The form also contains the assignment ID, the total hours worked either that week or up to the end of the month, and the bill number to which the work-log entry is charged. Obviously, each work-log entry can be related to only one bill. A sample list of the current work-log entries for the first sample project is shown in Table P5.11d.
- Finally, every 15 days, a *bill* is written and sent to the customer for the total hours worked on the project during that period. When GCS generates a bill, it uses the bill number to update the work-log entries that are part of the bill. In summary, a bill can refer to many work-log entries, and each work-log entry can be related to only

TABLE P5.11D

EMPLOYEE NAME	WEEK ENDING	ASSIGNMENT NUMBER	HOURS WORKED	BILL NUMBER
Burklow S.	3/1/18	1-102	4	xxx
Connor S.	3/1/18	1-101	4	xxx
Smith M.	3/1/18	1-103	4	xxx
Burklow S.	3/8/18	1-102	24	xxx
Connor S.	3/8/18	1-101	24	xxx
Smith M.	3/8/18	1-103	24	xxx
Burklow S.	3/15/18	1-105	40	xxx
Bush E.	3/15/18	1-106	40	xxx
Smith J.	3/15/18	1-108	6	xxx
Smith M.	3/15/18	1-104	32	xxx
Zebras S.	3/15/18	1-107	35	xxx
Burklow S.	3/22/18	1-105	40	
Bush E.	3/22/18	1-106	40	
Ellis M.	3/22/18	1-110	12	
Ephanor V.	3/22/18	1-111	12	
Smith J.	3/22/18	1-108	12	
Smith J.	3/22/18	1-112	12	
Summers A.	3/22/18	1-109	12	
Zebras S.	3/22/18	1-107	35	
Burklow S.	3/29/18	1-105	40	
Bush E.	3/29/18	1-106	40	
Ellis M.	3/29/18	1-110	35	
Ephanor V.	3/29/18	1-111	35	
Kilby S.	3/29/18	1-113	40	
Smith J.	3/29/18	1-112	35	
Summers A.	3/29/18	1-109	35	
Zebras S.	3/29/18	1-107	35	

Note: xxx represents the bill ID. Use the one that matches the bill number in your database.

one bill. GCS sent one bill on 3/15/18 for the first project (SEE ROCKS), totaling the hours worked between 3/1/18 and 3/15/18. Therefore, you can safely assume that there is only one bill in this table and that the bill covers the work-log entries shown in the preceding form.

Your assignment is to create a database that fulfills the operations described in this problem. The minimum required entities are employee, skill, customer, region, project, project schedule, assignment, work log, and bill. (There are additional required entities that are not listed.)

- Create all of the required tables and required relationships.
- Create the required indexes to maintain entity integrity when using surrogate primary keys.
- Populate the tables as needed, as indicated in the sample data and forms.