<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>INTRODUCTION TO DATABASE ENVIRONMENTS</strong></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Introduction to Database Management</td>
</tr>
<tr>
<td>2</td>
<td>Introduction to Database Development</td>
</tr>
<tr>
<td><strong>UNDERSTANDING RELATIONAL DATABASES</strong></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>The Relational Data Model</td>
</tr>
<tr>
<td>4</td>
<td>Query Formulation with SQL</td>
</tr>
<tr>
<td><strong>DATA MODELING</strong></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Understanding Entity Relationship Diagrams</td>
</tr>
<tr>
<td>6</td>
<td>Developing Data Models for Business Databases</td>
</tr>
<tr>
<td><strong>RELATIONAL DATABASE DESIGN</strong></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Normalization of Relational Tables</td>
</tr>
<tr>
<td>8</td>
<td>Physical Database Design</td>
</tr>
<tr>
<td><strong>APPLICATION DEVELOPMENT WITH RELATIONAL DATABASES</strong></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Advanced Query Formulation with SQL</td>
</tr>
<tr>
<td>10</td>
<td>Application Development with Views</td>
</tr>
<tr>
<td>11</td>
<td>Stored Procedures and Triggers</td>
</tr>
<tr>
<td><strong>ADVANCED DATABASE DEVELOPMENT</strong></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>View Design and Integration</td>
</tr>
<tr>
<td>13</td>
<td>Database Development for Student Loan Limited</td>
</tr>
<tr>
<td><strong>MANAGING DATABASE ENVIRONMENTS</strong></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Data and Data Administration</td>
</tr>
<tr>
<td>15</td>
<td>Transaction Management</td>
</tr>
<tr>
<td>16</td>
<td>Data Warehouse Concepts and Design</td>
</tr>
<tr>
<td>17</td>
<td>Data Integration Practices and Relational DBMS Extensions for Data Warehouses</td>
</tr>
<tr>
<td>18</td>
<td>Client-Server Processing, Parallel Database Processing, and Distributed Databases</td>
</tr>
<tr>
<td>19</td>
<td>Object Database Management Systems</td>
</tr>
</tbody>
</table>
Detailed Table of Contents

Preface

PART 1: INTRODUCTION TO DATABASE ENVIRONMENTS

CHAPTER 1 Introduction to Database Management

1.1 Database Characteristics 2

1.2 Features of Database Management Systems 4
   1.2.1 Database Definition 4
   1.2.2 Nonprocedural Access 6
   1.2.3 Application Development and Procedural Language Interface 7
   1.2.4 Features to Support Database Operations 8
   1.2.5 Third-Party Features 9

1.3 Development of Database Technology and Market Structure 9
   1.3.1 Evolution of Database Technology 9
   1.3.2 Current Market for Database Software 11

1.4 Architectures of Database Management Systems 12
   1.4.1 Data Independence and the Three Schema Architecture 12
   1.4.2 Parallel and Distributed Database Processing 14

1.5 Organizational Impacts of Database Technology 16
   1.5.1 Interacting with Databases 17
   1.5.2 Managing Data Resources in Organizations 18

CHAPTER 2 Introduction to Database Development

2.1 Information Systems 23
   2.1.1 Components of Information Systems 23
   2.1.2 Information Systems Development Process 24

2.2 Goals of Database Development 26
   2.2.1 Develop a Common Vocabulary 26
   2.2.2 Define the Meaning of Data 26
   2.2.3 Ensure Data Quality 27
   2.2.4 Find an Efficient Implementation 27

2.3 Database Development Process 28
   2.3.1 Phases of Database Development 28
      Conceptual Data Modeling
      Logical Database Design
      Distributed Database Design
      Physical Database Design
      Splitting Conceptual Design for Large Projects
      Cross-Checking with Application Development
   2.3.2 Skills in Database Development 32

2.4 Tools of Database Development 33
   2.4.1 Diagramming 33
   2.4.2 Documentation 34
   2.4.3 Analysis 34
   2.4.4 Prototyping Tools 34
   2.4.5 Commercial CASE Tools 35
4.2.2 Joining Tables
4.2.3 Summarizing Tables with GROUP BY and HAVING
4.2.4 Improving the Appearance of Results

4.3 Conceptual Evaluation Process for SELECT Statements

4.4 Critical Questions for Query Formulation

4.5 Refining Query Formulation Skills with Examples
4.5.1 Joining Multiple Tables with the Cross Product Style
4.5.2 Joining Multiple Tables with the Join Operator Style
4.5.3 Self-Joins and Multiple Joins between Two Tables
4.5.4 Combining Joins and Grouping
4.5.5 Traditional Set Operators in SQL

4.6 SQL Modification Statements
Part 1: SELECT
Part 2: INSERT, UPDATE, and DELETE statements

- Simplified SELECT Syntax
- INSERT Syntax
- UPDATE Syntax
- DELETE Syntax

Appendix 4.B: Syntax Differences among Major DBMS Products

CHAPTER 5 Understanding Entity Relationship Diagrams

5.1 Introduction to Entity Relationship Diagrams
5.1.1 Basic Symbols
5.1.2 Relationship Cardinality
- Crow’s Foot Representation of Cardinalities
- Classification of Cardinalities
5.1.3 Comparison to Relational Database Diagrams

5.2 Understanding Relationships
5.2.2 Relationship Patterns
- M-N Relationships with Attributes
- Self-Referencing (Unary) Relationships
- Associative Entity Types Representing Multi-Way (M-Way) Relationships
5.2.3 Equivalence between 1-M and M-N Relationships

5.3 Classification in the Entity Relationship Model
5.3.1 Generalization Hierarchies
5.3.2 Disjointness and Completeness Constraints
5.3.3 Multiple Levels of Generalization

5.4 Notation Summary and Diagram Rules
5.4.1 Notation Summary
- Representation of Business Rules in an ERD
5.4.2 Diagram Rules
- Example of Rule Violations and Resolutions
- Alternative Organization of Rules

5.5 Comparison to Other Notations
5.5.1 Range of ERD Variations
5.5.2 ERD Notation in Aqua Data Studio
5.5.3 ERD Notation in Oracle SQL Developer
CHAPTER 10 Application Development with Views

10.1 Background
   10.1.1 Motivation
   10.1.2 View Definition

10.2 Using Views for Retrieval
   10.2.1 Using Views in SELECT Statements
   10.2.2 Processing Queries with View References

10.3 Updating Using Views
   10.3.1 Single-Table Updatable Views
       Rules for Single-Table Updatable Views
       View Updates with Side Effects
   10.3.2 Multiple-Table Updatable Views
       Rules for 1-M Updatable Queries
       Inserting Rows in 1-M Updatable Queries
       1-M Updatable Queries with More than Two Tables

10.4 Using Views in Hierarchical Forms
   10.4.1 What Is a Hierarchical Form?
   10.4.2 Relationship between Hierarchical Forms and Tables
   10.4.3 Data Requirements for Hierarchical Forms
       Step 1: Identify the 1-M Relationship
       Step 2: Identify the Linking Columns
       Step 3: Determine Other Tables
       Step 4: Determine Updatable Tables
       Step 5: Write Form Queries
       Common Query Formulation Errors

10.5 Using Views in Reports
   10.5.1 What is a Hierarchical Report?
   10.5.2 Data Requirements for Hierarchical Reports

References for Further Study

Appendix 10.A: SQL:2011 Syntax Summary
   CREATE VIEW Statement
   DROP VIEW Statement

Appendix 10.B: Rules for Updatable Join Views in Oracle

CHAPTER 11 Stored Procedures and Triggers

11.1 Database Programming Languages and PL/SQL
   11.1.1 Motivation for Database Programming Languages
       Customization
12.2.2 Analysis of M-Way Relationships Using Forms

12.3 View Integration

12.3.1 Incremental and Parallel Integration Approaches
- Determining an Integration Strategy
- Precedence Relationships among Forms
- Resolving Synonyms and Homonyms

12.3.2 View Integration Examples
- Incremental Integration Example
- Parallel Integration Example

CHAPTER 13 Database Development for Student Loan Limited

13.1 Case Description

13.1.1 Overview

13.1.2 Flow of Work
- Loan Origination Form
- Disclosure Letter
- Statement of Account
- Loan Activity Report
- New Technology

13.2 Conceptual Data Modeling

13.2.1 ERD for the Loan Origination Form

13.2.2 Incremental Integration after Adding the Disclosure Letter

13.2.3 Incremental Integration after Adding the Statement of Account

13.2.4 Incremental Integration after Adding the Loan Activity Report

13.3 Refining the Conceptual Schema

13.3.1 Schema Conversion

13.3.2 Normalization

13.4 Physical Database Design and Application Development

13.4.1 Application and Table Profiles

13.4.2 Index Selection

13.4.3 Derived Data and Denormalization Decisions

13.4.4 Other Implementation Decisions

13.4.5 Application Development
- Data Requirements for the Loan Origination Form
- Data Requirements for the Loan Activity Report
- Derived Data Maintenance

Appendix 13.A: Glossary of Form and Report Fields

Appendix 13.B: CREATE TABLE Statements

PART 7: MANAGING DATABASE ENVIRONMENTS

CHAPTER 14 Data and Database Administration

14.1 Organizational Context for Managing Databases

14.1.1 Database Support for Management Decision Making

14.1.2 Approaches for Managing Data Resources

14.1.3 Responsibilities of Data Specialists

14.1.4 Challenges of Big Data
16.3 Relational Database Design for Data Warehouses 588
  16.3.1 Relational Data Modeling Patterns 588
    Variations to the Star Schema
  16.3.2 Dimension Summarizability Problems and Patterns 591
    Dimension Summarizability Problems
    Dimension Summarizability Patterns
  16.3.3 Dimension-Fact Summarizability Problems and Patterns 595
  16.3.4 Time Representation in Star Schemas 597
  16.3.5 Dimension Representation 598

16.4 Enterprise Data Warehouse Development 600
  16.4.1 Data Warehouse Design Methodologies 601
    Demand Driven Methodology
    Supply Driven Methodology
    Hybrid Methodology
  16.4.2 Colorado Education Data Warehouse 603

CHAPTER 17  Data Integration Practices and Relational DBMS 613
  Extensions for Data Warehouses 613

17.1 Data Integration Concepts 614
  17.1.1 Sources of Data 614
  17.1.2 Workflow for Maintaining a Data Warehouse 615
  17.1.3 Data Cleaning Techniques 616
    String Parsing with Regular Expressions
    Correcting and Standardizing Values
    Entity Matching
  17.1.4 Data Integration Architectures and Tools 621
    Talend Open Studio
    Oracle Data Integration Tools
  17.1.5 Managing the Refresh Process 626

17.2 Extensions to SQL for Multidimensional Data 627
  17.2.1 CUBE Operator 628
  17.2.2 ROLLUP Operator 632
  17.2.3 GROUPING SETS Operator 634
  17.2.4 GROUP BY Operator Variations and Functions for Business Intelligence 635

17.3 Summary Data Storage and Optimization 636
  17.3.1 Materialized Views in Oracle 637
  17.3.2 Query Rewriting Principles 638
    Query Rewriting Details
  17.3.3 Storage and Optimization Technologies 642
    MOLAP (Multidimensional OLAP)
    ROLAP (Relational OLAP)
    HOLAP (Hybrid OLAP)
    Data Warehouse Appliances

CHAPTER 18  Client-Server Processing, Parallel Database 653
  Processing, and Distributed Databases 653

18.1 Overview of Distributed Processing and Distributed Data 654
  18.1.1 Motivation for Client-Server Processing 654
  18.1.2 Motivation for Parallel Database Processing 655
  18.1.3 Motivation for Distributed Data 656
# 18.2 Client-Server Database Architectures

**Division of Processing**

- **Middleware**

**18.2.2 Basic Architectures**

- Two-Tier Architecture
- Three-Tier Architecture
- Multiple-Tier Architecture

**18.2.3 Specialized Architectures**

- Web Services Architecture
- Cloud Computing Architectures
- Middleware for Extreme Transaction Processing

# 18.3 Parallel Database Processing

**18.3.1 Architectures and Design Issues**

**18.3.2 Commercial Parallel Database Technology**

- Oracle Real Application Clusters
- IBM DB2 Enterprise Server Edition with the DPF Option

# 18.4 Architectures for Distributed Database Management Systems

**18.4.1 Component Architecture**

# 18.5 Transparency for Distributed Database Processing

**18.5.1 Motivating Example**

**18.5.2 Fragmentation Transparency**

**18.5.3 Location Transparency**

**18.5.4 Local Mapping Transparency**

**18.5.5 Transparency in Oracle Distributed Databases**

- Oracle Distributed Database Technology for Site Autonomy
- Oracle Partitioning Technology for Fragmentation Transparency

# 18.6 Distributed Database Processing

**18.6.1 Distributed Query Processing**

**18.6.2 Distributed Transaction Processing**

- Distributed Concurrency Control
- Distributed Commit Processing
- CAP Design Philosophy

# 19.1 Motivation for Object Database Management

**19.1.1 Complex Data**

**19.1.2 Type System Mismatch**

**19.1.3 Application Examples**

- Mapping Websites and GPS Devices
- Dental Office Support
- Real Estate Listing Service
- Auto Insurance Claims

# 19.2 Object-Oriented Principles

**19.2.1 Encapsulation**

**19.2.2 Inheritance**

**19.2.3 Polymorphism**

**19.2.4 Programming Languages versus DBMSs**

# 19.3 Architectures for Object Database Management
19.3.1 Large Objects and External Software 700
19.3.2 Specialized Media Servers 701
19.3.3 Object Database Middleware 702
19.3.4 Object-Relational Database Management Systems for User-Defined Types 703
19.3.5 Object-Oriented Database Management Systems 704
19.3.6 Summary of Object Database Architectures 705

19.4 Object Database Features in SQL:2011 705
  19.4.1 User-Defined Types 706
  19.4.2 Table Definitions 708
  19.4.4 Manipulating Complex Objects and Subtable Families 711

19.5 Object Database Features in Oracle 712
  19.5.1 Defining User-Defined Types and Typed Tables in Oracle 713
  19.5.2 Using Typed Tables in Oracle 716
  19.5.3 Dependencies among Types and Typed Tables 718
  19.5.4 Other Object Features in Oracle 719

  - Type Substitution and Object Views
  - Nested Tables
  - XML Document Support
Preface

Motivating Example

Paul Hong, the owner of International Industrial Adhesives, Inc., is excited about potential opportunities in the recovering global economy. He senses major opportunities in new product development, new sources of demand, and industry consolidation. These opportunities, however, involve substantial risks with major changes in his business and industry. He senses risk from new mergers and acquisitions, new competitors, and increased government regulation and litigation in areas impacting his business. New mergers and acquisitions may involve challenges integrating disparate information technology and sharp increases in data and transaction volumes. The success of his business has attracted new competitors focusing on his most profitable customers and products. New government environmental, financial, and health regulations impose costly data collection efforts, reporting requirements, and compliance activities. Despite the tremendous opportunities for growth, he remains cautious about new directions to effectively manage risk.

Paul Hong must make timely and appropriate information technology investments to deal with strategic acquisitions, respond to competitors, and control costs of government mandates. To manage mergers and acquisitions, he must increase information technology capacity to process large new volumes of transactions, manage increasing amounts of data for operations, business intelligence, and long-term archival storage, and integrate disparate systems and data. To match competitors, he needs more detailed and timely data about industry trends, competitors’ actions, and intellectual property developments. To comply with new regulations, he must develop new data collection practices, conduct information technology audits, and fulfill other government reporting requirements for public companies. For all of these concerns, he is unsure about managing risks especially for his data resources.

These concerns involve significant usage of database technology as well as new data management initiatives to ensure accountability. New developments in extreme transaction processing, data warehouse appliances, and data lifecycle management can provide cost effective solutions for increasing capacity to meet the challenges of big data. These technologies can be deployed in cloud computing environments that provide economies of scale, elimination of fixed infrastructure costs, and dynamic scalability. A data governance organization can mitigate risks associated with the complex regulatory environment through a system of checks and balances using data rules and policies. Mergers and acquisitions often trigger data governance initiatives to ensure consistent data definitions and integrate corporate policies involving data privacy and security.

However, the solutions to Paul Hong’s concerns are found not just in technology. Utilization of the appropriate level of technology involves a vision for an organization’s future, a deep understanding of technology, and traditional management skills to control risk. Paul Hong realizes that his largest challenge is to blend these skills so that effective solutions can be developed for International Industrial Adhesives, Inc.

Introduction

This textbook provides a foundation to understand database technology supporting enterprise computing concerns such as those faced by Paul Hong. As a new student of database management, you first need to understand the fundamental concepts of database management and the relational data model. Then you need to master skills in database design and database application development. This textbook provides tools to help you understand relational databases and acquire skills to solve basic and advanced problems in query formulation, data modeling, normalization, application data requirements, and customization of database applications.

After establishing these skills, you are ready to study the organizational context, role of database specialists, and the processing environments in which databases are used. Students will learn about decision making needs, accountability requirements, organization structures, and roles of database specialists associated with databases and database technology. For environments, this textbook presents the fundamental database technologies in each processing environment and relates these technologies to new advances in electronic commerce and enterprise computing. You will learn the vocabulary, architectures, and design issues of database technology that provide a background for advanced study of individual database management systems, electronic commerce applications, and enterprise computing.
What's New in the Sixth Edition

The sixth edition makes substantial revisions to the fifth edition while preserving the proven pedagogy developed in the first five editions. Experience gained from my own instruction of undergraduate and graduate students along with feedback from adopters of the earlier editions has led to the development of new material and refinements to existing material.

The most significant changes in the sixth edition are new coverage of data governance, big data, and hierarchical queries. The changing regulatory and litigation environment over the last 10 years has spurred many organizations to initiate data governance programs. In response, information systems curriculums have added substantial coverage of governance programs for information technology. The new textbook coverage on data governance supports this trend with a large section in Chapter 14 reinforced with introductory coverage in Chapters 1 and 2. In many organizations, data specialists confront the problem of exploding data growth known as big data. To provide students background about big data issues and solution approaches, substantial new material has been added. A new section in Chapter 14 defines the challenges of big data while technology developments are presented in Chapter 1 (NoSQL DBMS products), Chapter 8 (solid state drives and information lifecycle management), Chapter 15 (relaxed transaction consistency models), Chapter 17 (data warehouse appliances), and Chapter 18 (extreme transaction processing and CAP Theorem). Hierarchical query formulation supports retrievals on organization charts, part explosion diagrams, chart of accounts, and other hierarchical business structures. Chapter 9 provides a large new section covering both proprietary Oracle notation and SQL standard notation for hierarchical queries. Students may gain a competitive advantage in job placement and career advancement through advanced query formulation skills involving hierarchical queries.

Besides the expanded coverage of data warehouses, the sixth edition provides numerous refinements to existing material based on classroom experience. New examples have been added in chapters 4, 6, 9, 10, and 11 in response to difficulties students had with textbook gaps. New end of chapter problems and questions in many chapters provide additional opportunities for students to test their understanding of database concepts and skills. The sixth edition has made substantial revisions to coverage of SQL standards, the DBMS marketplace, data modeling transformations, trigger formulation guidelines, histograms, storage area networks, data requirements for hierarchical forms, data modeling notation in Visio Professional, update locks, transaction processing case study, and the Data Warehouse Maturity Model. In addition, refinements and updates to most chapters have improved the presentation and currency of the material.

For database application development, the sixth edition covers SQL:2011, an evolutionary change from previous SQL standard versions (SQL:1999 to SQL:2008). The sixth edition explains the scope of SQL:2011, the difficulty of conformance with the standard, and new elements of the standard. Numerous refinements of database application development coverage extend the proven coverage of the first five editions: query formulation guidelines, advanced matching problems, query formulation tips for hierarchal forms and reports, trigger formulation guidelines, and transaction design guidelines.

For database administration and processing environments, the sixth edition provides expanded coverage of new technology in Oracle 12c. The most significant new topics are hybrid histograms, extended statistics for query optimization, the GENERATED clause for unique value generation, and Oracle proprietary notation for hierarchical queries. Significantly revised coverage is provided for the Oracle SQL Developer, Oracle RAC, the Oracle SQL Tuning Advisor, the Oracle Heat Map, Automatic Data Optimization, and Oracle triggers.

In addition to new material and refinements to existing material, the sixth edition extends the chapter supplements. The sixth edition contains new end-of-chapter questions and problems in most chapters. New material in the textbook’s website includes detailed tutorials about Microsoft Access 2013 and Visio Professional (2010 and 2013 versions), assignments for first and second database courses, and sample exams. The software tutorials for Microsoft Access and Visio Professional support concepts presented in textbook chapters 4, 5, 6, 9, and 10.

Competitive Advantages

This textbook provides outstanding features unmatched in competing textbooks. The unique features include detailed SQL coverage for both Microsoft Access and Oracle, problem-solving guidelines to aid acquisition of key skills, carefully designed sample databases and examples, a comprehensive case study, advanced topic coverage, integrated lab
material, coverage of prominent data modeling tools, and extensive data warehouse details. These features provide a complete package for both introductory and advanced database courses. Each of these features is described in more detail in the list below whereas Table P-1 summarizes the competitive advantages by chapter.

- **SQL Coverage**: The breadth and depth of the SQL coverage in this text is unmatched by competing textbooks. Table P-2 summarizes SQL coverage by chapter. Parts 2 and 5 provide thorough coverage of the CREATE TABLE, SELECT, UPDATE, INSERT, DELETE, CREATE VIEW, and CREATE TRIGGER statements. Numerous examples of basic, intermediate, and advanced problems are presented. The chapters in Part 7 cover statements useful for database administrators as well as statements used in specific processing environments.

- **Access and Oracle Coverage**: The chapters in Parts 2 and 5 provide detailed coverage of both Microsoft Access and Oracle SQL. Each example for the SELECT, INSERT, UPDATE, DELETE, and CREATE VIEW statements are shown for both database management systems. Significant coverage of advanced Oracle 12c SQL features appears in Chapters 8, 9, 11, 15, 17, and 19. In addition, the chapters in Parts 2 and 5 cover SQL:2011 syntax to support instruction with other prominent database management systems.

- **Problem-Solving Guidelines**: Students need more than explanations of concepts and examples to solve problems. Students need guidelines to help structure their thinking process to tackle problems in a systematic manner. The guidelines provide mental models to help students apply the concepts to solve basic and advanced problems. Table P-3 summarizes the unique problem-solving guidelines by chapter.

- **Sample Databases and Examples**: Two sample databases are used throughout the chapters of Parts 2 and 5 to provide consistency and continuity. The University database is used in the chapter examples, while the Order Entry database is used in the end-of-chapter problems. Numerous examples and problems with these databases depict the fundamental skills of query formulation and application data requirements. Revised versions of the databases provide separation between basic and advanced examples. The website contains CREATE TABLE statements, sample data, data manipulation statements, and Access database files for both databases.

- Chapters in Parts 3, 4, and 7 use additional databases to broaden exposure to more diverse business situations. Students need exposure to a variety of business situations to acquire database design skills and understand concepts important to database specialists. The supplementary databases cover water utility operations, patient visits, academic paper reviews, personal financial tracking, airline reservations, placement office operations, automobile insurance, store sales tracking, and real estate sales. In addition, Chapter 16 on data warehouses presents a substantial data warehouse, the Colorado Education Data Warehouse.

- **Comprehensive Case Study**: The Student Loan Limited Case is found at the end of Part 6. The case description along with its solution integrates the concepts students learned in the preceding 12 chapters on application development and database design. The follow-up problems at the end of the chapter provide additional opportunities for students to apply their knowledge on a realistic case.

- **Optional Integrated Labs**: Database management is best taught when concepts are closely linked to the practice of designing, implementing, and using databases with a commercial DBMS. To help students apply the concepts described in the textbook, optional supplementary lab materials are available on the text’s website. The website contains labs for four Microsoft Access versions (2003, 2007, 2010, and 2013) as well as practice databases and exercises. The Microsoft Access labs integrate a detailed coverage of Access with the application development concepts covered in Parts 2 and 5.

- **Data Modeling Tools**: The sixth edition expands coverage of commercial data modeling tools for database development. Students will find details about Aqua Data Studio, Oracle SQL Developer, and Visio Professional. Supplementary material on the textbook’s website provides detailed tutorials of Visio Professional 2010 and 2013. The Visio tutorials are closely integrated with data modeling coverage in Chapters 5 and 6.

- **Data Warehouse Coverage**: The two data warehouse chapters (16 and 17) provide enough details for a third of an advanced database course. The material can support three major assignments, two exams, and a project in an advanced database course.

- **Current and Cutting-Edge Topics**: This book covers some topics that are missing from competing textbooks: advanced query formulation, updatable views, development and management of stored proce-
dures and triggers, data requirements for data entry forms and reports, hierarchical query formulation, view integration, management of the refresh process for data warehouses, the data warehouse maturity model, data integration practices, parallel database architectures, object database architectures, data warehouse features in Oracle 12c, object-relational features in SQL:2011 and Oracle 12c, and transaction design principles. These topics enable motivated students to obtain a deeper understanding of database management.

- **Complete Package for Courses:** Depending on the course criteria, some students may need to purchase as many as four books for an introductory database course: a textbook covering principles, laboratory books covering details of a DBMS, a supplemental SQL book, and a casebook with realistic practice problems. This textbook and supplemental material provide a complete, integrated, and less expensive resource for students.

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Unique Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Unique chapter providing a conceptual introduction to the database development process</td>
</tr>
<tr>
<td>3</td>
<td>Visual representation of relational algebra operators</td>
</tr>
<tr>
<td>4</td>
<td>Query formulation guidelines; Oracle, Access, and SQL:2011 SQL coverage</td>
</tr>
<tr>
<td>5</td>
<td>Emphasis on ERD notation, business rules, and diagram rules with details about data modeling notation in prominent commercial data modeling tools</td>
</tr>
<tr>
<td>6</td>
<td>Strategies for analyzing business information needs, data modeling transformations, and detection of common design errors</td>
</tr>
<tr>
<td>7</td>
<td>Normalization guidelines and procedures</td>
</tr>
<tr>
<td>8</td>
<td>Index selection rules; SQL tuning guidelines, integrated coverage of query optimization, file structures, and index selection</td>
</tr>
<tr>
<td>9</td>
<td>Query formulation guidelines; Oracle 12c, Access, and SQL:2011 coverage; advanced topic coverage of nested queries, division problems, null value handling, and hierarchical queries</td>
</tr>
<tr>
<td>10</td>
<td>Rules for updatable views, data requirement steps for forms and reports</td>
</tr>
<tr>
<td>11</td>
<td>Unique chapter covering concepts and practices of database programming languages, stored procedures, and triggers; trigger formulation guidelines</td>
</tr>
<tr>
<td>12</td>
<td>Unique chapter covering concepts and practices of view integration and design</td>
</tr>
<tr>
<td>13</td>
<td>Unique chapter providing a comprehensive case study on student loan processing</td>
</tr>
<tr>
<td>14</td>
<td>Guidelines for important processes and tools used by database professionals; data governance and big data principles</td>
</tr>
<tr>
<td>15</td>
<td>Transaction design guidelines; mini case study about transaction design</td>
</tr>
<tr>
<td>16</td>
<td>Data warehouse maturity model for evaluating technology impact on organizations; relational database design for data warehouses; data warehouse development methodologies</td>
</tr>
<tr>
<td>17</td>
<td>Data integration concepts and practices; relational database features for multidimensional data; summary data storage and optimization; extensive Oracle 12c data warehouse coverage</td>
</tr>
<tr>
<td>18</td>
<td>Integrated coverage of client-server processing, parallel database processing, and distributed databases integrated with impact of cloud computing</td>
</tr>
<tr>
<td>19</td>
<td>Advanced topic coverage of object-relational features in SQL:2011 and Oracle 12c</td>
</tr>
</tbody>
</table>
### Table P-2: SQL Statement Coverage by Chapter

<table>
<thead>
<tr>
<th>Chapter</th>
<th>SQL Statement Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>CREATE TABLE</td>
</tr>
<tr>
<td>4</td>
<td>SELECT, INSERT, UPDATE, DELETE</td>
</tr>
<tr>
<td>9</td>
<td>SELECT (nested queries, outer joins, null value handling); Access and Oracle coverage</td>
</tr>
<tr>
<td>10</td>
<td>CREATE VIEW; queries and manipulation statements using views</td>
</tr>
<tr>
<td>11</td>
<td>CREATE PROCEDURE (Oracle), CREATE TRIGGER (Oracle and SQL:2011)</td>
</tr>
<tr>
<td>14</td>
<td>GRANT, REVOKE, CREATE ROLE, CREATE ASSERTION, CHECK clause of the CREATE TABLE statement, CREATE DOMAIN</td>
</tr>
<tr>
<td>15</td>
<td>COMMIT, ROLLBACK, SET TRANSACTION, SET CONSTRAINTS, SAVEPOINT</td>
</tr>
<tr>
<td>17</td>
<td>CREATE MATERIALIZED VIEW (Oracle), GROUP BY clause extensions (Oracle and SQL:2011), ranking functions (Oracle)</td>
</tr>
<tr>
<td>19</td>
<td>CREATE TYPE, CREATE TABLE (typed tables and subtables), SELECT extensions (object identifiers, path expressions, dereference operator); SQL:2011 and Oracle coverage</td>
</tr>
</tbody>
</table>

### Table P-3: Problem Solving Guidelines by Chapter

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Problem-Solving Guidelines</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Visual representations of relationships and relational algebra operators</td>
</tr>
<tr>
<td>4</td>
<td>Conceptual evaluation process; query formulation questions</td>
</tr>
<tr>
<td>5</td>
<td>Diagram rules</td>
</tr>
<tr>
<td>6</td>
<td>Guidelines for analyzing business information needs; design transformations; identification of common design errors; conversion rules</td>
</tr>
<tr>
<td>7</td>
<td>Guidelines for identifying functional dependencies; usage of sample data to eliminate functional dependencies; simple synthesis procedure</td>
</tr>
<tr>
<td>8</td>
<td>Index selection rules; SQL tuning guidelines</td>
</tr>
<tr>
<td>9</td>
<td>Difference problem formulation guidelines; nested query evaluation; count method for division problem formulation; hierarchical query formulation guidelines</td>
</tr>
<tr>
<td>10</td>
<td>Rules for updatable join queries; steps for analyzing data requirements in forms and reports</td>
</tr>
<tr>
<td>11</td>
<td>Trigger execution procedure; trigger formulation guidelines</td>
</tr>
<tr>
<td>12</td>
<td>Form analysis steps; view integration strategies</td>
</tr>
<tr>
<td>14</td>
<td>Guidelines to manage stored procedures and triggers; data planning process; DBMS selection process; Core processes and risk matrix in the Microsoft Data Governance Framework</td>
</tr>
<tr>
<td>15</td>
<td>Transaction timeline; transaction design guidelines</td>
</tr>
<tr>
<td>16</td>
<td>Data modeling patterns for summarizability, relational data modeling patterns for multidimensional data, guidelines for time representation in dimension tables</td>
</tr>
<tr>
<td>17</td>
<td>Data quality problems and associated data cleaning tasks, tradeoffs in refresh policies</td>
</tr>
<tr>
<td>18</td>
<td>Progression of transparency levels for distributed databases</td>
</tr>
<tr>
<td>19</td>
<td>Object database architectures; comparison between relational and object-relational representations</td>
</tr>
</tbody>
</table>
**Text Audience**

This book supports two database courses at the undergraduate or graduate levels. At the undergraduate level, students should have a concentration (major or minor) or active interest in information systems. For two-year institutions, the instructor may want to skip the advanced topics and place more emphasis on the optional Access lab book. Undergraduate students should have a first course covering general information systems concepts, spreadsheets, word processing, and possibly a brief introduction to databases.

At the graduate level, this book is suitable in either MBA or Master of Science (in information systems) programs. The advanced material in this book should be especially suitable for Master of Science students.

Except for Chapter 11, a previous course in computer programming can be useful background but is not mandatory. The other chapters reference some computer programming concepts, but writing code is not covered. For a complete mastery of Chapter 11, computer programming background is essential. However, the basic concepts and trigger details in Chapter 11 can be covered even if students do not have a computer programming background.

**Organization**

As the title suggests, *Database Design, Application Development, and Administration* emphasizes three sets of skills. Before acquiring these skills, students need a foundation about basic concepts. Part 1 provides conceptual background for subsequent detailed study of database design, database application development, and database administration. The chapters in Part 1 present the principles of database management and a conceptual overview of the database development process.

Part 2 provides foundational knowledge about the relational data model. Chapter 3 covers table definition, integrity rules, and operators to retrieve useful information from relational databases. Chapter 4 presents guidelines for query formulation and numerous examples of SQL SELECT statements.

Parts 3 and 4 emphasize practical skills and design guidelines for the database development process. Students desiring a career as a database specialist should be able to perform each step of the database development process. Students should learn skills of data modeling, schema conversion, normalization, and physical database design. The Part 3 chapters (Chapters 5 and 6) cover data modeling using the Entity Relationship Model. Chapter 5 covers the structure of entity relationship diagrams, while Chapter 6 presents usage of entity relationship diagrams to analyze business information needs. The Part 4 chapters (Chapters 7 and 8) cover table design principles and practice for logical and physical design. Chapter 7 covers the motivation, functional dependencies, normal forms, and practical considerations of data normalization. Chapter 8 contains broad coverage of physical database design including the objectives, inputs, file structures, query optimization principles, and important design choices.

Part 5 provides a foundation for building database applications by helping students acquire skills in advanced query formulation, specification of data requirements for data entry forms and reports, and coding triggers and stored procedures. Chapter 9 presents additional examples of intermediate and advanced SQL, along with corresponding query formulation skills. Chapter 10 describes the motivation, definition, and usage of relational views along with specification of view definitions for data entry forms and reports. Chapter 11 presents concepts and coding practices of database programming languages, stored procedures, and triggers for customization of database applications.

Part 6 covers advanced topics of database development. Chapter 12 describes view design and view integration, which are data modeling concepts for large database development efforts. Chapter 13 provides a comprehensive case study that enables students to gain insights about the difficulties of applying database design and application development skills to a realistic business database.

Beyond the database design and application development skills, this textbook prepares students for careers as database specialists. Students need to understand the responsibilities, tools, and processes employed by data administrators and database administrators as well as the various environments in which databases operate.

The chapters in Part 7 emphasize the role of database specialists and the details of managing databases in various operating environments. Chapter 14 provides a context for the other chapters through coverage of the responsibilities, tools, and processes used by database administrators and data administrators. The other chapters in Part 7 provide a
foundation for managing databases in important environments: Chapter 15 on transaction processing, Chapters 16 and 17 on data warehouses, Chapter 18 on distributed processing and data, and Chapter 19 on object database management. These chapters emphasize concepts, architectures, and design choices important for database specialists, while providing some coverage of advanced application development topics.

**Text Approach and Theme**

To support acquisition of the necessary skills for learning and understanding application development, database design, and managing databases, this book adheres to three guiding principles:

1. **Combine concepts and practice.** Database management is more easily learned when concepts are closely linked to the practice of designing and implementing databases using a commercial DBMS. The textbook and the accompanying supplements have been designed to provide close integration between concepts and practice through the following features:
   - SQL examples for both Access and Oracle as well as SQL:2011 coverage
   - Emphasis of the relationship between application development and query formulation
   - Usage of data modeling notations supported by professional CASE tools
   - Supplemental laboratory practice chapters that combine textbook concepts with details of commercial DBMSs

2. **Emphasize problem-solving skills.** This book features problem-solving guidelines to help students master the fundamental skills of data modeling, normalization, query formulation, and application development. The textbook and associated supplements provide a wealth of questions, problems, case studies, and laboratory practices in which students can apply their skills. With mastery of the fundamental skills, students will be poised for future learning about databases and change the way they think about computing in general.

3. **Provide introductory and advanced material.** Business students who use this book may have a variety of backgrounds. This book provides enough depth to satisfy the most eager students. However, the advanced parts are placed so that they can be skipped by the less inclined.

**Pedagogical Features**

This book contains the following pedagogical features to help students navigate through chapter content in a systematic fashion:

- **Learning Objectives** focus on the knowledge and skills students will acquire from studying the chapter.
- **Overviews** provide a snapshot or preview of chapter contents.
- **Key Terms** are highlighted and defined in boxed areas as they appear in the chapter.
- **Examples** are clearly separated from the rest of the chapter material for easier review and studying purposes.
- **Running Database Examples** — examples using the University database as well as other databases with clear separation from surrounding text.
- **Closing Thoughts** summarize chapter content in relation to the learning objectives.
- **Review Concepts** are the important conceptual highlights from the chapter, not just a list of terminology.
- **Questions** are provided to review the chapter concepts.
- **Problems** help students practice and implement the detailed skills presented in the chapter.
- **References for Further Study** point students to additional sources on chapter content.
- **Chapter Appendices** provide additional details, convenient summaries of SQL:2011 syntax, and other topics beyond the normal chapter coverage.
At the end of the text, students will find the following additional resources:

- **Glossary**: Provides a complete list of terms and definitions used throughout the text.
- **Bibliography**: A list of helpful industry, academic, and other printed material for further research or study.
- **Index**: A list of keywords with page references to help readers of the printed edition.

**Microsoft Access Labs**

Lab books for several versions of Microsoft Access (2003, 2007, 2010, and 2013) are available on the textbook’s website. The lab books provide detailed coverage of features important to beginning database students as well as many advanced features. The lab chapters provide a mixture of guided practice and reference material organized into the following chapters:

1. An Introduction to Microsoft Access
2. Database Creation Lab
3. Query Lab
4. Single Table Form Lab
5. Hierarchical Form Lab
6. Report Lab
7. Pivot Tables
8. User Interface Lab

Each lab chapter follows the pedagogy of the textbook with Learning Objectives, Overview, Closing Thoughts, Additional Practice exercises, and Appendixes of helpful tips. Most lab chapters reference concepts from the textbook for close integration with corresponding textbook chapters. Each lab book also includes a glossary of terms and an index.

**Instructor Resources**

A comprehensive set of supplements for the text and lab manuals is available to adopters.

- Powerpoint slides for each chapter
- Solutions to end of chapter problems for each chapter
- Solutions to end of chapter questions for each chapter
- Access databases for the university and order entry textbook databases
- Oracle SQL statements to create and populate the university and order entry textbook databases
- Files containing SQL statements used in the textbook chapters
- Case studies along with case study solutions
- Assignments used in a first database course. The assignments involve database creation, query formulation, application development with forms, data modeling, and normalization. In addition, a project assignment integrates material about database development and application development.
- Assignments used in a second database course. The assignments involve database creation, triggers, data warehouse design, data integration practices, query formulation for data warehouses, and object relational databases. In addition, projects are provided about Oracle advanced features, benchmark development, and management practices to develop or manage a significant database or data warehouse in an organization.
- Sample exams for a first course in database management
- Sample exams for an advanced course in database management
- Access databases for each lab chapter
- Access databases for end of chapter problems in each lab chapter
Teaching Paths

The textbook can be covered in several orders in a one- or a two-semester sequence. The author has taught a one-semester course with the ordering of relational database basics, query formulation, application development, database development, and database processing environments. This ordering has the advantage of covering the more concrete material (query formulation and application development) before the more abstract material (database development). Lab chapters and assignments are used for practice beyond the textbook chapters. To fit into one semester, advanced topics are skipped in Chapters 8 and 11 to 18.

A second ordering is to cover database development before application development. For this ordering, the author recommends following the textbook chapter ordering: 1, 2, 5, 6, 3, 7, 4, 9, and 10. The material on schema conversion in Chapter 6 should be covered after Chapter 3. This ordering supports a more thorough coverage of database development while not neglecting application development. To fit into one semester, advanced topics are skipped in Chapters 8 and 11 to 18.

A third possible ordering is to use the textbook in a two-course sequence. The first course covers database management fundamentals from Parts 1 and 2, data modeling and normalization from Parts 3 and 4, and advanced query formulation, application development with views, and view integration from Parts 5 and 6. The second course emphasizes database administration skills with physical database design from Part 4, triggers and stored procedures from Part 5, and the processing environments from Part 7 along with additional material on managing enterprise databases. A comprehensive project can be used in the second course to integrate application development, database development, and database administration.

Acknowledgments

The sixth edition is the culmination of many years of instruction, research, and industry experience. Before beginning the first edition, I wrote tutorials, laboratory practices, and case studies. This material was first used to supplement other textbooks. After encouragement from students, this material was used without a textbook. This material, revised many times through student comments, was the foundation for the first edition. During the development of the first edition, the material was classroom tested for three years with hundreds of undergraduate and graduate students, along with careful review through four drafts by many outside reviewers. The second edition was developed through classroom usage of the first edition for three years, along with teaching an advanced database course for several years. The third edition was developed through experience of three years with the second edition in basic and advanced database courses. The fourth edition was developed through three years of instruction with the third edition in beginning and advanced database courses. The fifth edition was developed through two years of instruction with the fourth edition in beginning and advanced database courses. The sixth edition was developed through two years of instruction with the fifth edition in beginning and advanced database courses.

I wish to acknowledge the excellent support that I have received in completing this project. I thank my many database students, especially those in ISMG6080, ISMG6480, and ISMG4500 at the University of Colorado Denver. Your comments and reaction to the textbook have been invaluable to its improvement.

About the Author

Michael V. Mannino has been involved in the database field since 1980. He has taught database management since 1983 at several major universities (University of Florida, University of Texas at Austin, University of Washington, and University of Colorado Denver). His audiences have included undergraduate MIS students, graduate MIS students, MBA students, and doctoral students as well as corporate employees in retraining programs. He has also been active in database research as evidenced by publications in major journals of the IEEE (Transactions on Knowledge and Data Engineering and Transactions on Software Engineering), ACM (Communications and Computing Surveys), and INFORMS (Informs Journal on Computing and Information Systems Research). His research includes several popular survey and tutorial articles as well as many papers describing original research. Practical results of his research have been incorporated into Chapter 12 on a form-driven approach to database design and Chapter 17 on management of the refresh process.
Overview

Chapter 2 provided a broad presentation about the database development process. You learned about the relationship between database development and information systems development, the phases of database development, and the kinds of skills you need to master. This chapter presents the notation of entity relationship diagrams to provide a foundation for using entity relationship diagrams in the database development process. To extend your database design skills, Chapter 6 describes the process of using entity relationship diagrams to develop data models for business databases.

To become a good data modeler, you need to understand the notation in entity relationship diagrams and apply the notation on problems of increasing complexity. To help you master the notation, this chapter presents the symbols used in entity relationship diagrams and compares entity relationship diagrams to relational database diagrams that you have seen in previous chapters. This chapter then probes deeper into relationships, the most distinguishing part of entity relationship diagrams. You will learn about identification dependency, relationship patterns, and equivalence between two kinds of relationships. Finally, you will learn to represent similarities among entity types using generalization hierarchies.

The next part of the chapter presents business rule representation and diagram rules to deepen your understanding of the Crow’s Foot notation. To provide an organizational focus, this chapter presents formal and informal representation of business rules in an entity relationship diagram. To help you avoid common notation errors, this chapter presents consistency and completeness rules and depicts their usage in examples.

Because of the plethora of entity relationship notations, you may not have the opportunity to use the Crow’s Foot notation exactly as shown in Chapters 5 and 6. To prepare you for understanding other notations, the chapter concludes with a presentation of diagram variations including the Class Diagram notation of the Unified Modeling Notation, one of the popular alternatives to the Entity Relationship Model.
This chapter provides the basic skills of data modeling to enable you to understand the notation of entity relationship diagrams. To apply data modeling as part of the database development process, you should study Chapter 6 on developing data models for business databases. Chapter 6 emphasizes the problem-solving skills of generating alternative designs, mapping a problem statement to an entity relationship diagram, and justifying design decisions. With the background provided in both chapters, you will be prepared to perform data modeling on case studies and databases for moderate-size organizations.

5.1 Introduction to Entity Relationship Diagrams

Gaining an initial understanding of entity relationship diagrams (ERDs) requires careful study. This section introduces the Crow’s Foot notation for ERDs, a popular notation supported by many CASE tools. To get started, this section begins with the basic symbols of entity types, relationships, and attributes. This section then explains cardinalities and their appearance in the Crow’s Foot notation. This section concludes by comparing the Crow’s Foot notation to relational database diagrams. If you are covering data modeling before relational databases, you may want to skip the last part of this section.

5.1.1 Basic Symbols

ERDs have three basic elements: entity types, relationships, and attributes. Entity types are collections of things of interest (entities) in an application. Entity types represent collections of physical things such as books, people, and places, as well as events such as payments. An entity is a member or instance of an entity type. Entities are uniquely identified to allow tracking across business processes. For example, customers have a unique identification to support order processing, shipment, and product warranty processes. In the Crow’s Foot notation as well as most other notations, rectangles denote entity types. In Figure 5.1, the Course entity type represents the set of courses in the database.

**Entity Type**: a collection of entities (persons, places, events, or things) of interest represented by a rectangle in an entity relationship diagram.

**Attribute**: a property of an entity type or relationship. Each attribute has a data type that defines the kind of values and permissible operations on the attribute.

**Relationship**: named associations among entity types. In the Crow’s Foot notation, relationship names appear on the line connecting the entity types involved in the relationship. In Figure 5.1, the Has relationship shows that the Course and Offering entity types are directly related. Relationships store associations in both directions. For example, the Has relationship shows the offerings for a given course and the associated course for a given offering. The Has relationship is binary because it involves two entity types. Section 5.2 presents examples of more complex relationships involving only one distinct entity type (unary relationships) and more than two entity types (M-way relationships).
**Relationship:** a named association among entity types. A relationship represents a two-way or bidirectional association among entities. Most relationships involve two distinct entity types.

Informally, ERDs have a natural language correspondence. Entity types can correspond to nouns and relationships to verbs or prepositional phrases connecting nouns. In this sense, one can read an entity relationship diagram as a collection of sentences. For example, the ERD in Figure 5.1 can be read as “course has offerings.” Note that there is an implied direction in each relationship. In the other direction, one could write, “offering is given for a course.” If practical, it is a good idea to use active rather than passive verbs for relationships. Therefore, Has is preferred as the relationship name. You should use the natural language correspondence as a guide rather than as a strict rule. For large ERDs, you will not always find a good natural language correspondence for all parts of a diagram.

### 5.1.2 Relationship Cardinality

Cardinalities constrain the number of objects that participate in a relationship. To depict the meaning of cardinalities, an instance diagram is useful. Figure 5.2 shows a set of courses (\{Course1, Course2, Course3\}), a set of offerings (\{Offering1, Offering2, Offering3, Offering4\}), and connections between the two sets. In Figure 5.2, Course1 is related to Offering1, Offering2, and Offering3, Course2 is related to Offering4, and Course3 is not related to any Offering entities. Likewise, Offering1 is related to Course1, Offering2 is related to Course1, Offering3 is related to Course1, and Offering4 is related to Course2. From this instance diagram, we might conclude that each offering is related to exactly one course. In the other direction, each course is related to 0 or more offerings.

**Cardinality:** a constraint on the number of entities that participate in a relationship. In an ERD, the minimum and maximum cardinalities are specified for both directions of a relationship.

![Figure 5.2: Instance Diagram for the Has Relationship](image)

Crow’s Foot Representation of Cardinalities

The Crow’s Foot notation uses three symbols to represent cardinalities. The Crow’s Foot symbol (two angled lines and one straight line) denotes many (zero or more) related entities. In Figure 5.3, the Crow’s Foot symbol near the Offering entity type means that a course can be related to many offerings. The circle means a cardinality of zero, while a line perpendicular to the relationship line denotes a cardinality of one.

![Figure 5.3: Entity Relationship Diagram with Cardinalities Noted](image)
To depict minimum and maximum cardinalities, the cardinality symbols are placed adjacent to each entity type in a relationship. The minimum cardinality symbol appears toward the relationship name while the maximum cardinality symbol appears toward the entity type. In Figure 5.3, a course is related to a minimum of zero offerings (circle in the inside position) and a maximum of many offerings (Crow’s Foot in the outside position). Similarly, an offering is related to exactly one (one and only one) course as shown by the single vertical lines in both inside and outside positions.

Classification of Cardinalities

Cardinalities are classified by common values for minimum and maximum cardinality. Table 5-1 shows two classifications for minimum cardinalities. A minimum cardinality of one or more indicates a mandatory relationship. For example, participation in the Has relationship is mandatory for each Offering entity due to the minimum cardinality of one. A mandatory relationship makes the entity type existence dependent on the relationship. The Offering entity type depends on the Has relationship because an Offering entity cannot be stored without a related Course entity. In contrast, a minimum cardinality of zero indicates an optional relationship. For example, the Has relationship is optional to the Course entity type because a Course entity can be stored without being related to an Offering entity. Figure 5.4 shows that the Teaches relationship is optional for both entity types.

Existence Dependency: an entity that cannot exist unless another related entity exists. A mandatory relationship creates an existence dependency.

<table>
<thead>
<tr>
<th>Classification</th>
<th>Cardinality Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mandatory</td>
<td>Minimum cardinality ≥ 1</td>
</tr>
<tr>
<td>Optional</td>
<td>Minimum cardinality = 0</td>
</tr>
<tr>
<td>Functional or single-valued</td>
<td>Maximum cardinality = 1</td>
</tr>
<tr>
<td>1-M</td>
<td>Maximum cardinality = 1 in one direction and maximum cardinality &gt; 1 in the other direction.</td>
</tr>
<tr>
<td>M-N</td>
<td>Maximum cardinality is &gt; 1 in both directions.</td>
</tr>
<tr>
<td>1-1</td>
<td>Maximum cardinality = 1 in both directions.</td>
</tr>
</tbody>
</table>

Table 5-1: Summary of Cardinality Classifications

Figure 5.4: Optional Relationship for Both Entity Types

Table 5-1 also shows several classifications for maximum cardinalities. A maximum cardinality of one means the relationship is single-valued or functional. For example, the Has and Teaches relationships are functional for Offering because an Offering entity can be related to a maximum of one Course and one Faculty entity. The word function comes from mathematics where a function gives one value. A relationship that has a maximum cardinality of one in one direction and more than one (many) in the other direction is called a 1-M (read one-to-many) relationship. Both the Has and Teaches relationships are 1-M.

Similarly, a relationship that has a maximum cardinality of more than one in both directions is known as an M-N (many-to-many) relationship. In Figure 5.5, the TeamTeaches relationship allows multiple professors to jointly teach the same offering, as shown in the instance diagram of Figure 5.6. M-N relationships are common in business databases to represent the connection between parts and suppliers, authors and books, and skills and employees. For example, a part can be supplied by many suppliers and a supplier can supply many parts.

Less common are 1-1 relationships in which the maximum cardinality equals one in both directions. For example, the WorksIn relationship in Figure 5.5 allows a faculty to be assigned to one office and an office to be occupied by at most one faculty.
5.1.3 Comparison to Relational Database Diagrams

To finish this section, let us compare the notation in Figure 5.3 with the relational database diagrams (from Microsoft Access) which you seen in previous chapters. It is easy to become confused between the two notations. Some of the major differences are listed below. To help you visualize these differences, Figure 5.7 shows a relational database diagram for the Course-Offering example.

1. Relational database diagrams do not use names for relationships. Instead foreign keys represent relationships. The ERD notation does not use foreign keys. For example, Offering.CourseNo is a column in Figure 5.7 but not an attribute in Figure 5.3.

2. Relational database diagrams show only maximum cardinalities.

3. Some ERD notations (including the Crow’s Foot notation) allow both entity types and relationships to have attributes. Relational database diagrams only allow tables to have columns.

4. Relational database diagrams allow a relationship between two tables. Some ERD notations (although not the Crow’s Foot notation) allow M-way relationships involving more than two entity types. The next section shows how to represent M-way relationships in the Crow’s Foot notation.

5. In some ERD notations (although not the Crow’s Foot notation), the position of the cardinalities is reversed.

5.2 Understanding Relationships

This section explores the entity relationship notation in more depth by examining important aspects of relationships. The first subsection describes identification dependency, a specialized kind of existence dependency. The second subsection describes three important relationship patterns: (1) relationships with attributes, (2) self-referencing relationships, and (3) associative entity types representing multiway (M-way) relationships. The final subsection describes an important equivalence between M-N and 1-M relationships.

---

1 Chapter 6 presents conversion rules that describe the differences more precisely.
### 5.2.1 Identification Dependency (Weak Entity Types and Identifying Relationships)

In an ERD, some entity types may not have their own primary key. Entity types without their own primary key must borrow part (or all) of their primary key from other entity types. Entity types that borrow part or their entire primary key are known as weak entity types. A relationship that provides a component of the primary key is known as an identifying relationship. Thus, an identification dependency involves a weak entity type and one or more identifying relationships.

**Weak Entity Type**: an entity type that borrows all or part of its primary key from another entity type. Identifying relationships indicate the entity types that supply components of the borrowed primary key.

Identification dependency occurs because some entities are closely associated with other entities. For example, a room does not have a separate identity from its building because a room is physically contained in a building. You can reference a room only by providing its associated building identifier. In the ERD for buildings and rooms (Figure 5.8), the Room entity type is identification dependent on the Building entity type in the Contains relationship. A solid relationship line indicates an identifying relationship. For weak entity types, the underlined attribute (if present) is part of the primary key, but not the entire primary key. Thus, the primary key of Room is a combination of BldgID and RoomNo. As another example, Figure 5.9 depicts an identification dependency involving the weak entity type State and the identifying relationship Holds.

**Identification dependency symbols**:  
- Solid relationship line for identifying relationships  
- Diagonal lines in the corners of rectangles for weak entity types

![Figure 5.8: Identification Dependency Example](image)

Note: The weak entity type’s cardinality is always (1,1) in each identifying relationship.

![Figure 5.9: Another Identification Dependency Example](image)

Identification dependency is a specialized kind of existence dependency. Recall that an existent-dependent entity type has a mandatory relationship (minimum cardinality of one). Weak entity types are existent dependent on the identifying relationships. In addition to the existence dependency, a weak entity type borrows at least part of its primary key. Because of the existence dependency and the primary key borrowing, the minimum and maximum cardinalities of a weak entity type are always 1.

The next section shows several additional examples of identification dependency in the discussion of associative entity types and M-way relationships. The use of identification dependency is necessary for associative entity types.

### 5.2.2 Relationship Patterns

This section discusses three patterns for relationships that you may encounter in database development efforts: (1)
M-N relationships with attributes, (2) self-referencing (unary) relationships, and (3) associative entity types representing M-way relationships. Although these relationship patterns are not common, they are important when they occur. You need to study these patterns carefully to apply them correctly in database development efforts.

**M-N Relationships with Attributes**

As briefly mentioned in Section 5.1, relationships can have attributes. This situation typically occurs with M-N relationships. In an M-N relationship, attributes are associated with the combination of entity types, not just one of the entity types. If an attribute is associated with only one entity type, then it should be part of that entity type, not the relationship. Figures 5.10 and 5.11 depict M-N relationships with attributes. In Figure 5.10, the attribute EnrGrade is associated with the combination of a student and offering, not either one alone. For example, the EnrollsIn relationship records the fact that the student with StdNo 123-77-9993 has a grade of 3.5 in the offering with offer number 1256. In Figure 5.11(a), the attribute Qty represents the quantity of a part supplied by a given supplier. In Figure 5.11(b), the attribute AuthOrder represents the order in which the author’s name appears in the title of a book. To reduce clutter on a large diagram, relationship attributes may not be shown.

![Figure 5.10: M-N Relationship with an Attribute](image)

![Figure 5.11: Additional M-N Relationships with Attributes](image)

1-M relationships also can have attributes, but 1-M relationships with attributes are much less common than M-N relationships with attributes. In Figure 5.12, the Commission attribute is associated with the Lists relationship, not with either the Agent or the Home entity type. A home will only have a commission if an agent lists it. Typically, 1-M relationships with attributes are optional for the child entity type. The Lists relationship is optional for the Home entity type.

![Figure 5.12: 1-M Relationship with an Attribute](image)

**Self-Referencing (Unary) Relationships**

A self-referencing relationship involves connections among members of the same set. Self-referencing relationships are sometimes called reflexive relationships because they are like a reflection in a mirror. Figure 5.13 displays two self-referencing relationships involving the Faculty and Course entity types. Both relationships involve two entity types that are the same (Faculty for Supervises and Course for PreReqTo). These relationships depict important concepts in a university database. The Supervises relationship depicts an organizational chart, while the PreReqTo relationship depicts course dependencies that can affect a student’s course planning.
**Self-Referencing Relationship**: a relationship involving the same entity type. Self-referencing relationships represent associations among members of the same set.

![Figure 5.13: Examples of Self-Referencing (Unary) Relationships](image)

For self-referencing relationships, it is important to distinguish between 1-M and M-N relationships. An instance diagram can help you understand the difference. Figure 5.14(a) shows an instance diagram for the `Supervises` relationship. Notice that each faculty can have at most one superior. For example, Faculty2 and Faculty3 have Faculty1 as a superior. Therefore, `Supervises` is a 1-M relationship because each faculty can have at most one supervisor. In contrast, there is no such restriction in the instance diagram for the `PreReqTo` relationship (Figure 5.14(b)). For example, IS461 has two prerequisites (IS480 and IS460), while IS320 is a prerequisite to both IS480 and IS460. Therefore, `PreReqTo` is an M-N relationship because a course can be a prerequisite to many courses, and a course can have many prerequisites.

![Figure 5.14: Instance Diagrams for Self-Referencing Relationships](image)

Self-referencing relationships occur in a variety of business situations. Any data that can be visualized like Figure 5.14 can be represented as a self-referencing relationship. Typical examples include hierarchical charts of accounts, genealogical charts, part designs, and transportation routes. In these examples, self-referencing relationships are an important part of the database.

There is one other noteworthy aspect of self-referencing relationships. Sometimes a self-referencing relationship is not needed. For example, if you only want to know whether an employee is a supervisor, a self-referencing relationship is not needed. Rather, an attribute can be used to indicate whether an employee is a supervisor.

**Associative Entity Types Representing Multi-Way (M-Way) Relationships**

Some ERD notations support relationships involving more than two entity types known as **M-way (multiway) relationships** where the \( M \) means more than two. For example, the Chen\(^2\) ERD notation (with diamonds for relationships) allows relationships to connect more than two entity types, as depicted in Figure 5.15. The `Uses` relationship lists suppliers and parts used on projects. For example, a relationship instance involving Supplier1, Part1, and Project1 indicates that Supplier1 Supplies Part1 on Project1. An M-way relationship involving three entity types is called a **ternary relationship**. The letters in the Chen ERD indicate maximum cardinalities.

---

\(^2\) The Chen notation is named after Dr. Peter Chen, who published the paper defining the Entity Relationship Model in 1976.
Although you cannot directly represent M-way relationships in the Crow’s Foot notation, you should understand how to indirectly represent them. You use an associative entity type and a collection of identifying 1-M relationships to represent an M-way relationship. In Figure 5.16, three 1-M relationships link the associative entity type, Uses, to the Part, the Supplier, and the Project entity types. The Uses entity type is associative because its role is to connect other entity types. Because associative entity types provide a connecting role, they are sometimes given names using active verbs. In addition, associative entity types are always weak as they must borrow the entire primary key. For example, the Uses entity type obtains its primary key through the three identifying relationships.

**Associative Entity Type:** a weak entity that depends on two or more entity types for its primary key. An associative entity type with more than two identifying relationships is known as an M-way associative entity type.

As another example, Figure 5.17 shows the associative entity type Provides that connects the Employee, Skill, and Project entity types. An example instance of the Provides entity type contains Employee1 providing Skill1 on Project1.

The issue of when to use an M-way associative entity type (i.e., an associative entity type representing an M-way relationship) can be difficult to understand. If a database only needs to record pairs of facts, an M-way associative entity type is not needed. For example, if a database only needs to record who supplies a part and what projects use a part, then an M-way associative entity type should not be used. In this case, there should be binary relationships between
Supplier and Part and between Project and Part. You should use an M-way associative entity type when the database should record combinations of three (or more) entities rather than just combinations of two entities. For example, if a database needs to record which supplier provides parts on specific projects, an M-way associative entity type is needed. Because of the complexity of M-way relationships, Chapter 7 provides a way to reason about them using constraints, while Chapter 12 provides a way to reason about them using data entry forms.

5.2.3 Equivalence between 1-M and M-N Relationships

To improve your understanding of M-N relationships, you should know an important equivalence for M-N relationships. An M-N relationship can be replaced by an associative entity type and two identifying 1-M relationships. Figure 5.18 shows the EnrollsIn (Figure 5.10) relationship converted to this 1-M style. In Figure 5.18, two identifying relationships and an associative entity type replace the EnrollsIn relationship. The relationship name (EnrollsIn) has been changed to a noun (Enrollment) to follow the convention of nouns for entity type names. The 1-M style is similar to the representation in a relational database diagram. If you feel more comfortable with the 1-M style, then use it. In terms of the ERD, the M-N and 1-M styles have the same meaning.

| Relationship Equivalence: an M-N relationship can be replaced by an associative entity type and two identifying 1-M relationships. In most cases the choice between a M-N relationship and the associative entity type is personal preference. |

![Figure 5.18: EnrollsIn M-N Relationship (Figure 5.10) Transformed into 1-M Relationships](image)

The transformation of an M-N relationship into 1-M relationships is similar to representing an M-way relationship using 1-M relationships. Whenever an M-N relationship is represented as an associative entity type and two 1-M relationships, the new entity type is identification dependent on both 1-M relationships, as shown in Figure 5.18. Similarly, when representing M-way relationships, the associative entity type is identification dependent on all 1-M relationships as shown in Figures 5.16 and 5.17.

There is one situation when the 1-M style is preferred to the M-N style. When an M-N relationship must be related to other entity types in relationships, you should use the 1-M style. For example, assume that in addition to enrollment in a course offering, attendance in each class session should be recorded. In this situation, the 1-M style is preferred because it is necessary to link an enrollment with attendance records. Figure 5.19 shows the Attendance entity type added to the ERD of Figure 5.18. Note that an M-N relationship between the Student and Offering entity types would not have allowed another relationship with Attendance.

![Figure 5.19: Attendance Entity Type Added to the ERD of Figure 18](image)
Figure 5.19 provides other examples of identification dependencies. Attendance is identification dependent on Enrollment in the RecordedFor relationship. The primary key of Attendance consists of AtDate along with the primary key of Enrollment. Similarly, Enrollment is identification dependent on both Student and Offering. The primary key of Enrollment is a combination of StdNo and OfferNo.

5.3 Classification in the Entity Relationship Model

People classify entities to better understand their environment. For example, animals are classified into mammals, reptiles, and other categories to understand the similarities and differences among different species. In business, classification is also pervasive. Classification can be applied to investments, employees, customers, loans, parts, and so on. For example, when applying for a home mortgage, an important distinction is between fixed- and adjustable-rate mortgages. Within each kind of mortgage, there are many variations distinguished by features such as the repayment period, the prepayment penalties, and the loan amount.

This section describes ERD notation to support classification. You will learn to use generalization hierarchies, specify cardinality constraints for generalization hierarchies, and use multiple-level generalization hierarchies for complex classifications.

5.3.1 Generalization Hierarchies

Generalization hierarchies allow entity types to be related by the level of specialization. Figure 5.20 depicts a generalization hierarchy to classify employees as salaried versus hourly. Both salaried and hourly employees are specialized kinds of employees. The Employee entity type is known as the supertype (or parent). The entity types SalaryEmp and HourlyEmp are known as the subtypes (or children). Because each subtype entity is a supertype entity, the relationship between a subtype and supertype is known as ISA. For example, a salaried employee is an employee. Because the relationship name (ISA) is always the same, it is not shown on the diagram.

Inheritance supports sharing between a supertype and its subtypes. Because every subtype entity is also a supertype entity, the attributes of the supertype also apply to all subtypes. For example, every entity of SalaryEmp has an employee number, name, and hiring date because it is also an entity of Employee. Inheritance means that the attributes of a supertype are automatically part of its subtypes. That is, each subtype inherits the attributes of its supertype. For example, the attributes of the SalaryEmp entity type are its direct attribute (EmpSalary) and its inherited attributes from Employee (EmpNo, EmpName, EmpHireDate, etc.). Inherited attributes are not shown in an ERD. Whenever you have a subtype, assume that it inherits the attributes from its supertype.

Inheritance: a data modeling feature that supports sharing of attributes between a supertype and a subtype. Subtypes inherit attributes from their supertypes.
5.3.2 Disjointness and Completeness Constraints

Generalization hierarchies do not show cardinalities because they are always the same. Rather disjointness and completeness constraints can be shown. **Disjointness** means that subtypes in a generalization hierarchy do not have any entities in common. In Figure 5.21, the generalization hierarchy is disjoint because a security cannot be both a stock and a bond. In contrast, the generalization hierarchy in Figure 5.22 is not disjoint because teaching assistants can be considered both students and faculty. Thus, the set of students overlaps with the set of faculty. **Completeness** means that every entity of a supertype must be an entity in one of the subtypes in the generalization hierarchy. The completeness constraint in Figure 5.21 means that every security must be either a stock or a bond.

![Figure 5.21: Generalization Hierarchy for Securities](image)

![Figure 5.22: Generalization Hierarchy for University People](image)

Some generalization hierarchies lack both disjointness and completeness constraints. In Figure 5.20, the lack of a disjointness constraint means that some employees can be both salaried and hourly. The lack of a completeness constraint indicates that some employees are not paid by salary or the hour (perhaps by commission).

5.3.3 Multiple Levels of Generalization

Generalization hierarchies can be extended to more than one level. This practice can be useful in disciplines such as investments where knowledge is highly structured. In Figure 5.23, there are two levels of subtypes beneath securities. Inheritance extends to all subtypes, direct and indirect. Thus, both the **Common** and **Preferred** entity types inherit the attributes of **Stock** (the immediate parent) and **Security** (the indirect parent). Note that disjointness and completeness constraints can be made for each group of subtypes.

5.4 Notation Summary and Diagram Rules

You have seen a lot of ERD notation in the previous sections of this chapter. So that you do not become overwhelmed, this section provides a convenient summary as well as rules to help you avoid common diagramming errors.

5.4.1 Notation Summary

To help you recall the notation introduced in previous sections, Table 5-2 presents a summary while Figure 5.24 demonstrates the notation for the university database of Chapter 4. Figure 5.24 differs in some ways from the university
database in Chapter 4 to depict most of the Crow’s Foot notation. Figure 5.24 contains a generalization hierarchy to depict the similarities among students and faculty. You should note that the primary key of the Student and the Faculty entity types is PerNo, an attribute inherited from the UnivPerson entity type. The Enrollment entity type (associative) and the identifying relationships (Registers and Grants) could appear as an M-N relationship as previously shown in Figure 5.10. In addition to these issues, Figure 5.24 omits some attributes for brevity.

Figure 5.23: Multiple Levels of Generalization Hierarchies

![Diagram](image)

Table 5-2: Summary of Crow’s Foot Notation

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student Standard</td>
<td>Entity type with attributes (primary key underlined)</td>
</tr>
<tr>
<td>Enrolls_In</td>
<td>M-N relationship with attributes: attributes are shown if room permits; otherwise attributes are listed separately.</td>
</tr>
<tr>
<td>Contains</td>
<td>Identification dependency: identifying relationship(s) (solid relationship lines) and weak entity (diagonal lines in the corners of the rectangle). Associative entity types also are weak because they are (by definition) identification dependent. Generalization hierarchy with disjointness and completeness constraints</td>
</tr>
<tr>
<td>D,C</td>
<td>Existence dependent cardinality (minimum cardinality of 1): inner symbol is a line perpendicular to the relationship line.</td>
</tr>
<tr>
<td>Optional cardinality (minimum cardinality of 0): inner symbol is a circle.</td>
<td></td>
</tr>
<tr>
<td>Single-valued cardinality (maximum cardinality of 1): outer symbol is a perpendicular line.</td>
<td></td>
</tr>
</tbody>
</table>
As you develop an ERD, you should remember that an ERD contains business rules that enforce organizational policies and promote efficient communication among business stakeholders. An ERD contains important business rules represented as primary keys, relationships, cardinalities, and generalization hierarchies. Primary keys support entity identification, an important requirement in business communication. Identification dependency involves an entity that depends on other entities for identification, a requirement in some business communication. Relationships indicate direct connections among units of business communication. Cardinalities restrict the number of related entities in relationships supporting organizational policies and consistent business communication. Generalization hierarchies with disjointness and completeness constraints support classification of business entities and organizational policies. Thus, the elements of an ERD are crucial for enforcement of organizational policies and efficient business communication.

For additional kinds of business constraints, an ERD can be enhanced with informal documentation or a formal rules language. Since the SQL standard supports integrity constraints in the CREATE TABLE statement (see Chapter 3) for simple rules and a formal rules language (see Chapters 11 and 14) for complex constraints, a language is not proposed here for ERDs. In the absence of a formal rules language, business rules can be stored as informal documentation associated with entity types, attributes, and relationships. Typical kinds of business rules to specify as informal documentation are candidate key constraints, attribute comparison constraints, null value constraints, and default values. Candidate keys provide alternative ways to identify business entities. Attribute comparison constraints restrict the values of attributes either to a fixed collection of values or to values of other attributes. Null value constraints and default values support policies about completeness of data collection activities. Table 5-3 summarizes the common kinds of business rules that can be specified either formally or informally in an ERD.

5.4.2 Diagram Rules

To provide guidance about correct usage of the notation, Table 5-4 presents completeness and consistency rules. You should apply these rules when completing an ERD to ensure that there are no notation errors in your ERD. Thus, the diagram rules serve a purpose similar to syntax rules for a computer language. The absence of syntax errors does not mean that a computer program performs its tasks correctly. Likewise, the absence of notation errors does not mean that an ERD provides an adequate data representation. The diagram rules do not ensure that you have considered multiple alternatives, correctly represented user requirements, and properly documented your design. Chapter 6 discusses these issues to enhance your data modeling skills.
Table 5-3: Summary of Business Rules in an ERD

<table>
<thead>
<tr>
<th>Business Rule</th>
<th>ERD Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entity identification</td>
<td>Primary keys for entity types, identification dependency (weak entities and identifying relationships), informal documentation about other unique attributes</td>
</tr>
<tr>
<td>Connections among business entities</td>
<td>Relationships</td>
</tr>
<tr>
<td>Number of related entities</td>
<td>Minimum and maximum cardinalities</td>
</tr>
<tr>
<td>Inclusion among entity sets</td>
<td>Generalization hierarchies</td>
</tr>
<tr>
<td>Reasonable values</td>
<td>Informal documentation about attribute constraints (comparison to constant values or other attributes)</td>
</tr>
<tr>
<td>Data collection completeness</td>
<td>Informal documentation about null values and default values</td>
</tr>
</tbody>
</table>

Table 5-4: Completeness and Consistency Rules

<table>
<thead>
<tr>
<th>Type of Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Completeness</strong></td>
<td>1. Primary key rule: All entity types have a primary key (direct, borrowed, or inherited).</td>
</tr>
<tr>
<td></td>
<td>2. Naming rule: All entity types, relationships, and attributes are named.</td>
</tr>
<tr>
<td></td>
<td>3. Cardinality rule: Cardinality is given for both entity types in a relationship.</td>
</tr>
<tr>
<td></td>
<td>4. Entity participation rule: All entity types except those in a generalization hierarchy participate in at least one relationship.</td>
</tr>
<tr>
<td></td>
<td>5. Generalization hierarchy participation rule: Each generalization hierarchy participates in at least one relationship with an entity type not in the generalization hierarchy.</td>
</tr>
<tr>
<td><strong>Consistency</strong></td>
<td>1. Entity name rule: Entity type names are unique.</td>
</tr>
<tr>
<td></td>
<td>2. Attribute name rule: Attribute names are unique within entity types and relationships.</td>
</tr>
<tr>
<td></td>
<td>3. Inherited attribute name rule: Attribute names in a subtype do not match inherited (direct or indirect) attribute names.</td>
</tr>
<tr>
<td></td>
<td>4. Relationship/entity type connection rule: All relationships connect two entity types (not necessarily distinct).</td>
</tr>
<tr>
<td></td>
<td>5. Relationship/relationship connection rule: Relationships are not connected to other relationships.</td>
</tr>
<tr>
<td></td>
<td>6. Weak entity type rule: Weak entity types have at least one identifying relationship.</td>
</tr>
<tr>
<td></td>
<td>7. Identifying relationship rule: For each identifying relationship, at least one participating entity type must be weak.</td>
</tr>
<tr>
<td></td>
<td>8. Identification dependency cardinality rule: For each identifying relationship, the minimum and maximum cardinality must be 1 in the direction from the child (weak entity) to the parent entity type.</td>
</tr>
<tr>
<td></td>
<td>9. Redundant foreign key rule: Redundant foreign keys are not used.</td>
</tr>
</tbody>
</table>
Most of the rules in Table 5-4 do not require much elaboration. The first three completeness rules and the first five consistency rules are simple to understand. Even though the rules are simple, you should still check your ERDs for compliance as it is easy to overlook a violation in a moderate-size ERD.

The consistency rules do not require unique relationship names because participating entity types provide a context for relationship names. However, it is good practice to use unique relationship names as much as possible to make relationships easy to distinguish. In addition, two or more relationships involving the same entity types must be unique because the entity types no longer provide a context to distinguish the relationships. Since it is uncommon to have more than one relationship between the same entity types, the consistency rules do not include this provision.

Completeness rules 4 (entity participation rule) and 5 (generalization hierarchy participation rule) require elaboration. Violating these rules is a warning, not necessarily an error. In most ERDs, all entity types not in a generalization hierarchy and all generalization hierarchies are connected to at least one other entity type. In rare situations, an ERD contains an unconnected entity type just to store a list of entities. Rule 5 applies to an entire generalization hierarchy, not to each entity type in a generalization hierarchy. In other words, at least one entity type in a generalization hierarchy should be connected to at least one entity type not in the generalization hierarchy. In many generalization hierarchies, multiple entity types participate in relationships. Generalization hierarchies permit subtypes to participate in relationships thus constraining relationship participation. For example in Figure 5.24, Student and Faculty participate in relationships.

Consistency rules 6 through 9 involve common errors in the ERDs of novice data modelers. Novice data modelers violate consistency rules 6 to 8 because of the complexity of identification dependency. Identification dependency, involving a weak entity type and identifying relationships, provides more sources of errors than other parts of the Crow’s Foot notation. In addition, each identifying relationship also requires a minimum and maximum cardinality of 1 in the direction from the child (weak entity type) to the parent entity type. Novice data modelers violate consistency rule 9 (redundant foreign key rule) because of confusion between an ERD and the relational data model. The conversion process transforms 1-M relationships into foreign keys.

Example of Rule Violations and Resolutions

Because the identification dependency rules and the redundant foreign key rule are a frequent source of errors to novice designers, this section provides an example to depict rule violations and resolutions. Figure 5.25 demonstrates violations of the identification dependency rules (consistency rules 6 to 9) and the redundant foreign key rule (consistency rule 9) for the university database ERD. The following list explains the violations:

---

Figure 5.25: ERD with Violations of Consistency Rules 6 to 9
• **Consistency rule 6** (weak entity type rule) violation: *Faculty* cannot be a weak entity type without at least one identifying relationship.

• **Consistency rule 7** (identifying relationship rule) violation: The *Has* relationship is identifying but neither *Offering* nor *Course* is a weak entity type.

• **Consistency rule 8** (identification dependency cardinality rule) violation: The cardinality of the *Registers* relationship from *Enrollment* to *Student* should be (1, 1) not (0, Many).

• **Consistency rule 9** (redundant foreign key rule) violation: The *CourseNo* attribute in the *Offering* entity type is redundant with the *Has* relationship. Because *CourseNo* is the primary key of *Course*, it should not be an attribute of *Offering* to link an *Offering* to a *Course*. The *Has* relationship provides the linkage to *Course*.

For most rules, resolving violations is easy. The major task is recognition of the violation. For the identification dependency rules, resolution can depend on the problem details. The following list suggests possible corrective actions for diagram errors:

• **Consistency rule 6** (weak entity type rule) resolution: The problem can be resolved by either adding one or more identifying relationships or by changing the weak entity type into a regular entity type. In Figure 5.25, the problem is resolved by making *Faculty* a regular entity type. The more common resolution is to add one or more identifying relationships.

• **Consistency rule 7** (identifying relationship rule) resolution: The problem can be resolved by adding a weak entity type or making the relationship non-identifying. In Figure 5.25, the problem is resolved by making the *Has* relationship non-identifying. If there is more than one identifying relationship involving the same entity type, the typical resolution involves designating the common entity type as a weak entity type.

• **Consistency rule 8** (identification dependency cardinality rule) resolution: The problem can be resolved by changing the weak entity type’s cardinality to (1,1). Typically, the cardinality of the identifying relationship is reversed. In Figure 5.25, the cardinality of the *Registers* relationship should be reversed (1:1 near *Student* and 0:Many near *Enrollment*).

• **Consistency rule 9** (redundant foreign key rule) resolution: Normally the problem can be resolved by removing the redundant foreign key. In Figure 5.25, *CourseNo* should be removed as an attribute of *Offering*. In some cases, the attribute may not represent a foreign key. If the attribute does not represent a foreign key, it should be renamed instead of removed.

**Alternative Organization of Rules**

The organization of rules in Table 5-4 may be difficult to remember. Table 5-5 provides an alternative grouping by rule purpose. If you find this organization more intuitive, you should use it. However you choose to remember the rules, the important point is to apply them after you have completed an ERD. To help you apply diagram rules, most CASE tools perform checks specific to the notations supported by the tools.

These rules can be supported in CASE tools with data modeling features although support for structural rules is uneven in commercial products. Consistency rules 4 and 5 can be supported through diagram construction. Relationships must be connected to two entity types (not necessarily distinct) prohibiting violations of consistency rules 4 and 5. For the other completeness and consistency rules, an analysis tool can generate a report of rule violations. However, the analysis tool would not require fixing rule violations in an ERD because some rules are soft and the rules may be applied before an ERD is complete.

For the redundant foreign key rule (consistency rule 9), an analysis tool may use a simple implementation to determine if an ERD contains a redundant foreign key. The analysis tool can check the child entity type (entity type on the many side of the relationship) for an attribute with the same name and data type as the primary key in the parent entity type (entity type on the one side of the relationship). If the tool finds an attribute with the same name and data type, a violation is listed in the rule violation report.
### Table 5-5: Alternative Rule Organization

<table>
<thead>
<tr>
<th>Category</th>
<th>Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Names</strong></td>
<td>All entity types, relationships, and attributes are named. (Completeness rule 2)</td>
</tr>
<tr>
<td></td>
<td>Entity type names are unique. (Consistency rule 1)</td>
</tr>
<tr>
<td></td>
<td>Attribute names are unique within entity types and relationships. (Consistency rule 2)</td>
</tr>
<tr>
<td></td>
<td>Attribute names in a subtype do not match inherited (direct or indirect) attribute names. (Consistency rule 3)</td>
</tr>
<tr>
<td><strong>Content</strong></td>
<td>All entity types have a primary key (direct, borrowed, or inherited). (Completeness rule 1)</td>
</tr>
<tr>
<td></td>
<td>Cardinality is given for both entity types in a relationship. (Completeness rule 3)</td>
</tr>
<tr>
<td><strong>Connection</strong></td>
<td>All entity types except those in a generalization hierarchy participate in at least one relationship. (Completeness rule 4)</td>
</tr>
<tr>
<td></td>
<td>Each generalization hierarchy participates in at least one relationship with an entity type not in the generalization hierarchy. (Completeness rule 5)</td>
</tr>
<tr>
<td></td>
<td>All relationships connect two entity types. (Consistency rule 4)</td>
</tr>
<tr>
<td></td>
<td>Relationships are not connected to other relationships. (Consistency rule 5)</td>
</tr>
<tr>
<td></td>
<td>Redundant foreign keys are not used. (Consistency rule 9)</td>
</tr>
<tr>
<td><strong>Identification</strong></td>
<td>Weak entity types have at least one identifying relationship. (Consistency rule 6)</td>
</tr>
<tr>
<td><strong>Dependency</strong></td>
<td>For each identifying relationship, at least one participating entity type must be weak. (Consistency rule 7)</td>
</tr>
<tr>
<td></td>
<td>For each weak entity type, the minimum and maximum cardinality must equal 1 for each identifying relationship. (Consistency rule 8)</td>
</tr>
</tbody>
</table>

### 5.5 Comparison to Other Notations

The ERD notation presented in this chapter is similar to but not identical to what you may encounter later. There is no standard notation for ERDs. There are perhaps six reasonably popular ERD notations, each having its own small variations that appear in practice. The notation in this chapter comes from the Crow’s Foot stencil in Visio Professional 5 with the addition of the generalization notation. The notations that you encounter in practice will depend on factors such as the data modeling tool (if any) used in your organization and the industry. One thing is certain: you should be prepared to adapt to the notation in use. This section describes ERD variations that you may encounter and notations supported by several CASE tools, along with the Class Diagram notation of the Unified Modeling Language (UML), an emerging standard for data modeling.

#### 5.5.1 Range of ERD Variations

Because there is no widely accepted ERD standard, different symbols can be used to represent the same concept. Relationship cardinalities are a source of wide variation. You should pay attention to the placement of the cardinality symbols. The notation in this chapter places the symbols close to the “far” entity type, while other notations place the cardinality symbols close to the “near” entity type. The notation in this chapter uses a visual representation of cardinalities with the minimum and maximum cardinalities given by three symbols. Other notations use a text representation with letters and integers instead of symbols. For example, Figure 5.26 shows a Chen ERD containing maximum cardinalities (upper case letters) and relationships denoted by diamonds. In variations to the original ERD notation, minimum cardinalities were added to the notation.
CHAPTER 5 - UNDERSTANDING ENTITY RELATIONSHIP DIAGRAMS

Figure 5.26: Chen Notation for the Course-Offering ERD

Other symbol variations are visual representations for certain kinds of entity types. In some notations, weak entity types and M-N relationships have special representations. Weak entity types are sometimes enclosed in double rectangles. Identifying relationships are sometimes enclosed in double diamonds. M-N relationships with attributes are sometimes shown as a rectangle with a diamond inside denoting the dual qualities (both relationship and entity type).

In addition to symbol variations, there are also rule variations, as shown in the following list. For each restriction, there is a remedy. For example, if only binary relationships are supported, M-way relationships must be represented as an associative entity type with 1-M relationships.

- Some notations do not support M-way relationships.
- Some notations do not support M-N relationships.
- Some notations do not support relationships with attributes.
- Some notations do not support self-referencing (unary) relationships.
- Some notations permit relationships to be connected to other relationships.
- Some notations show foreign keys as attributes.
- Some notations allow attributes to have more than one value (multivalued attributes).

Restrictions in an ERD notation do not necessarily make the notation less expressive than other notations without the restrictions. Additional symbols in a diagram may be necessary, but the same concepts can still be represented. For example, the Crow’s Foot notation does not support M-way relationships. However, M-way relationships can be represented using M-way associative entity types. M-way associative entity types require additional symbols than M-way relationships, but the same concepts are represented.

Commercial CASE tools support a variety of ERD notations. Many tools support conversion to commercial DBMSs so the representations typically align to relational databases with foreign keys shown and no support for M-N relationships. The next three subsections depict tools (Aqua Data Studio, Oracle SQL Developer, and Microsoft Visio Professional) with these types of restrictions.

5.5.2 ERD Notation in Aqua Data Studio

The Aqua Data Studio (www.aquafold.com) supports a variety of DBMSs through query and DBA tools along with data modeling support. The data modeling tool in the Aqua Data Studio uses an ERD notation somewhat aligned to relational database representation. Figure 5.27 depicts the university database developed in the data modeling tool of Aqua Data Studio. The notation is similar to the Crow’s Foot notation used in this chapter with solid lines for identifying relationships, dashed lines for normal relationships, and cardinalities represented with the same symbols used in this chapter. However, the data modeling tool does not use the weak entity type symbol. In addition, the data modeling tool shows foreign keys (in asterisks) and does not support M-N relationships.
The data modeling tool in Aqua Data Studio supports generalization hierarchy relationships although in a different representation than presented in this chapter. Figure 5.28 shows the university database extended with generalization hierarchy relationships for university people. The data modeling tool does not support full generalization hierarchies as shown in Figure 5.24. Instead, generalization relationships are shown separately for each pair (supertype, subtype) of entity types. The data modeling tool supports inheritance as the attributes with asterisks in the Student and Faculty entity types are inherited from UnivPerson. For constraints, the data modeling tool supports only one constraint (inclusive or exclusive) for a generalization relationship instead of the two constraints (disjoint and completeness) presented in this chapter.

The data modeling tool in Aqua Data Studio supports other diagramming features besides diagram construction. The Aqua Data Studio supports several levels of detail in diagrams (attribute level with optional display of data types and null value constraints), entity level, primary key level, comment level, and relationship name level. ERDs can be printed, enlarged, distributed on multiple pages, and laminated for use as posters and quick reference. Regions permit grouping of diagram elements into colored areas that can be manipulated as a unit.
CHAPTER 5 - UNDERSTANDING ENTITY RELATIONSHIP DIAGRAMS

5.5.3 ERD Notation in Oracle SQL Developer

In contrast to Aqua Data Studio, the data modeling tool in Oracle SQL Developer uses a less standard ERD notation as shown in Figure 5.29. The Oracle data modeling tool does not provide relationship names and uses arrows for child to parent relationships. Other restrictions in the data modeling tool are no identifying relationships, no M-N relationships, and redundant display of foreign keys. The Oracle data modeling tool also has fewer display and printing options than Aqua Data Studio.

The Oracle data modeling tool supports design rules although not the complete set presented in this chapter. For entity types, the Oracle data modeling tool can detect entity types without relationships, attributes, and a primary key as well as inconsistencies with naming standards. For attributes, the Oracle data modeling tool can detect attributes without a data type and inconsistent naming standards.

5.5.4 Entity Relationship Stencil in Visio Professional

Since Section 2.4.5 provided an overview of Visio Professional, this section will provide details about the Entity Relationship stencil (collection of shapes) available in the Visio 2010 Professional Edition. Using display options for the IDEF1X notation and crow’s foot cardinality symbols, the Entity Relationship stencil supports most of the chapter notation for Crow’s Foot ERDs. The Entity Relationship stencil does not support M-N relationships, uses different symbols for weak entity types, and uses categories instead of generalization hierarchies.

Figure 5.30 depicts the ERD for the extended university database (Figure 5.24) using the IDEF1X notation of Visio Professional. Some foreign keys are not shown in Figure 5.30 although all foreign keys can be shown by setting a display option. Foreign keys that are part of primary keys are shown when the primary keys are shown. Thus, the Student, Faculty, and Enrollment entity types show foreign keys because they are part of the primary keys. The rounded corner shape denotes a weak entity type. The Student and Faculty entity types are weak because they inherit the primary key from the Person entity type. The associative entity type (Enrollment) is necessary because Visio Professional does not support M-N relationships.

In the IDEF1X notation of Visio Professional, a generalization hierarchy is known as a category. A category relates a collection of subtype entities. IDEF1X categories and generalization hierarchies in this chapter differ in several ways as shown in the following list.

- The Visio IDEF1X notation uses a circle with one or two lines below to represent categories.
- The Visio IDEF1X notation allows a parent entity type to have multiple categories. In contrast, the notation in this chapter allows only a single generalization hierarchy for a parent entity type.
- The Visio IDEF1X notation allows a parent entity type to contain a discriminating attribute. A discriminating attribute contains one value for each subtype in the category. The discriminating attribute can be checked in a condition to determine class membership of a parent entity occurrence. This chapter does not use discriminating-
ing attributes although such an attribute can be defined. In Visio Professional, a discriminating attribute is an alternative to a disjointness constraint as long as the attribute does not allow null values.

- The Visio IDEF1X notation supports completeness constraints, but the symbol (double lines below the circle) is different than shown in this chapter.

- Visio Professional does not support inheritance as the attributes of Person are not part of its child entity types (Student and Faculty).

![Figure 5.30: Extended University Database in Visio 2010 Professional Edition](image)

Visio Professional supports most of the diagram rules in Table 5-4 (Section 5.4.2). Visio Professional does not force entity types to participate in a relationship so completeness rules 4 and 5 are not enforced. Visio Professional allows attribute names in a subtype to have the same names as attributes in a parent entity type of a category so consistency rule 3 requiring unique attribute names is not enforced. For consistency rule 9, forbidding redundant foreign keys, Visio Professional requires that a foreign key attribute be defined in an entity type to specify a participating relationship. Visio Professional optionally displays foreign keys.

### 5.5.5 Class Diagram Notation of the Unified Modeling Language

The Unified Modeling Language has become the standard notation for object-oriented modeling. Object-oriented modeling emphasizes objects rather than processes, as emphasized in traditional systems development approaches. In object-oriented modeling, one defines the objects first, followed by the features (attributes and operations) of the objects, and then the dynamic interaction among objects. The UML contains class diagrams, interface diagrams, and interaction diagrams to support object-oriented modeling. The class diagram notation provides an alternative to the ERD notations presented in this chapter.

Class diagrams contain classes (collections of objects), associations (binary relationships) among classes, and object features (attributes and operations). Figure 5.31 shows a simple class diagram containing the Offering and Faculty
classes. The association in Figure 5.31 represents a 1-M relationship. The UML supports role names and cardinalities (minimum and maximum) for each direction in an association. The 0..1 cardinality means that an offering object can be related to a minimum of zero faculty objects and a maximum of one faculty object. Operations are listed below the attributes. Each operation contains a parenthesized list of parameters along with the data type returned by the operation.

```
Attributes
OfferNo: Long
OffTerm: String
OffYear: Integer
OffLocation: String
EnrollmentCount(): Integer
OfferingFull(): Boolean

Operations

Figure 5.31: Simple Class Diagram
```

Associations in the UML are similar to relationships in the Crow’s Foot notation. Associations can represent binary or unary relationships. To represent an M-way relationship, a class and a collection of associations are required. To represent an M-N relationship with attributes, the UML provides the association class to allow associations to have attributes and operations. Figure 5.32 shows an association class that represents an M-N relationship between the Student and the Offering classes. The association class contains the relationship attributes.

```
Attributes
StdNo: Long
StdFirstName: String
StdLastName: String
StdDOB: Date
StdAge(): Integer

Operations

Figure 5.32: Association Class Representing an M-N Relationship with Attributes
```

Unlike most ERD notations, support for generalization was built into the UML from its inception. In most ERD notations, generalization was added as an additional feature after a notation was well established. In Figure 5.33, the large empty arrow denotes a classification of the Student class into Undergraduate and Graduate classes. The UML supports generalization names and constraints. In Figure 5.33, the Status generalization is complete, meaning that every student must be an undergraduate or a graduate student.

```
Attributes
StdNo: Long
StdFirstName: String
StdLastName: String
StdDOB: Date
StdAge(): Integer

Operations

Figure 5.33: Class Diagram with a Generalization Relationship
```
The UML also provides a special symbol for composition relationships, similar to identification dependencies in ERD notations. In a composition relationship, the objects in a child class belong only to objects in the parent class. In Figure 5.34, each \textit{OrdLine} object belongs to one \textit{Order} object. Deletion of a parent object causes deletion of the related child objects. As a consequence, the child objects usually borrow part of their primary key from the parent object. However, the UML does not require this identification dependency.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{composition_diagram.png}
\caption{Class Diagram with a Composition Relationship}
\end{figure}

UML class diagrams provide many other features not presented in this brief overview. The UML supports different kinds of classes to integrate programming language concerns with data modeling concerns. Other kinds of classes include value classes, stereotype classes, parameterized classes, and abstract classes. For generalization, the UML supports additional constraints such as static and dynamic classification and different interpretations of generalization relationships (subtype and subclass). For data integrity, the UML supports the specification of constraints in a class diagram.

You should note that class diagrams are just one part of the UML. To some extent, class diagrams must be understood in the context of object-oriented modeling and the entire UML. You should expect to devote an entire academic term to understanding object-oriented modeling and the UML.

\section*{Closing Thoughts}

This chapter has explained the notation of entity relationship diagrams as a prerequisite to applying entity relationship diagrams in the database development process. Using the Crow’s Foot notation, this chapter described the symbols, important relationship patterns, and generalization hierarchies. The basic symbols are entity types, relationships, attributes, and cardinalities to depict the number of entities participating in a relationship. Four important relationship patterns were described: many-to-many (M-N) relationships with attributes, associative entity types representing M-way relationships, identifying relationships providing primary keys to weak entity types, and self-referencing (unary) relationships. Generalization hierarchies allow classification of entity types to depict similarities among entity types.

To help improve your usage of the Crow’s Foot notation, business rule representations, diagram rules, and comparisons to other notations were presented. This chapter presented formal and informal representation of business rules in an entity relationship diagram to provide an organizational context for entity relationship diagrams. The diagram rules involve completeness and consistency requirements. The diagram rules ensure that an ERD does not contain obvious errors. To broaden your background of ERD notations, this chapter presented common diagram variations in several CASE tools as well as the Class Diagram notation of the Unified Modeling Language, the standard notation for object-oriented modeling.

This chapter emphasized the notation of ERDs to provide a solid foundation for the more difficult study of applying the notation on business problems. To master data modeling, you need to understand the ERD notation and obtain ample practice building ERDs. Chapter 6 emphasizes the practice of building ERDs for business problems. Applying the notation involves consistent and complete representation of user requirements, generation of alternative designs, and documentation of design decisions. In addition to these skills, Chapter 6 presents rules to convert an ERD into a table design. With careful study, Chapters 5 and 6 provide a solid foundation to perform data modeling on business databases.
Review Concepts

- Basic concepts: entity types, relationships, and attributes
- Minimum and maximum cardinalities to constrain relationship participation
- Classification of cardinalities as optional, mandatory, and functional
- Existence dependency for entities that cannot be stored without storage of related entities
- Identification dependency involving weak entity types and identifying relationships to support entity types that borrow at least part of their primary keys
- M-N relationships with attributes: attributes are associated with the combination of entity types, not just with one of the entity types
- Equivalence between an M-N relationship and an associative entity type with identifying 1-M relationships
- M-way associative entity types to represent M-way relationships among more than two entity types
- Self-referencing (unary) relationships to represent associations among entities of the same entity type
- Instance diagrams to help distinguish between 1-M and M-N self-referencing relationships
- Generalization hierarchies to show similarities among entity types
- Representation of business rules in an ERD: entity identification, connections among business entities, number of related entities, inclusion among entity sets, reasonable values, and data collection completeness
- Diagram rules to prevent obvious data modeling errors
- Common sources of diagram errors: identification dependency and redundant foreign keys
- Support for the diagram rules in data modeling tools through diagram construction and analysis tools
- ERD variations: symbols and diagram rules
- Modified Crow’s Foot notation in the data modeling tools of Aqua Data Studio, Oracle SQL Developer, and Visio 2010 Professional Edition
- Class Diagram notation of the Unified Modeling Language as an alternative to the Entity Relationship Model

Questions

1. What is an entity type?
2. What is an attribute?
3. What is a relationship?
4. What is the natural language correspondence for entity types and relationships?
5. What is the difference between an ERD and an instance diagram?
6. What symbols are the ERD counterparts of foreign keys in the Relational Model?
7. What cardinalities indicate functional, optional, and mandatory relationships?
8. When is it important to convert an M-N relationship into 1-M relationships?
9. How can an instance diagram help to determine whether a self-referencing relationship is a 1-M or an M-N relationship?
10. When should an ERD contain weak entity types?
11. What is the difference between an existence-dependent and a weak entity type?
12. Why is classification important in business?
13. What is inheritance in generalization hierarchies?
14. What is the purpose of disjointness and completeness constraints for a generalization hierarchy?
15. What symbols are used for cardinality in the Crow’s Foot notation?
16. What are the two components of identification dependency?
17. How are M-way relationships represented in the Crow’s Foot notation?
18. What is an associative entity type?
19. What is the equivalence between an M-N relationship and 1-M relationships?
20. What does it mean to say that part of a primary key is borrowed?
21. What is the purpose of the diagram rules?
22. What are the limitations of the diagram rules?
23. What consistency rules are commonly violated by novice data modelers?
24. Why do novice data modelers violate the identification dependency rules (consistency rules 6 through 8)?
25. Why do novice data modelers violate consistency rule 9 about redundant foreign keys?
26. Why should a CASE tool support diagram rules?
27. How does a data modeling tool typically support consistency rules 4 and 5?
28. How can a data modeling tool support all rules except consistency rules 4 and 5?
29. Why should an analysis tool not require resolution of all diagram errors found in an ERD?
30. How can an analysis tool implement consistency rule 9 about redundant foreign keys?
31. List some symbol differences in ERD notation that you may experience in your career.
32. List some diagram rule differences in ERD notation that you may experience in your career.
33. What is the Unified Modeling Language (UML)?
34. What are the modeling elements in a UML class diagram?
35. What kinds of business rules are formally represented in the Crow’s Foot ERD notation?
36. What kinds of business rules are defined through informal documentation in the absence of a rules language for an ERD?
37. How are M-way relationships represented in the Crow’s Foot notation?
38. What is a self-referencing relationship?
39. What tool can be useful to distinguish between a 1-M and M-N self-referencing relationship?
40. Please explain the importance of specialized modeling elements including M-way relationships and self-referencing relationships.
41. What is the difference between a weak entity type and an associative entity type?
42. What are the differences between the basic Crow’s Foot notation (without generalization support) and the notation supported in the data modeling tool of Aqua Data Studio?
43. What are the differences between the basic Crow’s Foot notation (without generalization support) and the notation supported in the data modeling tool of Oracle SQL Developer?
44. What diagram rules presented in Section 5.4.2 are supported by the data modeling tool of the Oracle SQL Developer?
45. What are the differences between the basic Crow’s Foot notation and the notation supported in the Entity Relationship stencil of Visio Professional?
46. What diagram rules presented in Section 5.4.2 are supported by the Entity Relationship stencil in Visio Professional?

**Problems**

The problems emphasize correct usage of the Crow’s Foot notation and application of the diagram rules. This emphasis is consistent with the pedagogy of the chapter. The more challenging problems in Chapter 6 emphasize user requirements, diagram transformations, design documentation, and schema conversion. To develop a good understanding of data modeling, you should complete the problems in both chapters.
1. Draw an ERD containing the Order and Customer entity types connected by a 1-M relationship from Customer to Order. Choose an appropriate relationship name using your common knowledge of interactions between customers and orders. Define minimum cardinalities so that an order is optional for a customer and a customer is mandatory for an order. For the Customer entity type, add attributes CustNo (primary key), CustFirstName, CustLastName, CustStreet, CustCity, CustState, CustZip, and CustBal (balance). For the Order entity type, add attributes for the OrdNo (primary key), OrdDate, OrdName, OrdStreet, OrdCity, OrdState, and OrdZip. If you are using a data modeling tool that supports data type specification, choose appropriate data types for the attributes based on your common knowledge.

2. Extend the ERD from problem 1 with the Employee entity type and a 1-M relationship from Employee to Order. Choose an appropriate relationship name using your common knowledge of interactions between employees and orders. Define minimum cardinalities so that an employee is optional to an order and an order is optional to an employee. For the Employee entity type, add attributes EmpNo (primary key), EmpFirstName, EmpLastName, EmpPhone, EmpEmail, EmpCommRate (commission rate), and EmpDeptName. If you are using a data modeling tool that supports data type specification, choose appropriate data types for the attributes based on your common knowledge.

3. Extend the ERD from problem 2 with a self-referencing 1-M relationship involving the Employee entity type. Choose an appropriate relationship name using your common knowledge of organizational relationships among employees. Define minimum cardinalities so that the relationship is optional in both directions.

4. Extend the ERD from problem 3 with the Product entity type and an M-N relationship between Product and Order. Choose an appropriate relationship name using your common knowledge of connections between products and orders. Define minimum cardinalities so that an order is optional to a product, and a product is mandatory to an order. For the Product entity type, add attributes ProdNo (primary key), ProdName, ProdQOH, ProdPrice, and ProdNextShipDate. For the M-N relationship, add an attribute for the order quantity. If you are using a data modeling tool that supports data type specification, choose appropriate data types for the attributes based on your common knowledge.

5. Revise the ERD from problem 4 by transforming the M-N relationship into an associative entity type and two identifying, 1-M relationships.

6. Check your ERDs from problems 4 and 5 for violations of the diagram rules. If you followed the problem directions, your diagrams should not have any errors.

7. Using your corrected ERD from problem 6, add violations of consistency rules 6 to 9.

8. Design an ERD for the Task entity type and an M-N self-referencing relationship. For the Task entity type, add attributes TaskNo (primary key), TaskDesc, TaskEstDuration, TaskStatus, TaskStartTime, and TaskEndTime. Choose an appropriate relationship name using your common knowledge of precedence connections among tasks. Define minimum cardinalities so that the relationship is optional in both directions.

9. Revise the ERD from problem 8 by transforming the M-N relationship into an associative entity type and two identifying, 1-M relationships.

10. Define a generalization hierarchy containing the Student entity type, the UndStudent entity type, and the GradStudent entity type. The Student entity type is the supertype and UndStudent and GradStudent are subtypes. The Student entity type has attributes StdNo (primary key), StdName, StdGender, StdDOB (date of birth), StdEmail, and StdAdmitDate. The UndStudent entity type has attributes UndMajor, UndMinor, and UndClass. The GradStudent entity type has attributes GradAdvisor, GradThesisTitle, and GradAsstStatus (assistantship status). The generalization hierarchy should be complete and disjoint.

11. Define a generalization hierarchy containing the Employee entity type, the Faculty entity type, and the Administrator entity type. The Employee entity type is the supertype and Faculty and Administrator are subtypes. The Employee entity type has attributes EmpNo (primary key), EmpName, EmpGender, EmpDOB (date of birth), EmpPhone, EmpEmail, and EmpHireDate. The Faculty entity type has attributes FacRank, FacPayPeriods, and FacTenure. The Administrator entity type has attributes AdmTitle, AdmContractLength, and AdmAppointmentDate. The generalization hierarchy should be complete and overlapping.

12. Combine the generalization hierarchies from problems 10 and 11. The root of the generalization hierarchy is the UnivPerson entity type. The primary key of UnivPerson is UnivPerNo. The other attributes in the UnivPerson entity type should be the attributes common to Employee and Student. You should rename the attributes to be con-
13. Draw an ERD containing the Patient, Physician, and Visit entity types connected by 1-M relationships from Patient to Visit and Physician to Visit. Choose appropriate names for the relationships. Define minimum cardinalities so that patients and physicians are mandatory for a visit, but visits are optional for patients and physicians. For the Patient entity type, add attributes PatNo (primary key), PatFirstName, PatLastName, PatStreet, PatCity, PatState, PatZip, and PatHealthPlan. For the Physician entity type, add attributes PhyNo (primary key), PhyFirstName, PhyLastName, PhySpecialty, PhyPhone, PhyEmail, PhyHospital, and PhyCertification. For the Visit entity type, add attributes for the VisitNo (primary key), VisitDate, VisitPayMethod (cash, check, or credit card), and VisitCharge. If you are using a data modeling tool that supports data type specification, choose appropriate data types for the attributes based on your common knowledge.

14. Extend the ERD in problem 13 with the Nurse, the Item, and the VisitDetail entity types connected by 1-M relationships from Visit to VisitDetail, Nurse to VisitDetail, and Item to VisitDetail. VisitDetail is a weak entity with the 1-M relationship from Visit to VisitDetail an identifying relationship. Choose appropriate names for the relationships. Define minimum cardinalities so that a nurse is optional for a visit detail, an item is mandatory for a visit detail, and visit details are optional for nurses and items. For the Item entity type, add attributes ItemNo (primary key), ItemDesc, ItemPrice, and ItemType. For the Nurse entity type, add attributes NurseNo (primary key), NurseFirstName, NurseLastName, NurseTitle, NursePhone, NurseSpecialty, and NursePayGrade. For the VisitDetail entity type, add attributes for the DetailNo (part of the primary key) and DetailCharge. If you are a design tool that supports data type specification, choose appropriate data types for the attributes based on your common knowledge.

15. Refine the ERD from problem 14 with a generalization hierarchy consisting of Provider, Physician, and Nurse. The root of the generalization hierarchy is the Provider entity type. The primary key of Provider is ProvNo replacing the attributes PhyNo and NurseNo. The other attributes in the Provider entity type should be the attributes common to Nurse and Physician. You should rename the attributes to be consistent with inclusion in the Provider entity type. The generalization hierarchy should be complete and disjoint.

16. Check your ERD from problem 15 for violations of the diagram rules. If you followed the problem directions, your diagram should not have any errors. Apply the consistency and completeness rules to ensure that your diagram does not have errors.

17. Using your corrected ERD from problem 16, add violations of consistency rules 3 and 6 to 9.

18. For each consistency error in Figure 5.P1, identify the consistency rule violated and suggest possible resolutions of the error. The ERD has generic names so that you will concentrate on finding diagram errors rather than focusing on the meaning of the diagram.
19. For each consistency error in Figure 5.P2, identify the consistency rule violated and suggest possible resolutions of the error. The ERD has generic names to help you concentrate on finding diagram errors rather than focusing on the meaning of the diagram.
20. For each consistency error in Figure 5.P3, identify the consistency rule violated and suggest possible resolutions of the error. The ERD has generic names to help will concentrate on finding diagram errors rather than focusing on the meaning of the diagram.

21. Draw an ERD containing the Employee and Appointment entity types connected by an M-N relationship. Choose an appropriate relationship name using your common knowledge of interactions between employees and appointments. Define minimum cardinalities so that an appointment is optional for an employee and an employee is mandatory for an appointment. For the Employee entity type, add attributes EmpNo (primary key), EmpFirst-
22. Extend the ERD from problem 21 with the Location entity type and a 1-M relationship from Location to Appointment. Choose an appropriate relationship name using your common knowledge of interactions between locations and appointments. Define minimum cardinalities so that a location is optional for an appointment and an appointment is optional for a location. For the Location entity type, add attributes LocNo (primary key), LocBuilding, LocRoomNo, and LocCapacity.

23. Extend the ERD in problem 22 with the Calendar entity type and an M-N relationship from Appointment to Calendar. Choose an appropriate relationship name using your common knowledge of interactions between appointments and calendars. Define minimum cardinalities so that an appointment is optional for a calendar and a calendar is mandatory for an appointment. For the Calendar entity type, add attributes CalNo (primary key), CalDate, and CalHour.

24. Revise the ERD from problem 23 by transforming the M-N relationship between Employee and Appointment into an associative entity type along with two identifying 1-M relationships.

25. Draw an ERD containing Student and Paper entity types connected by 1-M relationships. The Student entity type should have attributes for StdNo (primary key), StdFirstName, StdLastName, StdAdmitSemester, StdAdmitYear, and StdEnrollStatus (full or part-time). The Paper entity type should have attributes for PaperNo (primary key), PaperTitle, PaperSubmitDate, PaperAccepted (yes or no), and PaperType (first, second, proposal, or dissertation). Add a 1-M relationship from Student to Paper.

26. Extend the ERD with an Evaluator entity type and an M-N relationship between Paper and Evaluator. The Evaluator entity type should have attributes for EvalNo (primary key), EvalFirstName, EvalLastName, EvalEmail, and EvalOffice. The M-N relationship should have attributes for EvalDate, EvalLitReview (1 to 5 rating), EvalProbId (1 to 5 rating), EvalTechWriting (1 to 5 rating), EvalModelDev (1 to 5 rating), EvalOverall (1 to 5 rating), and EvalComments.

27. Transform the M-N relationship from problem 26 into an associative entity type and identifying relationships.

28. Transform the M-N relationship from problem 26 into three 1-M relationships from Evaluator to Paper. Each paper can have up to three evaluations. Each relationship should be optional to both evaluators and papers. The five evaluation attributes should be associated with each 1-M relationship.

29. Consider data modeling choices to represent student classifications by program year (freshman, sophomore, junior, and senior) and degree type (undergraduate, masters, and doctorate). Identify data modeling alternatives to support classification of students by program year and degree type. What additional information would you need to decide on the appropriate representation?

30. Construct an ERD to represent employees and positions. For employees, the ERD should record the unique employee number, first name, last name, department name, office number, hire date, and date of birth. For position, the ERD should record the unique position number, position name, and step number. Positions can be classified as management, associate, or other. Management positions have a salary range (minimum and maximum) and a car allowance amount. Associate positions have an hourly rate range. There are no attributes to record for other position types. An employee must hold one position. A position can be held by many employees.

31. Draw an ERD containing the Lab, LabVisit, and Patient entity types connected by 1-M relationships from Lab to LabVisit and Patient to LabVisit. Choose appropriate relationship names using your common knowledge of interactions between labs, lab visits, and patients. Define minimum cardinalities so that a lab is required for a lab visit and a patient is required for a lab visit. For the Lab entity type, add attributes LabNo (primary key), LabName, LabStreet, LabCity, LabState, and LabZip. For the Patient entity type, add attributes PatNo (primary key), PatLastName, PatFirstName, PatDOB (date of birth). For the LabVisit entity type, add attributes for the LVNo (primary key), LVDate, LVProvNo, and optional LVOrdNo (for orders from physicians). If you are using a data modeling tool that supports data type specification, choose appropriate data types for the attributes based on your common knowledge.

32. Augment your ERD from problem 31 with the Specimen entity type. For each specimen collected, the database should record a unique SpecNo, SpecArea (vaginal, cervical, or endocervical), and SpecCollMethod (thin prep or sure path). A 1-M relationship from LabVisit to Specimen. A lab visit must produce at least one specimen. A speci-
men is associated with exactly one lab visit.

33. Augment your ERD from problem 32 with the TestOrder entity type and a relationship between TestOrder and Specimen. Multiple test orders can be created for a specimen, but a specimen does not have a test order until a delay, from hours to days. A test order is created for exactly one specimen. A test order contains a TONo (primary key), TOTestName, TOTestType (HPV, CT/GC, CT, or GC), and TOTestResult (positive, negative, equivocal, or failure). If a test order produces a failure, the specimen is given a new test order and tested again until a non-failure result is obtained.

34. Augment your ERD from problem with the Supply entity type and a relationship between TestOrder and Supply. A test can use a collection of supplies (0 or more) and a supply can be used on a collection of tests (0 or more). The Supply entity type contains SuppNo (primary key), SuppName, SuppLotNo, and SuppQOH.

References for Further Study

Four specialized books on database design are Batini, Ceri, and Navathe (1992), Nijssen and Halpin (1989), Teorey et al. (2005), and Carlis and Maguire (2001). The DevX Database Zone (www.devx.com) has practical advice about database development and data modeling. If you would like more details about the UML, consult the UML Resource Page of the Object Management Group (www.uml.org).
Overview

As the first chapter in Part 5 of the textbook, this chapter builds on material covered in Chapter 4. Chapter 4 provided a foundation for query formulation using SQL. Most importantly, you learned an important subset of the SELECT statement and usage of the SELECT statement for problems involving joins and grouping. This chapter extends your knowledge of query formulation to advanced matching problems. To solve these advanced matching problems, additional parts of the SELECT statement are introduced.

This chapter continues with the learning approaches of Chapter 4: provide many examples to imitate and problem-solving guidelines to help you reason through difficult problems. You first will learn to formulate problems involving the outer join operator using new keywords in the FROM clause. Next you will learn to recognize nested queries and apply them to formulate problems involving the join and difference operators. Then you will learn to recognize problems involving the division operator and formulate them using the GROUP BY clause, nested queries in the HAVING clause, and the COUNT function. You will then learn the effect of null values on simple conditions, compound conditions with logical operators, aggregate calculations, and grouping. Finally, you will learn about problems involving hierarchically structured data and SQL extensions (both standard and proprietary) to formulate queries.

The presentation in this chapter covers additional features in Core SQL:2011, especially features not part of SQL-92. All examples execute in recent versions of Microsoft Access (2002 and beyond) and Oracle (9i and beyond) except where noted.
9.1 Outer Join Problems

One of the powerful but sometimes confusing aspects of the SELECT statement is the number of ways to express a join. In Chapter 4, you formulated joins using the cross product style and the join operator style. In the cross product style, you list the tables in the FROM clause and the join conditions in the WHERE clause. In the join operator style, you write join operations directly in the FROM clause using the INNER JOIN and ON keywords.

The major advantage of the join operator style is that problems involving the outer join operator can be formulated. Outer join problems cannot be formulated with the cross product style except with proprietary SQL extensions. This section demonstrates the join operator style for outer join problems and combinations of inner and outer joins. In addition, the proprietary outer join extension of older Oracle versions (8i and previous versions) is shown in Appendix 9C. For your reference, the relationship diagram of the university database is repeated from Chapter 4 (see Figure 9.1).

9.1.1 SQL Support for Outer Join Problems

A join between two tables generates a table with the rows that match on the join column(s). The outer join operator generates the join result (the matching rows) plus the non-matching rows. A one-sided outer join generates a new table with the matching rows plus the non-matching rows from one of the input tables. For example, it can be useful to see all offerings listed in the output even if an offering does not have an assigned faculty.

One-Sided Outer Join: an operator that generates the join result (the matching rows) plus the non-matching rows from one of the input tables. SQL supports the one-sided outer join operator through the LEFT JOIN and RIGHT JOIN keywords.

SQL uses the LEFT JOIN and RIGHT JOIN keywords¹ to specify a one-sided outer join. The LEFT JOIN keyword creates a result table containing the matching rows and the non-matching rows of the left table. The RIGHT JOIN keyword creates a result table containing the matching rows and the non-matching rows of the right table. Thus, the result of a one-sided outer join depends on the direction (RIGHT or LEFT) and the position of the table names. Examples 9.1 and 9.2 demonstrate one-sided outer joins using both the LEFT and RIGHT keywords. The result rows with blank values for certain columns are non-matched rows.

Example 9.1 (Access): One-Sided Outer Join using LEFT JOIN

For offerings beginning with IS in the associated course number, retrieve the offer number, the course number, the faculty number, and the faculty name. Include an offering in the result even if the faculty is not yet assigned.

---

¹ The full SQL keywords are LEFT OUTER JOIN and RIGHT OUTER JOIN. The SQL:2011 standard and most DBMSs allow omission of the OUTER keyword.
The Oracle counterpart of this example uses % instead of * as the wildcard character.

SELECT OfferNo, CourseNo, Offering.FacNo, Faculty.FacNo,  
    FacFirstName, FacLastName  
FROM Offering LEFT JOIN Faculty  
ON Offering.FacNo = Faculty.FacNo  
WHERE CourseNo LIKE 'IS*'

Example 9.2 (Access): One-Sided Outer Join using RIGHT JOIN

For offerings beginning with “IS” in the associated course number, retrieve the offer number, the course number, the faculty number, and the faculty name. Include an offering in the result even if the faculty is not yet assigned. The result is identical to Example 9.1. The Oracle counterpart of this example uses % instead of * as the wildcard character.

SELECT OfferNo, CourseNo, Offering.FacNo, Faculty.FacNo,  
    FacFirstName, FacLastName  
FROM Faculty RIGHT JOIN Offering  
ON Offering.FacNo = Faculty.FacNo  
WHERE CourseNo LIKE 'IS*'

A full outer join generates a table with the matching rows plus the nonmatching rows from both input tables. Typically, a full outer join is used to combine two similar but not union compatible tables. For example, the Student and Faculty tables are similar because they contain information about university people. However, they are not union compatible. They have common columns such as first name, last name, and city but also unique columns such as GPA and salary. Occasionally, you will need to write a query that combines both tables. For example, a full outer join should be used to find all details about university people within a certain city.

**Full Outer Join**: an operator that generates the join result (the matching rows) plus the nonmatching rows from both input tables. SQL supports the full outer join operator through the FULL JOIN keyword.

SQL:2011 provides the FULL JOIN keyword as demonstrated in Example 9.3. Note the null values in both halves (Student and Faculty) of the result.

Example 9.3 (SQL:2011 and Oracle 9i and beyond): Full Outer Join

Combine the Faculty and Student tables using a full outer join. List the person number (faculty or student number), the name (first and last), the salary (faculty only), and the GPA (students only) in the result. This SQL statement does not execute in Microsoft Access.
SELECT FacNo, FacFirstName, FacLastName, FacSalary, StdNo, StdFirstName, StdLastName, StdGPA
FROM Faculty FULL JOIN Student
ON Student.StdNo = Faculty.FacNo

<table>
<thead>
<tr>
<th>FacNo</th>
<th>FacFirstName</th>
<th>FacLastName</th>
<th>FacSalary</th>
<th>StdNo</th>
<th>StdFirstName</th>
<th>StdLastName</th>
<th>StdGPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>123456789</td>
<td>HOMER</td>
<td>WELLS</td>
<td>3.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>124567890</td>
<td>BOB</td>
<td>NORBERT</td>
<td>2.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>234567890</td>
<td>CANDY</td>
<td>KENDALL</td>
<td>3.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>345678901</td>
<td>WALLY</td>
<td>KENDALL</td>
<td>2.8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>456789012</td>
<td>JOE</td>
<td>ESTRADA</td>
<td>3.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>567890123</td>
<td>MARIAH</td>
<td>DODGE</td>
<td>3.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>678901234</td>
<td>TESS</td>
<td>DODGE</td>
<td>3.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>789012345</td>
<td>ROBERTO</td>
<td>MORALES</td>
<td>2.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>890123456</td>
<td>LUKE</td>
<td>BRAZZI</td>
<td>2.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>901234567</td>
<td>WILLIAM</td>
<td>PILGRIM</td>
<td>3.8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>098765432</td>
<td>LEONARD</td>
<td>VINCE</td>
<td>35000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>543210987</td>
<td>VICTORIA</td>
<td>EMMANUEL</td>
<td>120000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>654321098</td>
<td>LEONARD</td>
<td>FIBON</td>
<td>70000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>765432109</td>
<td>NICKI</td>
<td>MACON</td>
<td>65000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>876543210</td>
<td>CRISTOPHER</td>
<td>COLAN</td>
<td>40000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>987654321</td>
<td>JULIA</td>
<td>MILLS</td>
<td>75000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Some DBMSs (such as Microsoft Access) do not directly support the full outer join operator. In these systems, a full outer join is formulated by taking the union of two one-sided outer joins using the steps shown below. The SELECT statement implementing these steps is shown in Example 9.4.

1. Construct a right join of Faculty and Student (non-matched rows of Student).
2. Construct a left join of Faculty and Student (non-matched rows of Faculty).
3. Construct a union of these two temporary tables. Remember when using the UNION operator, the two table arguments must be “union compatible”: each corresponding column from both tables must have compatible data types. Otherwise, the UNION operator will not work as expected.

Example 9.4 (Access): Full Outer Join Using a Union of Two One-Sided Outer Joins

Combine the Faculty and Student tables using a full outer join. List the person number (faculty or student number), the name (first and last), the salary (faculty only), and the GPA (students only) in the result. The result is identical to Example 9.3. This statement executes in Oracle although the FULL JOIN syntax as demonstrated in Example 9.3 is preferred for Oracle.

SELECT FacNo, FacFirstName, FacLastName, FacSalary, StdNo, StdFirstName, StdLastName, StdGPA
FROM Faculty RIGHT JOIN Student
ON Student.StdNo = Faculty.FacNo
UNION
SELECT FacNo, FacFirstName, FacLastName, FacSalary, StdNo, StdFirstName, StdLastName, StdGPA
FROM Faculty LEFT JOIN Student
ON Student.StdNo = Faculty.FacNo
### 9.1.2 Mixing Inner and Outer Joins

Inner and outer joins can be mixed as demonstrated in Examples 9.5 and 9.6. For readability, it is generally preferred to use the join operator style rather than to mix the join operator and cross product styles.

#### Example 9.5 (Access): Mixing a One-Sided Outer Join and an Inner Join

Combine columns from the `Faculty`, `Offering`, and `Course` tables for information systems courses (IS in the beginning of the course number) offered in 2013. Include a row in the result even if there is not an assigned instructor. The Oracle counterpart of this example uses % instead of * as the wild card character.

```sql
SELECT OfferNo, Offering.CourseNo, OffTerm, CrsDesc, Faculty.FacNo, FacFirstName, FacLastName
FROM ( Faculty RIGHT JOIN Offering
ON Offering.FacNo = Faculty.FacNo )
INNER JOIN Course
ON Course.CourseNo = Offering.CourseNo
WHERE Course.CourseNo LIKE 'IS*' AND OffYear = 2013
```

<table>
<thead>
<tr>
<th>OfferNo</th>
<th>CourseNo</th>
<th>OffTerm</th>
<th>CrsDesc</th>
<th>FacNo</th>
<th>FacFirstName</th>
<th>FacLastName</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111</td>
<td>IS320</td>
<td>SUMMER</td>
<td>FUNDAMENTALS OF BUSINESS PROGRAMMING</td>
<td>098-76-5432</td>
<td>LEONARD</td>
<td>VINCE</td>
</tr>
<tr>
<td>3333</td>
<td>IS320</td>
<td>SPRING</td>
<td>FUNDAMENTALS OF BUSINESS PROGRAMMING</td>
<td>543-21-0987</td>
<td>VICTORIA</td>
<td>EMMANUEL</td>
</tr>
<tr>
<td>4444</td>
<td>IS320</td>
<td>WINTER</td>
<td>FUNDAMENTALS OF BUSINESS PROGRAMMING</td>
<td>987-65-4321</td>
<td>JULIA</td>
<td>MILLS</td>
</tr>
<tr>
<td>5678</td>
<td>IS480</td>
<td>WINTER</td>
<td>FUNDAMENTALS OF DATABASE MANAGEMENT</td>
<td>876-54-3210</td>
<td>CRISTOPHER</td>
<td>COLAN</td>
</tr>
<tr>
<td>5679</td>
<td>IS480</td>
<td>SPRING</td>
<td>FUNDAMENTALS OF DATABASE MANAGEMENT</td>
<td>654-32-1098</td>
<td>LEONARD</td>
<td>FIBON</td>
</tr>
<tr>
<td>9876</td>
<td>IS460</td>
<td>SPRING</td>
<td>SYSTEMS ANALYSIS</td>
<td>654-32-1098</td>
<td>LEONARD</td>
<td>FIBON</td>
</tr>
</tbody>
</table>

#### Example 9.6 (Access): Mixing a One-Sided Outer Join and Two Inner Joins

List the rows of the `Offering` table where there is at least one student enrolled, in addition to the requirements of Example 9.5. Remove duplicate rows when there is more than one student enrolled in an offering. The Oracle counterpart of this example uses % instead of * as the wild card character.

```sql
SELECT DISTINCT Offering.OfferNo, Offering.CourseNo, OffTerm, CrsDesc, Faculty.FacNo, FacFirstName, FacLastName
FROM ( ( Faculty RIGHT JOIN Offering
ON Offering.FacNo = Faculty.FacNo )
INNER JOIN Course
ON Course.CourseNo = Offering.CourseNo )
INNER JOIN Enrollment
ON Offering.OfferNo = Enrollment.OfferNo
WHERE Offering.OfferNo LIKE 'IS*' AND OffYear = 2013
```

<table>
<thead>
<tr>
<th>OfferNo</th>
<th>CourseNo</th>
<th>OffTerm</th>
<th>CrsDesc</th>
<th>FacNo</th>
<th>FacFirstName</th>
<th>FacLastName</th>
</tr>
</thead>
<tbody>
<tr>
<td>5678</td>
<td>IS480</td>
<td>WINTER</td>
<td>FUNDAMENTALS OF DATABASE MANAGEMENT</td>
<td>987-65-4321</td>
<td>JULIA</td>
<td>MILLS</td>
</tr>
<tr>
<td>5679</td>
<td>IS480</td>
<td>SPRING</td>
<td>FUNDAMENTALS OF DATABASE MANAGEMENT</td>
<td>876-54-3210</td>
<td>CRISTOPHER</td>
<td>COLAN</td>
</tr>
<tr>
<td>9876</td>
<td>IS460</td>
<td>SPRING</td>
<td>SYSTEMS ANALYSIS</td>
<td>654-32-1098</td>
<td>LEONARD</td>
<td>FIBON</td>
</tr>
</tbody>
</table>

In most queries, you can change the order of inner and outer joins without any problem. The inner join operator is associative meaning that the order of inner join operations does not matter. The same result always occurs when changing the order of inner join operations. Although the one-sided outer join is not associative, the order of operations does not matter in most queries. For example, Example 9.6a returns the same results as Example 9.6.
Example 9.6a (Access): Mixing a One-Sided Outer Join and Two Inner Joins with the Outer Join Performed Last

List the rows of the Offering table where there is at least one student enrolled, in addition to the requirements of Example 9.5. Remove duplicate rows when there is more than one student enrolled in an offering. The Oracle counterpart of this example uses % instead of * as the wildcard character. The result is identical to Example 9.6.

```
SELECT DISTINCT Offering.OfferNo, Offering.CourseNo, OffTerm, CrsDesc, Faculty.FacNo, FacFirstName, FacLastName
FROM Faculty RIGHT JOIN
  ( ( Offering INNER JOIN Course
      ON Course.CourseNo = Offering.CourseNo )
   INNER JOIN Enrollment
     ON Offering.OfferNo = Enrollment.OfferNo )
   ON Offering.FacNo = Faculty.FacNo
WHERE Offering.CourseNo LIKE 'IS*' AND OffYear = 2013
```

Queries with Ambiguous Combinations of Joins and Outer Joins

Some queries combining inner and outer joins are ambiguous. In ambiguous queries, different orders of operations may produce different results. A query is ambiguous if a non-preserved table (table with only matching rows in the result) in a one-sided outer join is involved in another join or outer join operation. In Examples 9.6 and 9.6a, the Offering table is preserved (both matching and non-matching rows) so no ambiguity exists. The Offering table can participate in other join and outer join operations without causing ambiguity.

Ambiguity occurs if the direction of the one-sided outer join is reversed to preserve the Faculty table rather than the Offering table. In Example 9.6b, the result is different than Example 9.6c if the Faculty table has unmatched rows. Example 9.6b eliminates the unmatched rows because the outer join is performed before the inner join. Example 9.6c preserves the unmatched Faculty rows because the outer join is performed after the inner join. Microsoft Access will not execute either query. Oracle executes both examples but returns different results if the Faculty table contains unmatched rows.

Example 9.6b: Ambiguous Query Mixing a One-Sided Outer Join and Two Inner Joins with the Outer Join Performed First (Oracle)

This query eliminates the non-matching Faculty rows because the outer join is performed first. This query does not execute nor even display in Microsoft Access because of the ambiguity rule.

```
SELECT Offering.OfferNo, Offering.CourseNo, OffTerm, CrsDesc, Faculty.FacNo, FacFirstName, FacLastName
FROM (Faculty LEFT JOIN Offering
      ON Offering.FacNo = Faculty.FacNo)
INNER JOIN Course ON Course.CourseNo = Offering.CourseNo
```

Example 9.6c: Ambiguous Query Mixing a One-Sided Outer Join and Two Inner Joins with the Outer Join Performed Last (Oracle)

This query preserves the non-matching Faculty rows because the outer join is performed last. This query does not execute in Microsoft Access because of the ambiguity rule.
SELECT Offering.OfferNo, Offering.CourseNo, OffTerm, CrsDesc, Faculty.FacNo, FacFirstName, FacLastName
FROM (Offering INNER JOIN Course
    ON Course.CourseNo = Offering.CourseNo)
    RIGHT JOIN Faculty ON Offering.FacNo = Faculty.FacNo

Ambiguous Query: A query is ambiguous if a non-preserved table (table with only matching rows in the result) in a one-sided outer join is involved in another join or outer join operation. The result of an ambiguous query may depend on the order of joins and one-sided join operations in the FROM clause.

Despite the possibility for ambiguous queries, they are not common because the non-preserved table is typically not involved in other operations. The parent table is typically the non-preserved table. The child table is usually preserved and involved in other operations. An ambiguous query may indicate an error in query formulation rather than a valid formulation to address a legitimate business request.

9.2 Understanding Nested Queries

A nested query or subquery is a query (SELECT statement) inside a query. A nested query typically appears as part of a condition in the WHERE or HAVING clauses. Nested queries also can be used in the FROM clause. Nested queries can be used like a procedure (Type I nested query) in which the nested query is executed one time or like a loop (Type II nested query) in which the nested query is executed repeatedly. This section demonstrates examples of both kinds of nested queries and explains problems in which they can be applied.

9.2.1 Type I Nested Queries

Type I nested queries are like procedures in a programming language. A Type I nested query evaluates one time and produces a table. The nested (or inner) query does not reference the outer query. Using the IN comparison operator, a Type I nested query can be used to express a join. In Example 9.7, the nested query on the Enrollment table generates a list of qualifying student number values. A row is selected in the outer query on Student if the student number is an element of the nested query result.

Type I Nested Query: a nested query in which the inner query does not reference any tables used in the outer query. Type I nested queries can be used for some join problems and some difference problems.

Example 9.7: Using a Type I Nested Query to Perform a Join

List the student number, name, and major of students who have a high grade (≥ 3.5) in a course offering.

<table>
<thead>
<tr>
<th>StdNo</th>
<th>StdFirstName</th>
<th>StdLastName</th>
<th>StdMajor</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-45-6789</td>
<td>HOMER</td>
<td>WELLS</td>
<td>IS</td>
</tr>
<tr>
<td>124-56-7890</td>
<td>BOB</td>
<td>NORBERT</td>
<td>FIN</td>
</tr>
<tr>
<td>234-56-7890</td>
<td>CANDY</td>
<td>KENDALL</td>
<td>ACCT</td>
</tr>
<tr>
<td>567-89-0123</td>
<td>MARIAH</td>
<td>DODGE</td>
<td>IS</td>
</tr>
<tr>
<td>789-01-2345</td>
<td>ROBERTO</td>
<td>MORALES</td>
<td>FIN</td>
</tr>
<tr>
<td>890-12-3456</td>
<td>LUKE</td>
<td>BRAZZI</td>
<td>IS</td>
</tr>
<tr>
<td>901-23-4567</td>
<td>WILLIAM</td>
<td>PILGRIM</td>
<td>IS</td>
</tr>
</tbody>
</table>
Type I nested queries should be used only when the result does not contain any columns from the tables in the nested query. In Example 9.7, no columns from the Enrollment table are used in the result. In Example 9.8, the join between the Student and Enrollment tables cannot be performed with a Type I nested query because EnrGrade appears in the result.

**Example 9.8: Combining a Type I Nested Query and the Join Operator Style**

Retrieve the name, city, and grade of students who have a high grade (≥ 3.5) in a course offered in fall 2012.

```sql
SELECT StdFirstName, StdLastName, StdCity, EnrGrade
FROM Student INNER JOIN Enrollment
ON Student.StdNo = Enrollment.StdNo
WHERE EnrGrade >= 3.5 AND Enrollment.OfferNo IN
  ( SELECT OfferNo FROM Offering
    WHERE OffTerm = 'FALL' AND OffYear = 2012 )
```

<table>
<thead>
<tr>
<th>StdFirstName</th>
<th>StdLastName</th>
<th>StdCity</th>
<th>EnrGrade</th>
</tr>
</thead>
<tbody>
<tr>
<td>CANDY</td>
<td>KENDALL</td>
<td>TACOMA</td>
<td>3.5</td>
</tr>
<tr>
<td>MARIAH</td>
<td>DODGE</td>
<td>SEATTLE</td>
<td>3.8</td>
</tr>
<tr>
<td>HOMER</td>
<td>WELLS</td>
<td>SEATTLE</td>
<td>3.5</td>
</tr>
<tr>
<td>ROBERTO</td>
<td>MORALES</td>
<td>SEATTLE</td>
<td>3.5</td>
</tr>
</tbody>
</table>

It is possible to have multiple levels of nested queries although this practice is not encouraged because the statements can be difficult to read. In a nested query, you can have another nested query using the IN comparison operator in the WHERE clause. In Example 9.9, the nested query on the Offering table has a nested query on the Faculty table. No Faculty columns are needed in the main query or in the nested query on Offering.

**Example 9.9: Using a Type I Nested Query inside Another Type I Nested Query**

Retrieve the name, city, and grade of students who have a high grade (≥ 3.5) in a course offered in fall 2012 taught by Leonard Vince.

```sql
SELECT StdFirstName, StdLastName, StdCity, EnrGrade
FROM Student, Enrollment
WHERE Student.StdNo = Enrollment.StdNo
AND EnrGrade >= 3.5 AND Enrollment.OfferNo IN
  ( SELECT OfferNo FROM Offering
    WHERE OffTerm = 'FALL' AND OffYear = 2012
    AND FacNo IN
      ( SELECT FacNo FROM Faculty
        WHERE FacFirstName = 'LEONARD'
        AND FacLastName = 'VINCE' )
  )
```

<table>
<thead>
<tr>
<th>StdFirstName</th>
<th>StdLastName</th>
<th>StdCity</th>
<th>EnrGrade</th>
</tr>
</thead>
<tbody>
<tr>
<td>CANDY</td>
<td>KENDALL</td>
<td>TACOMA</td>
<td>3.5</td>
</tr>
<tr>
<td>MARIAH</td>
<td>DODGE</td>
<td>SEATTLE</td>
<td>3.8</td>
</tr>
<tr>
<td>HOMER</td>
<td>WELLS</td>
<td>SEATTLE</td>
<td>3.5</td>
</tr>
<tr>
<td>ROBERTO</td>
<td>MORALES</td>
<td>SEATTLE</td>
<td>3.5</td>
</tr>
</tbody>
</table>

The Type I style gives a visual feel to a query. You can visualize a Type I subquery as navigating between tables. Visit the table in the subquery to collect join values that can be used to select rows from the table in the outer query. The use of Type I nested queries is largely a matter of preference. Even if you do not prefer this join style, you should be prepared to interpret queries written by others with Type I nested queries.

DELETE and UPDATE statements provide another use of a Type I nested query. A Type I nested query is useful in a DELETE statement with conditions referencing related tables, as demonstrated in Example 9.10a. Using a Type I nested query is the standard way to reference related tables in DELETE statements. Similarly, a Type I nested query
is useful in an UPDATE statement with conditions referencing related tables as shown in Example 9.11a. Chapter 4 demonstrated the join operator style inside DELETE and UPDATE statements, a proprietary extension of Microsoft Access. For your reference, Examples 9.10b and 9.11b show equivalent DELETE and UPDATE statements using the join operator style.

**Example 9.10a: DELETE Statement Using a Type I Nested Query**

Delete offerings taught by Leonard Vince. Three Offering rows are deleted. In addition, this statement deletes related rows in the Enrollment table because the ON DELETE clause is set to CASCADE.

```
DELETE FROM Offering
WHERE Offering.FacNo IN
  ( SELECT FacNo FROM Faculty
    WHERE FacFirstName = 'LEONARD'
    AND FacLastName = 'VINCE' )
```

**Example 9.10b (Access only): DELETE Statement Using an INNER JOIN Operation**

Delete offerings taught by Leonard Vince. Three Offering rows are deleted. In addition, this statement deletes related rows in the Enrollment table because the ON DELETE clause is set to CASCADE.

```
DELETE Offering.*
FROM Offering INNER JOIN Faculty
ON Offering.FacNo = Faculty.FacNo
WHERE FacFirstName = 'LEONARD'
AND FacLastName = 'VINCE'
```

**Example 9.11a: UPDATE Statement Using a Type I Nested Query**

Update the location of offerings taught by Leonard Fibon in 2013 to BLM412. Two Offering rows are updated.

```
UPDATE Offering SET OffLocation = 'BLM412'
WHERE OffYear = 2013 AND FacNo IN
  ( SELECT FacNo FROM Faculty
    WHERE FacFirstName = 'LEONARD'
    AND FacLastName = 'FIBON')
```

**Example 9.11b (Access): UPDATE Statement Using an INNER JOIN Operation**

Update the location of offerings taught by Leonard Fibon in 2013 to BLM412. Two Offering rows are updated.

```
UPDATE Offering INNER JOIN Faculty
ON Offering.FacNo = Faculty.FacNo
SET OffLocation = 'BLM412'
WHERE OffYear = 2013 AND FacFirstName = 'LEONARD'
AND FacLastName = 'FIBON'
```

### 9.2.2 Limited SQL Formulations for Difference Problems

You should recall from Chapter 3 that the difference operator combines tables by finding the rows of a first table not in a second table. A typical usage of the difference operator is to combine two tables with some similar columns but not entirely union compatible. For example, you may want to find faculty who are not students. Although the Faculty and Student tables contain some compatible columns, the tables are not union compatible. The placement of the word *not* in the problem statement indicates that the result contains rows only in the Faculty table, not in the Student table. This requirement involves a difference operation.
Difference Problems: problem statements involving the difference operator often have a *not* relating two nouns in a sentence. For example, “students who are not faculty” and “employees who are not customers” are problem statements involving a difference operation.

Some difference problems can be formulated using a Type I nested query with the NOT IN operator. As long as the comparison among tables involves a single column, a Type I nested query can be used. In Example 9.12, a Type I nested query can be used because the comparison only involves a single column from the Faculty table (FacNo).

**Example 9.12: Using a Type I Nested Query for a Difference Problem**

Retrieve the faculty number, name (first and last), department, and salary of faculty who are *not* students.

```sql
SELECT FacNo, FacFirstName, FacLastName, FacDept, FacSalary
FROM Faculty
WHERE FacNo NOT IN ( SELECT StdNo FROM Student )
```

<table>
<thead>
<tr>
<th>FacNo</th>
<th>FacFirstName</th>
<th>FacLastName</th>
<th>FacDept</th>
<th>FacSalary</th>
</tr>
</thead>
<tbody>
<tr>
<td>098-76-5432</td>
<td>LEONARD</td>
<td>VINCE</td>
<td>MS</td>
<td>$35,000.00</td>
</tr>
<tr>
<td>543-21-0987</td>
<td>VICTORIA</td>
<td>EMMANUEL</td>
<td>MS</td>
<td>$120,000.00</td>
</tr>
<tr>
<td>654-32-1098</td>
<td>LEONARD</td>
<td>FIBON</td>
<td>MS</td>
<td>$70,000.00</td>
</tr>
<tr>
<td>765-43-2109</td>
<td>NICKI</td>
<td>MACON</td>
<td>FIN</td>
<td>$65,000.00</td>
</tr>
<tr>
<td>987-65-4321</td>
<td>JULIA</td>
<td>MILLS</td>
<td>FIN</td>
<td>$75,000.00</td>
</tr>
</tbody>
</table>

Another formulation approach for some difference problems involves a one-sided outer join operation to generate a table with only non-matching rows. The IS NULL comparison operator can remove rows that match, as demonstrated in Example 9.13. However, this formulation cannot be used with conditions to test on the non-preserved table (Student in Example 9.13) other than the IS NULL condition on the primary key column. If there are conditions to test on the Student table (such as on student class), another SQL formulation approach must be used.

**Example 9.13: One-Sided Outer Join with Only Non-matching Rows**

Retrieve the student number, name, department, and salary of faculty who are *not* students. The result is identical to Example 9.12.

```sql
SELECT FacNo, FacFirstName, FacLastName, FacSalary
FROM Faculty LEFT JOIN Student
ON Faculty.FacNo = Student.StdNo
WHERE Student.StdNo IS NULL
```

Although SQL:2011 does have a difference operator (the EXCEPT keyword), it is sometimes not convenient because only the common columns can be shown in the result. Example 9.14 does not provide the same result as Example 9.12 because the columns unique to the Faculty table (FacDept and FacSalary) are not in the result. Another query that uses the first result must be formulated to retrieve the unique Faculty columns.

**Example 9.14 (Oracle): Difference Query**

Show faculty who are *not* students (pure faculty). Only show the common columns in the result. Note that Microsoft Access does not support the EXCEPT keyword. Oracle uses the MINUS keyword instead of EXCEPT. The result is identical to Example 9.12 except for FacCity and FacState instead of FacDept and FacSalary.
SELECT FacNo AS PerNo, FacFirstName AS FirstName,
FacLastName AS LastName, FacCity AS City,
FacState AS State
FROM Faculty
MINUS
SELECT StdNo AS PerNo, StdFirstName AS FirstName,
StdLastName AS LastName, StdCity AS City,
StdState AS State
FROM Student

Difference Problems Cannot Be Solved with Inequality Joins

It is important to note that difference problems such as Example 9.12 cannot be solved with a join alone. Example 9.12 requires that every row of the Student table be searched to select a faculty row. In contrast, a join selects a faculty row when the first matching student row is found. To contrast difference and join problems, you should examine Example 9.15. Although it looks correct, it does not provide the desired result. Every faculty row will be in the result because there is at least one student row that does not match every faculty row.

Example 9.15: Inequality Join

Erroneous formulation for the problem “Retrieve the faculty number, name (first and last), and rank of faculty who are not students.” The result contains all faculty rows.

```
SELECT DISTINCT FacNo, FacFirstName, FacLastName,
FacRank
FROM Faculty, Student
WHERE Student.StdNo <> Faculty.FacNo
```

To understand Example 9.15, you can use the conceptual evaluation process discussed in Chapter 4 (Section 4.3). The result tables show the cross product (Table 9-3) of Tables 9-1 and 9-2 followed by the rows that satisfy the WHERE condition (Table 9-4). Notice that only one row of the cross product is deleted. The final result (Table 9-5) contains all rows of Table 9-2.

<table>
<thead>
<tr>
<th>StdNo</th>
<th>StdFirstName</th>
<th>StdLastName</th>
<th>StdMajor</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-45-6789</td>
<td>HOMER</td>
<td>WELLS</td>
<td>IS</td>
</tr>
<tr>
<td>124-56-7890</td>
<td>BOB</td>
<td>NORBERT</td>
<td>FIN</td>
</tr>
<tr>
<td>876-54-3210</td>
<td>CHRISTOPHE</td>
<td>COLAN</td>
<td>IS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FacNo</th>
<th>FacFirstName</th>
<th>FacLastName</th>
<th>FacRank</th>
</tr>
</thead>
<tbody>
<tr>
<td>098-76-5432</td>
<td>LEONARD</td>
<td>VINCE</td>
<td>ASST</td>
</tr>
<tr>
<td>543-21-0987</td>
<td>VICTORIA</td>
<td>EMMANUEL</td>
<td>PROF</td>
</tr>
<tr>
<td>876-54-3210</td>
<td>CRISTOPHE</td>
<td>COLAN</td>
<td>ASST</td>
</tr>
</tbody>
</table>
Table 9-3: Cross Product of the Sample Student and Faculty Tables

<table>
<thead>
<tr>
<th>FacNo</th>
<th>FacFirstName</th>
<th>FacLastName</th>
<th>FacRank</th>
<th>StdNo</th>
<th>StdFirstName</th>
<th>StdLastName</th>
<th>StdMajor</th>
</tr>
</thead>
<tbody>
<tr>
<td>098-76-5432</td>
<td>LEONARD</td>
<td>VINCE</td>
<td>ASST</td>
<td>123-45-6789</td>
<td>HOMER</td>
<td>WELLS</td>
<td>IS</td>
</tr>
<tr>
<td>098-76-5432</td>
<td>LEONARD</td>
<td>VINCE</td>
<td>ASST</td>
<td>124-56-7890</td>
<td>BOB</td>
<td>NORBERT</td>
<td>FIN</td>
</tr>
<tr>
<td>098-76-5432</td>
<td>LEONARD</td>
<td>VINCE</td>
<td>ASST</td>
<td>876-54-3210</td>
<td>CRISTOPHER</td>
<td>COLAN</td>
<td>IS</td>
</tr>
<tr>
<td>543-21-0987</td>
<td>VICTORIA</td>
<td>EMMANUEL</td>
<td>PROF</td>
<td>123-45-6789</td>
<td>HOMER</td>
<td>WELLS</td>
<td>IS</td>
</tr>
<tr>
<td>543-21-0987</td>
<td>VICTORIA</td>
<td>EMMANUEL</td>
<td>PROF</td>
<td>124-56-7890</td>
<td>BOB</td>
<td>NORBERT</td>
<td>FIN</td>
</tr>
<tr>
<td>543-21-0987</td>
<td>VICTORIA</td>
<td>EMMANUEL</td>
<td>PROF</td>
<td>876-54-3210</td>
<td>CRISTOPHER</td>
<td>COLAN</td>
<td>IS</td>
</tr>
<tr>
<td>876-54-3210</td>
<td>CRISTOPHER</td>
<td>COLAN</td>
<td>ASST</td>
<td>123-45-6789</td>
<td>HOMER</td>
<td>WELLS</td>
<td>IS</td>
</tr>
<tr>
<td>876-54-3210</td>
<td>CRISTOPHER</td>
<td>COLAN</td>
<td>ASST</td>
<td>124-56-7890</td>
<td>BOB</td>
<td>NORBERT</td>
<td>FIN</td>
</tr>
<tr>
<td>876-54-3210</td>
<td>CRISTOPHER</td>
<td>COLAN</td>
<td>ASST</td>
<td>876-54-3210</td>
<td>CRISTOPHER</td>
<td>COLAN</td>
<td>IS</td>
</tr>
</tbody>
</table>

Table 9-4: Restriction of Table 9-3 to Eliminate Matching Rows

<table>
<thead>
<tr>
<th>FacNo</th>
<th>FacFirstName</th>
<th>FacLastName</th>
<th>FacRank</th>
<th>StdNo</th>
<th>StdFirstName</th>
<th>StdLastName</th>
<th>StdMajor</th>
</tr>
</thead>
<tbody>
<tr>
<td>098-76-5432</td>
<td>LEONARD</td>
<td>VINCE</td>
<td>ASST</td>
<td>123-45-6789</td>
<td>HOMER</td>
<td>WELLS</td>
<td>IS</td>
</tr>
<tr>
<td>098-76-5432</td>
<td>LEONARD</td>
<td>VINCE</td>
<td>ASST</td>
<td>124-56-7890</td>
<td>BOB</td>
<td>NORBERT</td>
<td>FIN</td>
</tr>
<tr>
<td>098-76-5432</td>
<td>LEONARD</td>
<td>VINCE</td>
<td>ASST</td>
<td>876-54-3210</td>
<td>CRISTOPHER</td>
<td>COLAN</td>
<td>IS</td>
</tr>
<tr>
<td>543-21-0987</td>
<td>VICTORIA</td>
<td>EMMANUEL</td>
<td>PROF</td>
<td>123-45-6789</td>
<td>HOMER</td>
<td>WELLS</td>
<td>IS</td>
</tr>
<tr>
<td>543-21-0987</td>
<td>VICTORIA</td>
<td>EMMANUEL</td>
<td>PROF</td>
<td>124-56-7890</td>
<td>BOB</td>
<td>NORBERT</td>
<td>FIN</td>
</tr>
<tr>
<td>543-21-0987</td>
<td>VICTORIA</td>
<td>EMMANUEL</td>
<td>PROF</td>
<td>876-54-3210</td>
<td>CRISTOPHER</td>
<td>COLAN</td>
<td>IS</td>
</tr>
<tr>
<td>876-54-3210</td>
<td>CRISTOPHER</td>
<td>COLAN</td>
<td>ASST</td>
<td>123-45-6789</td>
<td>HOMER</td>
<td>WELLS</td>
<td>IS</td>
</tr>
<tr>
<td>876-54-3210</td>
<td>CRISTOPHER</td>
<td>COLAN</td>
<td>ASST</td>
<td>124-56-7890</td>
<td>BOB</td>
<td>NORBERT</td>
<td>FIN</td>
</tr>
<tr>
<td>876-54-3210</td>
<td>CRISTOPHER</td>
<td>COLAN</td>
<td>ASST</td>
<td>876-54-3210</td>
<td>CRISTOPHER</td>
<td>COLAN</td>
<td>IS</td>
</tr>
</tbody>
</table>

Table 9-5: Projection of Table 9-4 to Eliminate Student Columns

<table>
<thead>
<tr>
<th>FacNo</th>
<th>FacFirstName</th>
<th>FacLastName</th>
<th>FacRank</th>
</tr>
</thead>
<tbody>
<tr>
<td>098-76-5432</td>
<td>LEONARD</td>
<td>VINCE</td>
<td>ASST</td>
</tr>
<tr>
<td>543-21-0987</td>
<td>VICTORIA</td>
<td>EMMANUEL</td>
<td>PROF</td>
</tr>
<tr>
<td>876-54-3210</td>
<td>CRISTOPHER</td>
<td>COLAN</td>
<td>ASST</td>
</tr>
</tbody>
</table>

Summary of Limited Formulations for Difference Problems

This section has discussed three SQL formulations for difference problems. Each formulation has limitations as noted in Table 9-6. In practice, the one-sided outer join approach is the most restrictive as many problems involve conditions on the excluded table. Section 9.2.3 presents a more general formulation without the restrictions noted in Table 9-6.

Table 9-6: Limitations of SQL Formulations for Difference Problems

<table>
<thead>
<tr>
<th>SQL Formulation</th>
<th>Limitations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type I nested query with the NOT IN operator</td>
<td>Only one column for comparing rows of the two tables</td>
</tr>
<tr>
<td>One-sided outer join with an IS NULL condition</td>
<td>No conditions (except the IS NULL condition) on the non-preserved table</td>
</tr>
<tr>
<td>Difference operation using the EXCEPT or MINUS keywords</td>
<td>Result must contain only union-compatible columns</td>
</tr>
</tbody>
</table>
9.2.3 Using Type II Nested Queries for Difference Problems

Although Type II nested queries provide a more general solution for difference problems, they are conceptually more complex than Type I nested queries. Type II nested queries have two distinguishing features. First, Type II nested queries reference one or more columns from an outer query. Type II nested queries are sometimes known as related subqueries because they reference columns used in outer queries. In contrast, Type I nested queries are not correlated with outer queries. In Example 9.16, the nested query contains a reference to the Faculty table used in the outer query.

Example 9.16: Using a Type II Nested Query for a Difference Problem

Retrieve the faculty number, the name (first and last), the department, and the salary of faculty who are not students.

```
SELECT FacNo, FacFirstName, FacLastName, FacDept, FacSalary
FROM Faculty
WHERE NOT EXISTS
  ( SELECT * FROM Student
    WHERE Student.StdNo = Faculty.FacNo )
```

<table>
<thead>
<tr>
<th>FacNo</th>
<th>FacFirstName</th>
<th>FacLastName</th>
<th>FacDept</th>
<th>FacSalary</th>
</tr>
</thead>
<tbody>
<tr>
<td>098-76-5432</td>
<td>LEONARD</td>
<td>VINCE</td>
<td>MS</td>
<td>$35,000.00</td>
</tr>
<tr>
<td>543-21-0987</td>
<td>VICTORIA</td>
<td>EMMANUEL</td>
<td>MS</td>
<td>$120,000.00</td>
</tr>
<tr>
<td>654-32-1098</td>
<td>LEONARD</td>
<td>FIBON</td>
<td>MS</td>
<td>$70,000.00</td>
</tr>
<tr>
<td>765-43-2109</td>
<td>NICKI</td>
<td>MACON</td>
<td>FIN</td>
<td>$65,000.00</td>
</tr>
<tr>
<td>987-65-4321</td>
<td>JULIA</td>
<td>MILLS</td>
<td>FIN</td>
<td>$75,000.00</td>
</tr>
</tbody>
</table>

The second distinguishing feature of Type II nested queries involves execution. A Type II nested query executes one time for each row in the outer query. In this sense, a Type II nested query is similar to a nested loop that executes one time for each execution of the outer loop. In each execution of the inner loop, variables used in the outer loop are used in the inner loop. In other words, the inner query uses one or more values from the outer query in each execution.

**Type II Nested Query**: a nested query in which the inner query references a table used in the outer query. Because a Type II nested query executes for each row of its outer query, Type II nested queries are more difficult to understand and execute than Type I nested queries.

To help you understand Example 9.16, Table 9-9 traces the execution of the nested query using Tables 9-7 and 9-8. The EXISTS operator is true if the nested query returns one or more rows. In contrast, the NOT EXISTS operator is true if the nested query returns 0 rows. Thus, a faculty row in the outer query is selected only if there are no matching student rows in the nested query. For example, the first two rows in Table 9-7 are selected because there are no matching rows in Table 9-8. The third row is not selected because the nested query returns one row (the third row of Table 9-7).

Table 9-7: Sample Faculty Table

<table>
<thead>
<tr>
<th>FacNo</th>
<th>FacFirstName</th>
<th>FacLastName</th>
<th>FacRank</th>
</tr>
</thead>
<tbody>
<tr>
<td>098-76-5432</td>
<td>LEONARD</td>
<td>VINCE</td>
<td>ASST</td>
</tr>
<tr>
<td>543-21-0987</td>
<td>VICTORIA</td>
<td>EMMANUEL</td>
<td>PROF</td>
</tr>
<tr>
<td>876-54-3210</td>
<td>CRISTOPHER</td>
<td>COLAN</td>
<td>ASST</td>
</tr>
</tbody>
</table>

Table 9-8: Sample Student Table

<table>
<thead>
<tr>
<th>StdNo</th>
<th>StdFirstName</th>
<th>StdLastName</th>
<th>StdMajor</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-45-6789</td>
<td>HOMER</td>
<td>WELLS</td>
<td>IS</td>
</tr>
<tr>
<td>124-56-7890</td>
<td>BOB</td>
<td>NORBERT</td>
<td>FIN</td>
</tr>
<tr>
<td>876-54-3210</td>
<td>CRISTOPHER</td>
<td>COLAN</td>
<td>IS</td>
</tr>
</tbody>
</table>
NOT EXISTS operator: a table comparison operator often used with Type II nested queries. NOT EXISTS is true for a row in an outer query if the inner query returns no rows and false if the inner query returns one or more rows.

Example 9.17 shows another formulation that clarifies the meaning of the NOT EXISTS operator. Here, a faculty row is selected if the number of rows in the nested query is 0. Using the sample tables (Tables 9-7 and 9-8), the nested query result is 0 for the first two faculty rows.

**Example 9.17: Using a Type II Nested Query with the COUNT Function**

Retrieve the faculty number, the name, the department, and the salary of faculty who are not students. The result is the same as Example 9.16.

```sql
SELECT FacNo, FacFirstName, FacLastName, FacDept, FacSalary
FROM Faculty
WHERE 0 =
    ( SELECT COUNT(*) FROM Student
    WHERE Student.StdNo = Faculty.FacNo )
```

More Difficult Difference Problems

More difficult difference problems combine a difference operation with join operations. For example, consider the query to list students who took all of their information systems (IS) offerings in winter 2013 from the same instructor. The query results should include students who took only one offering as well as students who took more than one offering.

- Construct a list of students who have taken IS courses in winter 2013 (a join operation).
- Construct another list of students who have taken IS courses in winter 2013 from more than one instructor (a join operation).
- Use a difference operation (first student list minus the second student list) to produce the result.

Conceptualizing a problem in this manner forces you to recognize that it involves a difference operation. If you recognize the difference operation, you can make a formulation in SQL involving a nested query (Type II with NOT EXISTS or Type I with NOT IN) or the EXCEPT keyword. Example 9.18 shows a NOT EXISTS solution in which the outer query retrieves a student row if the student does not have an offering from a different instructor in the inner query.

**Example 9.18 (Access): More Difficult Difference Problem Using a Type II Nested Query**

List the student number and the name of students who took all of their information systems offerings in winter 2013 from the same instructor. Include students who took one or more offerings. Note that in the nested query, the columns Enrollment.StdNo and Offering.FacNo refer to the outer query.

```sql
SELECT DISTINCT Enrollment.StdNo, StdFirstName, StdLastName
FROM Student, Enrollment, Offering
WHERE Student.StdNo = Enrollment.StdNo
    AND Enrollment.OfferNo = Offering.OfferNo
    AND CourseNo LIKE 'IS*'
    AND OffTerm = 'WINTER'
    AND OffYear = 2013
    AND NOT EXISTS
        ( SELECT * FROM Enrollment E1, Offering O1
        WHERE E1.StdNo = StdNo
            AND O1.OfferNo = OfferNo
            AND CourseNo LIKE 'IS*'
            AND OffTerm = 'WINTER'
            AND OffYear = 2013
    )
```
WHERE E1.OfferNo = O1.OfferNo
AND Enrollment.StdNo = E1.StdNo
AND O1.CourseNo LIKE 'IS*'
AND O1.OffYear = 2013
AND O1.OffTerm = 'WINTER'
AND Offering.FacNo <> O1.FacNo )

<table>
<thead>
<tr>
<th>StdNo</th>
<th>StdFirstName</th>
<th>StdLastName</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-45-6789</td>
<td>HOMER</td>
<td>WELLS</td>
</tr>
<tr>
<td>234-56-7890</td>
<td>CANDY</td>
<td>KENDALL</td>
</tr>
<tr>
<td>345-67-8901</td>
<td>WALLY</td>
<td>KENDALL</td>
</tr>
<tr>
<td>456-78-9012</td>
<td>JOE</td>
<td>ESTRADA</td>
</tr>
<tr>
<td>567-89-0123</td>
<td>MARIAH</td>
<td>DODGE</td>
</tr>
</tbody>
</table>

Example 9.18 (Oracle): More Difficult Difference Problem Using a Type II Nested Query

List the student number and name of the students who took all of their information systems offerings in winter 2013 from the same instructor. Include students who took one or more offerings.

```
SELECT DISTINCT Enrollment.StdNo, StdFirstName, StdLastName
FROM Student, Enrollment, Offering
WHERE Student.StdNo = Enrollment.StdNo
AND Enrollment.OfferNo = Offering.OfferNo
AND CourseNo LIKE 'IS%' AND OffTerm = 'WINTER'
AND OffYear = 2013 AND NOT EXISTS
  ( SELECT * FROM Enrollment E1, Offering O1
    WHERE E1.OfferNo = O1.OfferNo
    AND Enrollment.StdNo = E1.StdNo
    AND O1.CourseNo LIKE 'IS%'
    AND O1.OffYear = 2013
    AND O1.OffTerm = 'WINTER'
    AND Offering.FacNo <> O1.FacNo )
```

Example 9.19 shows a second example using the NOT EXISTS operator to solve a complex difference problem. Conceptually this problem involves a difference operation between two sets: the set of all faculty members and the set of faculty members teaching in the specified term. The difference operation can be implemented by selecting a faculty in the outer query list if the faculty does not teach an offering during the specified term in the inner query result.

Example 9.19: Another Difference Problem Using a Type II Nested Query

List the name (first and last) and department of faculty who are not teaching in winter term 2013.

```
SELECT DISTINCT FacFirstName, FacLastName, FacDept
FROM Faculty
WHERE NOT EXISTS
  ( SELECT * FROM Offering
    WHERE Offering.FacNo = Faculty.FacNo
    AND OffTerm = 'WINTER' AND OffYear = 2013 )
```

<table>
<thead>
<tr>
<th>FacFirstName</th>
<th>FacLastName</th>
<th>FacDept</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRISTOPHER</td>
<td>COLAN</td>
<td>MS</td>
</tr>
<tr>
<td>LEONARD</td>
<td>FIBON</td>
<td>MS</td>
</tr>
<tr>
<td>LEONARD</td>
<td>VINCE</td>
<td>MS</td>
</tr>
</tbody>
</table>
Example 9.20 shows a third example using the NOT EXISTS operator to solve a complex difference problem. In this problem, the word *only* connecting different parts of the sentence indicates a difference operation. Conceptually this problem involves a difference operation between two sets: the set of all faculty members teaching in winter 2013 and the set of faculty members teaching in winter 2013 in addition to teaching in another term. The difference operation can be implemented by selecting a faculty teaching in winter 2013 in the outer query if the same faculty does not teach an offering in a different term in the nested query.

**Example 9.20: Another Difference Problem Using a Type II Nested Query**

List the name (first and last) and department of faculty who are *only* teaching in winter term 2013.

```sql
SELECT DISTINCT FacFirstName, FacLastName, FacDept
FROM Faculty F1, Offering O1
WHERE F1.FacNo = O1.FacNo
AND OffTerm = 'WINTER' AND OffYear = 2013
AND NOT EXISTS
  ( SELECT * FROM Offering O2
     WHERE O2.FacNo = F1.FacNo
     AND ( OffTerm <> 'WINTER' OR OffYear <> 2013 ) )
```

<table>
<thead>
<tr>
<th>FacFirstName</th>
<th>FacLastName</th>
<th>FacDept</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMMANUEL</td>
<td>VICTORIA</td>
<td>MS</td>
</tr>
<tr>
<td>MILLS</td>
<td>JULIA</td>
<td>FIN</td>
</tr>
</tbody>
</table>

### 9.2.4 Nested Queries in the FROM Clause

So far, you have seen nested queries in the WHERE clause with certain comparison operators (IN and EXISTS) as well as with traditional comparison operators when the nested query produces a single value such as the count of the number of rows. Similar to the usage in the WHERE clause, nested queries also can appear in the HAVING clause as demonstrated in the next section. Nested queries in the WHERE and the HAVING clauses have been part of SQL since its initial design.

In contrast, nested queries in the FROM clause were supported beginning with SQL:1999. The design of SQL:1999 began a philosophy of consistency in language design. Consistency means that wherever an object is permitted, an object expression should be permitted. In the FROM clause, this philosophy means that wherever a table is permitted, a table expression (a nested query) should be allowed. Nested queries in the FROM clause are not as widely used as nested queries in the WHERE and HAVING clauses. The remainder of this section demonstrates some specialized uses of nested queries in the FROM clause.

One usage of nested queries in the FROM clause is to compute an aggregate function within an aggregate function (nested aggregates). SQL does not permit an aggregate function inside another aggregate function. A nested query in the FROM clause overcomes the prohibition against nested aggregates as demonstrated in Example 9.21. Without a nested query in the FROM clause, two queries would be necessary to produce the output. In Access, the nested query would be a stored query. In Oracle, the nested query would be a view (see Chapter 10 for an explanation of views).

**Example 9.21: Using a Nested Query in the FROM Clause**

List the course number, the course description, the number of offerings, and the average enrollment across offerings.
SELECT T.CourseNo, T.CrsDesc, COUNT(*) AS NumOfferings, 
Avg(T.EnrollCount) AS AvgEnroll 
FROM 
( SELECT Course.CourseNo, CrsDesc, 
Offering.OfferNo, COUNT(*) AS EnrollCount 
FROM Offering, Enrollment, Course 
WHERE Offering.OfferNo = Enrollment.OfferNo 
AND Course.CourseNo = Offering.CourseNo 
GROUP BY Course.CourseNo, CrsDesc, Offering.OfferNo 
) T 
GROUP BY T.CourseNo, T.CrsDesc

<table>
<thead>
<tr>
<th>CourseNo</th>
<th>CrsDesc</th>
<th>NumOfferings</th>
<th>AvgEnroll</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIN300</td>
<td>FUNDAMENTALS OF FINANCE</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>FIN450</td>
<td>PRINCIPLES OF INVESTMENTS</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>FIN480</td>
<td>CORPORATE FINANCE</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>IS320</td>
<td>FUNDAMENTALS OF BUSINESS PROGRAMMING</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>IS460</td>
<td>SYSTEMS ANALYSIS</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>IS480</td>
<td>FUNDAMENTALS OF DATABASE MANAGEMENT</td>
<td>2</td>
<td>5.5</td>
</tr>
</tbody>
</table>

Another usage of a nested query in the FROM clause is to compute aggregates from multiple groupings. Without a nested query in the FROM clause, a query can contain aggregates from only one grouping. For example, multiple groupings are needed to summarize the number of students per offering and the number of resources per offering. This query would be useful if the design of the university database was extended with a Resource table and an associative table (ResourceUsage) connected to the Offering and the Resource tables via 1-M relationships. The query would require two nested queries in the FROM clause, one to retrieve the enrollment count for offerings and the other to retrieve the resource count for offerings.

In Access, a nested query in the FROM clause can compensate for the inability to use the DISTINCT keyword inside aggregate functions. For example, the DISTINCT keyword is necessary to compute the number of distinct courses taught by faculty as shown in Example 9.22. To produce the same results in Access, a nested query in the FROM clause is necessary as shown in Example 9.23. The nested query in the FROM clause uses the DISTINCT keyword to eliminate duplicate course numbers. Section 9.3.3 contains additional examples using nested queries in the FROM clause to compensate for the DISTINCT keyword inside the COUNT function.

Example 9.22 (Oracle): Using the DISTINCT Keyword inside the COUNT Function

List the faculty number, the last name, and the number of unique courses taught.

```
SELECT Faculty.FacNo, FacLastName,
       COUNT(DISTINCT CourseNo) AS NumPreparations
FROM Faculty, Offering
WHERE Faculty.FacNo = Offering.FacNo
GROUP BY Faculty.FacNo, FacLastName
```
Example 9.23: Using a Nested Query in the FROM Clause Instead of the DISTINCT Keyword inside the COUNT Function

List the faculty number, the last name, and the number of unique courses taught. The result is identical to Example 9.22. Although this SELECT statement executes in Access and Oracle, you should use the statement in Example 9.22 in Oracle because it will execute faster.

```
SELECT T.FacNo, T.FacLastName, 
       COUNT(*) AS NumPreparations 
FROM 
( SELECT DISTINCT Faculty.FacNo, FacLastName, CourseNo 
    FROM Offering, Faculty 
    WHERE Offering.FacNo = Faculty.FacNo ) T 
GROUP BY T.FacNo, T.FacLastName
```

9.3 Formulating Division Problems

Division problems can be some of the most difficult problems. Because of the difficulty, the divide operator of Chapter 3 is briefly reviewed. After this review, this section discusses some easier division problems before moving to more advanced problems.

9.3.1 Review of the Divide Operator

To review the divide operator, consider a simplified university database consisting of three tables: `Student1` (Table 9-10), `Club` (Table 9-11), and `StdClub` (Table 9-12) showing student membership in clubs. The divide operator is typically applied to linking tables showing M-N relationships. The `StdClub` table links the `Student1` and `Club` tables: a student may belong to many clubs and a club may have many students.

### Table 9-10: Student1 Table Listing

<table>
<thead>
<tr>
<th>StdNo</th>
<th>SName</th>
<th>SCity</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>JOE</td>
<td>SEATTLE</td>
</tr>
<tr>
<td>S2</td>
<td>SALLY</td>
<td>SEATTLE</td>
</tr>
<tr>
<td>S3</td>
<td>SUE</td>
<td>PORTLAND</td>
</tr>
</tbody>
</table>

### Table 9-11: Club Table Listing

<table>
<thead>
<tr>
<th>ClubNo</th>
<th>CName</th>
<th>CPurpose</th>
<th>CBudget</th>
<th>CActual</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>DELTA</td>
<td>SOCIAL</td>
<td>$1,000.00</td>
<td>$1,200.00</td>
</tr>
<tr>
<td>C2</td>
<td>BITS</td>
<td>ACADEMIC</td>
<td>$500.00</td>
<td>$350.00</td>
</tr>
<tr>
<td>C3</td>
<td>HELPS</td>
<td>SERVICE</td>
<td>$300.00</td>
<td>$330.00</td>
</tr>
<tr>
<td>C4</td>
<td>SIGMA</td>
<td>SOCIAL</td>
<td></td>
<td>$150.00</td>
</tr>
</tbody>
</table>

### Table 9-12: StdClub Table Listing

<table>
<thead>
<tr>
<th>StdNo</th>
<th>ClubNo</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>C1</td>
</tr>
<tr>
<td>S1</td>
<td>C2</td>
</tr>
<tr>
<td>S1</td>
<td>C3</td>
</tr>
<tr>
<td>S1</td>
<td>C4</td>
</tr>
<tr>
<td>S2</td>
<td>C1</td>
</tr>
<tr>
<td>S2</td>
<td>C4</td>
</tr>
<tr>
<td>S3</td>
<td>C3</td>
</tr>
</tbody>
</table>
The divide operator builds a table consisting of the values of one column (\textit{StdNo}) that match \textit{all} of the values in a specified column (\textit{ClubNo}) of a second table (\textit{Club}). A typical division problem is to list the students who belong to \textit{all} clubs. The resulting table contains only student S1 because S1 is associated with all four clubs.

\begin{center}
\textbf{Divide}: an operator of relational algebra that combines rows from two tables. The divide operator produces a table in which values of a column from one input table are associated with all the values from a column of the second table.
\end{center}

Division is more conceptually difficult than join because division matches on all values whereas join matches on a single value. If this problem involved a join, it would be stated as “list students who belong to \textit{any} club.” The key difference is the word \textit{any} versus \textit{all}. Most division problems can be written with adjectives \textit{every} or \textit{all} between a verb phrase representing a table and a noun representing another table. In this example, the phrase “students who belong to all clubs” fits this pattern. Another example is “students who have taken every course.”

\subsection{9.3.2 Simple Division Problems}

There are a number of ways to perform division in SQL. Some books describe an approach using Type II nested queries. Because this approach can be difficult to understand if you have not had a course in logic, a different approach is used here. The approach here uses the \texttt{COUNT} function with a nested query in the HAVING clause.

The basic idea is to compare the number of clubs associated with a student in the \textit{StdClub} table with the number of clubs in the \textit{Club} table. To perform this operation, group the \textit{StdClub} table on \textit{StdNo} and compare the number of rows in each \textit{StdNo} group with the number of rows in the \textit{Club} table. You can make this comparison using a nested query in the HAVING clause as shown in Example 9.24.

\example{9.24: Simplest Division Problem}

\begin{verbatim}
List the student number of students who belong to all of the clubs.

\begin{verbatim}
SELECT StdNo
FROM StdClub
GROUP BY StdNo
HAVING COUNT(*) = ( SELECT COUNT(*) FROM Club )
\end{verbatim}

\begin{verbatim}
\begin{tabular}{|c|}
\hline
\textbf{StdNo} \\
S1 \\
\hline
\end{tabular}
\end{verbatim}
\end{verbatim}

\endexample

Note that the \texttt{COUNT(*)} on the left-hand side tallies the number of rows in a \textit{StdNo} group. The right-hand side contains a nested query with only a \texttt{COUNT(*)} in the result. The nested query is Type I because there is no connection to the outer query. Therefore, the nested query only executes one time and returns a single row with a single value (the number of rows in the \textit{Club} table).

Now let us examine some variations of the first problem. The most typical variation is to retrieve students who belong to a subset of the clubs rather than all of the clubs. For example, retrieve students who belong to all social clubs. To accomplish this change, you should modify Example 9.24 by including a \texttt{WHERE} condition in both the outer and the nested query. Instead of counting all \texttt{Student1} rows in a \textit{StdNo} group, count only the rows where the club's purpose is social. Compare this count to the number of social clubs in the \textit{Club} table. Example 9.25 shows these modifications.

\example{9.25: Division Problem to Find a Subset Match}

\begin{verbatim}
List the student number of students who belong to all of the social clubs.

\begin{verbatim}
SELECT StdNo
FROM StdClub, Club
WHERE StdClub.ClubNo = Club.ClubNo
AND CPurpose = 'SOCIAL'
GROUP BY StdNo
HAVING COUNT(*) =
( SELECT COUNT(*) FROM Club
WHERE CPurpose = 'SOCIAL' )
\end{verbatim}
\end{verbatim}

\endexample
Other variations are shown in Examples 9.26 and 9.27. In Example 9.26, a join between StdClub and Student is necessary to obtain the student name. Example 9.27 reverses the previous problems by looking for clubs rather than students.

**Example 9.26: Division Problem with Joins**

List the student number and the name of students who belong to all social clubs.

```
SELECT Student1.StdNo, SName 
FROM StdClub, Club, Student1 
WHERE StdClub.ClubNo = Club.ClubNo 
  AND Student1.StdNo = StdClub.StdNo 
  AND CPurpose = 'SOCIAL' 
GROUP BY Student1.StdNo, SName 
HAVING COUNT(*) = 
  ( SELECT COUNT(*) FROM Club 
    WHERE CPurpose = 'SOCIAL' )
```

<table>
<thead>
<tr>
<th>StdNo</th>
<th>SName</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>JOE</td>
</tr>
<tr>
<td>S2</td>
<td>SALLY</td>
</tr>
</tbody>
</table>

**Example 9.27: Another Division Problem**

List the club numbers of clubs that have all Seattle students as members.

```
SELECT ClubNo 
FROM StdClub, Student1 
WHERE Student1.StdNo = StdClub.StdNo 
  AND SCity = 'SEATTLE' 
GROUP BY ClubNo 
HAVING COUNT(*) = 
  ( SELECT COUNT(*) FROM Student1 
    WHERE SCity = 'SEATTLE' )
```

<table>
<thead>
<tr>
<th>ClubNo</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
</tr>
<tr>
<td>C4</td>
</tr>
</tbody>
</table>

### 9.3.3 Advanced Division Problems

Example 9.28 (using the original university database tables) depicts another complication of division problems in SQL. Before tackling this additional complication, let us examine a simpler problem. Example 9.28 can be formulated with the same technique as shown in Section 9.3.2. First, join the Faculty and Offering tables, select rows matching the WHERE conditions, and group the result by faculty name (first and last). Then, compare the count of the rows in each faculty name group with the number of fall 2012, information systems offerings from the Offering table.

**Example 9.28 (Access): Division Problem with a Join**

List faculty number and the name (first and last) of faculty who teach all of the fall 2012, information systems offerings.
SELECT Faculty.FacNo, FacFirstName, FacLastName
FROM Faculty, Offering
WHERE Faculty.FacNo = Offering.FacNo
    AND OffTerm = 'FALL' AND CourseNo LIKE 'IS*'
    AND OffYear = 2012
GROUP BY Faculty.FacNo, FacFirstName, FacLastName
HAVING COUNT(*) = 
    ( SELECT COUNT(*) FROM Offering
            WHERE OffTerm = 'FALL' AND OffYear = 2012
            AND CourseNo LIKE 'IS*' )

<table>
<thead>
<tr>
<th>FacNo</th>
<th>FacFirstName</th>
<th>FacLastName</th>
</tr>
</thead>
<tbody>
<tr>
<td>098-76-5432</td>
<td>LEONARD</td>
<td>VINCE</td>
</tr>
</tbody>
</table>

Example 9.28 (Oracle): Division Problem with a Join

List faculty number and the name (first and last) of faculty who teach all of the fall 2012, information systems offerings.

SELECT Faculty.FacNo, FacFirstName, FacLastName
FROM Faculty, Offering
WHERE Faculty.FacNo = Offering.FacNo
    AND OffTerm = 'FALL' AND CourseNo LIKE 'IS%'
    AND OffYear = 2012
GROUP BY Faculty.FacNo, FacFirstName, FacLastName
HAVING COUNT(*) = 
    ( SELECT COUNT(*) FROM Offering
            WHERE OffTerm = 'FALL' AND OffYear = 2012
            AND CourseNo LIKE 'IS%' )

Example 9.28 is not particularly useful because it is unlikely that any instructor has taught every offering. Rather, it is more useful to retrieve instructors who have taught one offering of every course as demonstrated in Example 9.29. Rather than counting the rows in each group, count the unique CourseNo values. This change is necessary because CourseNo is not unique in the Offering table. There can be multiple rows with the same CourseNo, corresponding to multiple offerings for the same course. The solution only executes in Oracle because Access does not support the DISTINCT keyword in aggregate functions. Example 9.30 shows an Access solution using two nested queries in FROM clauses. The second nested query occurs inside the nested query in the HAVING clause. Appendix 9.A shows an alternative to nested queries in the FROM clause using multiple SELECT statements.

Example 9.29 (Oracle): Division Problem with DISTINCT inside COUNT

List the faculty number and the name (first and last) of faculty who teach at least one section of all of the fall 2012 information systems courses.

SELECT Faculty.FacNo, FacFirstName, FacLastName
FROM Faculty, Offering
WHERE Faculty.FacNo = Offering.FacNo
    AND OffTerm = 'FALL' AND CourseNo LIKE 'IS%'
    AND OffYear = 2012
GROUP BY Faculty.FacNo, FacFirstName, FacLastName
HAVING COUNT(DISTINCT CourseNo) = 
    ( SELECT COUNT(DISTINCT CourseNo) FROM Offering
            WHERE OffTerm = 'FALL' AND OffYear = 2012
            AND CourseNo LIKE 'IS%')
Example 9.30 (Access): Division Problem Using Nested Queries in the FROM Clauses instead of the DISTINCT Keyword inside the COUNT Function

List the faculty number and the name (first and last) of faculty who teach at least one section of all of the fall 2012 information systems courses. The result is the same as Example 9.29.

```
SELECT FacNo, FacFirstName, FacLastName
FROM
  (SELECT DISTINCT Faculty.FacNo, FacFirstName, FacLastName, CourseNo
   FROM Faculty, Offering
   WHERE Faculty.FacNo = Offering.FacNo
     AND OffTerm = 'FALL' AND OffYear = 2012
     AND CourseNo LIKE 'IS*'
  )
GROUP BY FacNo, FacFirstName, FacLastName
HAVING COUNT(*) =
  ( SELECT COUNT(*) FROM
    ( SELECT DISTINCT CourseNo
     FROM Offering
     WHERE OffTerm = 'FALL' AND OffYear = 2012
       AND CourseNo LIKE 'IS*'
    )
  )
```

Example 9.31 is another variation of the technique used in Example 9.29. The DISTINCT keyword is necessary so that students taking more than one offering from the same instructor are not counted twice. Note that the DISTINCT keyword is not necessary for the nested query because only rows of the Student table are counted. Example 9.32 shows an Access solution using a nested query in the FROM clause.

Example 9.31 (Oracle): Another Division Problem with DISTINCT inside COUNT

List the faculty who have taught all seniors in their fall 2012 information systems offerings.

```
SELECT Faculty.FacNo, FacFirstName, FacLastName
FROM Faculty, Offering, Enrollment, Student
WHERE Faculty.FacNo = Offering.FacNo
  AND OffTerm = 'FALL' AND CourseNo LIKE 'IS%
  AND OffYear = 2012 AND StdClass = 'SR'
  AND Offering.OfferNo = Enrollment.OfferNo
  AND Student.StdNo = Enrollment.StdNo
GROUP BY Faculty.FacNo, FacFirstName, FacLastName
HAVING COUNT(DISTINCT Student.StdNo) =
  ( SELECT COUNT(*) FROM Student
    WHERE StdClass = 'SR' );
```

Example 9.32 (Access): Another Division Problem Using Nested Queries in the FROM Clauses Instead of the DISTINCT Keyword inside the COUNT Function

List the faculty who have taught all seniors in their fall 2012 information systems offerings. The result is identical to Example 9.31. The Oracle version of this statement uses the % as the wild card character.

```
SELECT FacNo, FacFirstName, FacLastName
FROM
  ( SELECT DISTINCT Faculty.FacNo, FacFirstName, FacLastName, Student.StdNo
  )
```

Example 9.30 (Access): Division Problem Using Nested Queries in the FROM Clauses instead of the DISTINCT Keyword inside the COUNT Function

List the faculty number and the name (first and last) of faculty who teach at least one section of all of the fall 2012 information systems courses. The result is the same as Example 9.29.

```
SELECT FacNo, FacFirstName, FacLastName
FROM
  (SELECT DISTINCT Faculty.FacNo, FacFirstName, FacLastName, CourseNo
   FROM Faculty, Offering
   WHERE Faculty.FacNo = Offering.FacNo
     AND OffTerm = 'FALL' AND OffYear = 2012
     AND CourseNo LIKE 'IS*'
  )
GROUP BY FacNo, FacFirstName, FacLastName
HAVING COUNT(*) =
  ( SELECT COUNT(*) FROM
    ( SELECT DISTINCT CourseNo
     FROM Offering
     WHERE OffTerm = 'FALL' AND OffYear = 2012
       AND CourseNo LIKE 'IS*'
    )
  )
```

Example 9.31 is another variation of the technique used in Example 9.29. The DISTINCT keyword is necessary so that students taking more than one offering from the same instructor are not counted twice. Note that the DISTINCT keyword is not necessary for the nested query because only rows of the Student table are counted. Example 9.32 shows an Access solution using a nested query in the FROM clause.

Example 9.31 (Oracle): Another Division Problem with DISTINCT inside COUNT

List the faculty who have taught all seniors in their fall 2012 information systems offerings.

```
SELECT Faculty.FacNo, FacFirstName, FacLastName
FROM Faculty, Offering, Enrollment, Student
WHERE Faculty.FacNo = Offering.FacNo
  AND OffTerm = 'FALL' AND CourseNo LIKE 'IS%
  AND OffYear = 2012 AND StdClass = 'SR'
  AND Offering.OfferNo = Enrollment.OfferNo
  AND Student.StdNo = Enrollment.StdNo
GROUP BY Faculty.FacNo, FacFirstName, FacLastName
HAVING COUNT(DISTINCT Student.StdNo) =
  ( SELECT COUNT(*) FROM Student
    WHERE StdClass = 'SR' );
```

Example 9.32 (Access): Another Division Problem Using Nested Queries in the FROM Clauses Instead of the DISTINCT Keyword inside the COUNT Function

List the faculty who have taught all seniors in their fall 2012 information systems offerings. The result is identical to Example 9.31. The Oracle version of this statement uses the % as the wild card character.

```
SELECT FacNo, FacFirstName, FacLastName
FROM
  ( SELECT DISTINCT Faculty.FacNo, FacFirstName, FacLastName, Student.StdNo
  )
```
FROM Faculty, Offering, Enrollment, Student
WHERE Faculty.FacNo = Offering.FacNo
    AND OffTerm = 'FALL' AND CourseNo LIKE 'IS*'
    AND OffYear = 2012 AND StdClass = 'SR'
    AND Offering.OfferNo = Enrollment.OfferNo
    AND Student.StdNo = Enrollment.StdNo)
GROUP BY FacNo, FacFirstName, FacLastName
HAVING COUNT(*) =
    (SELECT COUNT(*) FROM Student
    WHERE StdClass = 'SR');

9.4 Null Value Considerations

This section does not involve difficult matching problems or new parts of the SELECT statement. Rather, this section explains interpretation of query results when tables contain null values. These effects have largely been ignored until this section to simplify the presentation. Because many databases use null values, you need to understand the effects to attain a deeper understanding of query formulation.

Null values affect simple conditions involving comparison operators, compound conditions involving logical operators, aggregate calculations, and grouping. As you will see, some of the null value effects are rather subtle. Because of these subtle effects, a good table design minimizes, although it usually does not eliminate, the use of null values. The null value effects described in this section are specified in the SQL standards (1992 through 2011). Because specific DBMSs may provide different results, you may need to experiment with your DBMS.

9.4.1 Effect on Simple Conditions

Simple conditions involve a comparison operator, a column or column expression, and a constant, column, or column expression. A simple condition results in a null value if either column (or column expression) in a comparison is null. A row qualifies in the result if the simple condition evaluates to true for the row. Rows evaluating to false or null are discarded. Example 9.33 depicts a simple condition evaluating to null for one of the rows.

**Example 9.33: Simple Condition Using a Column with Null Values**

List the clubs (Table 9-11) with a budget greater than $200. The club with a null budget (C4) is omitted because the condition evaluates as a null value.

```
SELECT *
FROM Club
WHERE CBudget > 200
```

<table>
<thead>
<tr>
<th>ClubNo</th>
<th>CName</th>
<th>CPurpose</th>
<th>CBudget</th>
<th>CActual</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>DELTA</td>
<td>SOCIAL</td>
<td>$1,000.00</td>
<td>$1,200.00</td>
</tr>
<tr>
<td>C2</td>
<td>BITS</td>
<td>ACADEMIC</td>
<td>$500.00</td>
<td>$350.00</td>
</tr>
<tr>
<td>C3</td>
<td>HELPS</td>
<td>SERVICE</td>
<td>$300.00</td>
<td>$330.00</td>
</tr>
</tbody>
</table>

A more subtle result can occur when a simple condition involves two columns and at least one column contains null values. If neither column contains null values, every row will be in the result of either the simple condition or the opposite (negation) of the simple condition. For example, if < is the operator of a simple condition, the opposite condition contains assuming the columns remain in the same positions. If at least one column contains null values, some rows will not appear in the result of either the simple condition or its negation. More precisely, rows containing null values will be excluded in both results as demonstrated in Examples 9.34 and 9.35.

**Example 9.34: Simple Condition Involving Two Columns**

List the clubs with the budget greater than the actual spending. The club with a null budget (C4) is omitted because the condition evaluates to null.
SELECT *
FROM Club
WHERE CBudget > CActual

<table>
<thead>
<tr>
<th>ClubNo</th>
<th>CName</th>
<th>CPurpose</th>
<th>CBudget</th>
<th>CActual</th>
</tr>
</thead>
<tbody>
<tr>
<td>C2</td>
<td>BITS</td>
<td>ACADEMIC</td>
<td>$500.00</td>
<td>$350.00</td>
</tr>
</tbody>
</table>

**Example 9.35: Opposite Condition of Example 9.34**

List the clubs with the budget less than or equal to the actual spending. The club with a null budget (C4) is omitted because the condition evaluates to null.

```
SELECT *
FROM Club
WHERE CBudget <= CActual
```

<table>
<thead>
<tr>
<th>ClubNo</th>
<th>CName</th>
<th>CPurpose</th>
<th>CBudget</th>
<th>CActual</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>DELTA</td>
<td>SOCIAL</td>
<td>$1,000.00</td>
<td>$1,200.00</td>
</tr>
<tr>
<td>C3</td>
<td>HELPS</td>
<td>SERVICE</td>
<td>$300.00</td>
<td>$330.00</td>
</tr>
</tbody>
</table>

**9.4.2 Effect on Compound Conditions**

Compound conditions involve one or more simple conditions connected by the logical or Boolean operators AND, OR, and NOT. Like simple conditions, compound conditions evaluate to true, false, or null. A row is selected if the entire compound condition in the WHERE clause evaluates to true.

To evaluate the result of a compound condition, the SQL:2011 standard uses truth tables with three values. A truth table shows the results of combinations of values (true, false, and null) with the Boolean operators. Truth tables with three values define a three-valued logic. Tables 9-13 through 9-15 depict truth tables for the AND, OR, and NOT operators. The internal cells in these tables are the result values. For example, the first internal cell (True) in Table 9-13 results from the AND operator applied to two conditions with true values. You can test your understanding of the truth tables using Examples 9.36 and 9.37.

**Table 9-13: AND Truth Table**

<table>
<thead>
<tr>
<th>AND</th>
<th>True</th>
<th>False</th>
<th>Null</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>False</td>
<td>Null</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>Null</td>
<td>Null</td>
<td>False</td>
<td>Null</td>
</tr>
</tbody>
</table>

**Table 9-14: OR Truth Table**

<table>
<thead>
<tr>
<th>OR</th>
<th>True</th>
<th>False</th>
<th>Null</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
<td>Null</td>
</tr>
<tr>
<td>Null</td>
<td>True</td>
<td>Null</td>
<td>Null</td>
</tr>
</tbody>
</table>

**Table 9-15: NOT Truth Table**

<table>
<thead>
<tr>
<th>NOT</th>
<th>True</th>
<th>False</th>
<th>Null</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
<td>Null</td>
</tr>
</tbody>
</table>

**Example 9.36: Evaluation of a Compound OR Condition with a Null Value**

List the clubs with the budget less than or equal to the actual spending or the actual spending less than $200. The
club with a null budget (C4) is included because the second condition evaluates to true.

```
SELECT *
FROM Club
WHERE CBudget <= CActual OR CActual < 200
```

<table>
<thead>
<tr>
<th>ClubNo</th>
<th>CName</th>
<th>CPurpose</th>
<th>CBudget</th>
<th>CActual</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>DELTA</td>
<td>SOCIAL</td>
<td>$1,000.00</td>
<td>$1,200.00</td>
</tr>
<tr>
<td>C3</td>
<td>HELPS</td>
<td>SERVICE</td>
<td>$300.00</td>
<td>$330.00</td>
</tr>
<tr>
<td>C4</td>
<td>SIGMA</td>
<td>SOCIAL</td>
<td>$150.00</td>
<td></td>
</tr>
</tbody>
</table>

**Example 9.37: Evaluation of a Compound AND Condition with a Null Value**

List the clubs (Table 9-11) with the budget less than or equal to the actual spending and the actual spending less than $500. The club with a null budget (C4) is not included because the first condition evaluates to null.

```
SELECT *
FROM Club
WHERE CBudget <= CActual AND CActual < 500
```

<table>
<thead>
<tr>
<th>ClubNo</th>
<th>CName</th>
<th>CPurpose</th>
<th>CBudget</th>
<th>CActual</th>
</tr>
</thead>
<tbody>
<tr>
<td>C3</td>
<td>HELPS</td>
<td>SERVICE</td>
<td>$300.00</td>
<td>$330.00</td>
</tr>
</tbody>
</table>

**9.4.3 Effect on Aggregate Calculations and Grouping**

Null values are ignored in aggregate calculations. Although this statement seems simple, the results can be subtle. For the COUNT function, COUNT(*) returns a different value than COUNT(column) if the column contains null values. COUNT(*) always returns the number of rows. COUNT(column) returns the number of non-null values in the column. Example 9.38 demonstrates the difference between COUNT(*) and COUNT(column).

**Example 9.38: COUNT Function with Null Values**

List the number of rows in the `Club` table and the number of values in the `CBudget` column.

```
SELECT COUNT(*) AS NumRows,
       COUNT(CBudget) AS NumBudgets
FROM Club
```

<table>
<thead>
<tr>
<th>NumRows</th>
<th>NumBudgets</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

An even more subtle effect can occur if the SUM or AVG functions are applied to a column with null values. Without regard to null values, the following equation is true: \( \text{SUM(Column1)} + \text{SUM(Column2)} = \text{SUM(Column1 + Column2)} \). With null values in at least one of the columns, the equation may not be true because a calculation involving a null value yields a null value. If Column1 has a null value in one row, the plus operation in \( \text{SUM(Column1 + Column2)} \) produces a null value for that row. However, the value of Column2 in the same row is counted in \( \text{SUM(Column2)} \). Example 9.39 demonstrates this subtle effect using the minus operator instead of the plus operator.

**Example 9.39: SUM Function with Null Values**

Using the `Club` table, list the sum of the budget values, the sum of the actual values, the difference of the two sums, and the sum of the differences (budget – actual). The last two columns differ because of a null value in the `CBudget` column. Parentheses enclose negative values in the result.

```
SELECT SUM(CBudget) AS SumBudget,
       SUM(CActual) AS SumActual,
       SUM(CBudget) - SUM(CActual) AS SumDifference,
       SUM(CBudget-CActual) AS SumOfDifferences
FROM Club
```
Null values also can affect grouping operations performed in the GROUP BY clause. The SQL standard stipulates that all rows with null values are grouped together. The grouping column shows null values in the result. In the university database, this kind of grouping operation is useful to find course offerings without assigned professors, as demonstrated in Example 9.40.

Example 9.40: Grouping on a Column with Null Values

For each faculty number in the Offering table, list the number of offerings. In Microsoft Access and Oracle, an Offering row with a null FacNo value displays as a blank. In Access, the null row displays before the non-null rows as shown below. In Oracle, the null row displays after the non-null rows.

```
SELECT FacNo, COUNT(*) AS NumRows
FROM Offering
GROUP BY FacNo
```

<table>
<thead>
<tr>
<th>FacNo</th>
<th>NumRows</th>
</tr>
</thead>
<tbody>
<tr>
<td>098-76-5432</td>
<td>2</td>
</tr>
<tr>
<td>543-21-0987</td>
<td>3</td>
</tr>
<tr>
<td>654-32-1098</td>
<td>1</td>
</tr>
<tr>
<td>765-43-2109</td>
<td>2</td>
</tr>
<tr>
<td>876-54-3210</td>
<td>1</td>
</tr>
<tr>
<td>987-65-4321</td>
<td>2</td>
</tr>
</tbody>
</table>

9.5 Hierarchical Queries

Hierarchical queries involve self-referencing relationships in which a child entity is related to at most one parent entity. Classical organization charts, part explosion diagrams, chart of accounts, and XML documents are the most prominent examples amendable to hierarchical queries. A typical hierarchical query may involve finding details or summarizing features about employees managed directly or indirectly by a specified manager. Self-referencing relationships are specialized but important parts of applications so understanding hierarchical query formulation provides advanced skills not shared by typical database professionals.

This section covers two approaches for hierarchical queries. Oracle developed a proprietary extension of the SELECT statement using the CONNECT BY PRIOR clause. Standard SQL (SQL:1999 onwards) provides an extension to the SELECT statement involving the WITH clause and recursive common table expressions. Since the Oracle notation is more succinct, the proprietary Oracle approach is covered first in more detail. The standard SQL notation is presented in less detail to provide some background for other DBMSs. Hierarchical queries are not supported in Microsoft Access so none of the examples in this section execute in any version of Microsoft Access. Before covering the hierarchical query approaches, an example of hierarchically structured data is presented.

Hierarchical Query: a query involving self-referencing relationships in which a child row is related to at most one parent row. Hierarchical queries typically retrieve details about child rows (both direct and indirect) or summarize column values of child rows.

9.5.1 Hierarchical Data Example

To study hierarchical query formulation, you need to clearly understand hierarchically structured data. Table 9-16 shows the new Faculty2 table expanded from the Faculty table with additional rows, some columns removed, and some rows slightly altered to fit into hierarchical query examples. The Faculty2 table has a self-referencing relationship with FacSupervisor as a foreign key referencing FacNo. Each row has at most one parent row. Rows having a
null value for FacSupervisor reside at the top of the organization chart. In Table 9-16, Victoria Emmanuel and Nicki Macon reside at the top of the organization chart.

To clarify the hierarchical structure, Figures 9.2 and 9.3 graphically depict organization charts along with important column values for easy comparison. A graphical representation of sample data can help you to formulate hierarchical queries.

Table 9-16: Sample Faculty2 Table

<table>
<thead>
<tr>
<th>FacNo</th>
<th>FacSupervisor</th>
<th>FacFirstName</th>
<th>FacLastName</th>
<th>FacHireDate</th>
<th>FacRank</th>
<th>FacSalary</th>
</tr>
</thead>
<tbody>
<tr>
<td>098-76-5432</td>
<td>654-32-1098</td>
<td>LEONARD</td>
<td>VINCE</td>
<td>10-Apr-2000</td>
<td>ASST</td>
<td>$55,000</td>
</tr>
<tr>
<td>543-21-0987</td>
<td>VICTORIA</td>
<td>EMMANUEL</td>
<td></td>
<td>15-Apr-2001</td>
<td>PROF</td>
<td>$120,000</td>
</tr>
<tr>
<td>654-32-1098</td>
<td>543-21-0987</td>
<td>LEONARD</td>
<td>FIBON</td>
<td>01-May-1999</td>
<td>ASSC</td>
<td>$70,000</td>
</tr>
<tr>
<td>765-43-2109</td>
<td></td>
<td>NICKI</td>
<td>MACON</td>
<td>11-Apr-2002</td>
<td>ASSC</td>
<td>$105,000</td>
</tr>
<tr>
<td>876-54-3210</td>
<td>654-32-1098</td>
<td>CRISTOPHER</td>
<td>COLAN</td>
<td>01-Mar-2004</td>
<td>ASST</td>
<td>$90,000</td>
</tr>
<tr>
<td>987-65-4321</td>
<td>765-43-2109</td>
<td>JULIA</td>
<td>MILLS</td>
<td>15-Mar-2005</td>
<td>ASSC</td>
<td>$95,000</td>
</tr>
<tr>
<td>111-22-3333</td>
<td>543-21-0987</td>
<td>JOHN</td>
<td>MILLSON</td>
<td>01-May-2005</td>
<td>PROF</td>
<td>$110,000</td>
</tr>
<tr>
<td>333-22-4444</td>
<td>111-22-3333</td>
<td>SALLY</td>
<td>SCOTT</td>
<td>01-May-2006</td>
<td>ASST</td>
<td>$90,000</td>
</tr>
<tr>
<td>555-66-7777</td>
<td>111-22-3333</td>
<td>SUSAN</td>
<td>JONES</td>
<td>01-May-2007</td>
<td>ASSC</td>
<td>$125,000</td>
</tr>
<tr>
<td>777-11-4321</td>
<td>765-43-2109</td>
<td>AIMEE</td>
<td>MANNING</td>
<td>15-Mar-2006</td>
<td>ASST</td>
<td>$85,000</td>
</tr>
<tr>
<td>888-33-1111</td>
<td>987-65-4321</td>
<td>JAMES</td>
<td>BLOKE</td>
<td>15-Apr-2008</td>
<td>ASST</td>
<td>$85,000</td>
</tr>
<tr>
<td>789-12-3210</td>
<td>987-65-4321</td>
<td>JAIME</td>
<td>SANCHEZ</td>
<td>10-May-2009</td>
<td>PROF</td>
<td>$107,000</td>
</tr>
</tbody>
</table>

Figure 9.2: Organization Chart with Victoria Emmanuel at the Top
The basic hierarchical query involves listing each top level faculty member along with all related subordinates, direct and indirect. The sequence of rows from root (top row) to leaf (lowest level) is known as a path. A SELECT statement cannot list all subordinates on a path unless the number of levels of subordinates is known. A self-join can be done for each level but a variable number of self-joins is necessary even if the maximum number of levels is known. Thus, an extension to the SELECT statement is necessary to formulate even the basic hierarchical query.

Compiler optimization and higher productivity are important advantages of query language support for hierarchical queries. SQL compilers have specialized algorithms and optimization methods for hierarchical queries. In contrast, coding a hierarchical query in a procedure with explicit loops eliminates the possibility of optimization by a SQL compiler. In addition, procedural coding reduces software productivity as more lines of code are necessary along with programming language knowledge.

**9.5.2 Proprietary Oracle Extensions for Hierarchical Queries**

Oracle provides the CONNECT BY PRIOR clause along with other clauses, operators, functions, and pseudo columns to support hierarchical queries. Syntactically, the CONNECT BY PRIOR clause and other clauses follow the FROM and WHERE clauses in a SELECT statement. The operators, functions, and pseudo columns can appear in expressions in the list of result columns and conditions. Pseudo columns are not actual columns in a table, but they behave like columns.

The examples begin with the simplest hierarchical query, although not particularly useful. The CONNECT BY PRIOR clause contains a condition relating parent and child rows, typically the self-join condition. Example 9.41 uses the CONNECT BY PRIOR clause to visit each row on a path. Each row is visited one time for each level on a path. For example, the row with COLAN appears three times in the result because it resides on level 3 as previously shown in Figure 9.2. The LEVEL pseudo column identifies the hierarchical level of a row starting with 1 for the root level. The Faculty2 table contains two root rows appearing with level 1 in the result.

**Example 9.41: Simple Hierarchical Query using CONNECT BY PRIOR (Oracle)**

The result contains 28 rows (Table 9-17) with 18 rows for the six leaf rows (faculty not managing other faculty) at level 3 (6 * 3), 8 rows for the 4 rows at level 2 (4 * 2), and 2 root rows. The sorting order makes it easier to verify the visitation of rows.
Example 9.41 treats all rows as roots of hierarchies. Typically, a small number of rows are designated as roots. Oracle provides the START WITH clause to identify root rows. In Example 9.42, the rows with null values for FacSupervisor are designated as the starting rows. The START WITH clause eliminates duplicate rows in the result caused by treating each row as a root.

**Example 9.42: Hierarchical Query using START WITH (Oracle)**

Example 9.42 revises Example 9.41 with a START WITH clause to limit starting rows to the roots of the faculty hierarchies. The results are conveniently sorted to indicate that each Faculty2 row appears once in the result.
SELECT FacNo, FacSupervisor, FacFirstName, FacLastName, 
FacHireDate, FacSalary, FacRank, LEVEL 
FROM Faculty2 
START WITH FacSupervisor IS NULL 
CONNECT BY PRIOR FacNo = FacSupervisor 
ORDER BY LEVEL;

In Example 9.42, the relationship of rows on the same level is not clear. To depict relationships among rows on the same level, Oracle provides the SIBLINGS keyword to specify a sort order for siblings, rows with the same parent. Example 9.43 sorts siblings by FacLastName, a more convenient order than FacNo. In the result, you can see the lexicographic order for siblings with COLAN followed by VINCE under parent row FIBON. The rows are indented in the first column to show the hierarchical relationships.

**Example 9.43: Hierarchical Query using the SIBLINGS Keyword (Oracle)**

The LPAD function adds spaces on the left to show the hierarchical structure. The LEVEL pseudo column determines the amount of padding with no padding for the root rows.

```
SELECT LPAD(' ',2*(LEVEL-1)) || FacLastName AS LastName, 
FacHireDate, FacSalary, FacRank, LEVEL 
FROM Faculty2 
START WITH FacSupervisor IS NULL 
CONNECT BY PRIOR FacNo = FacSupervisor 
ORDER SIBLINGS BY FacLastName;
```

<table>
<thead>
<tr>
<th>FacNo</th>
<th>FacSupervisor</th>
<th>FacFirstName</th>
<th>FacLastName</th>
<th>FacHireDate</th>
<th>FacRank</th>
<th>FacSalary</th>
<th>LEVEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>765-43-2109</td>
<td>NICKI</td>
<td>MACON</td>
<td>11-APR-02</td>
<td>ASSC</td>
<td>105000</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>543-21-0987</td>
<td>VICTORIA</td>
<td>EMMANUEL</td>
<td>15-APR-01</td>
<td>PROF</td>
<td>120000</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>654-32-1098</td>
<td>543-21-0987</td>
<td>LEONARD</td>
<td>FIBON</td>
<td>01-MAY-99</td>
<td>ASSC</td>
<td>70000</td>
<td>2</td>
</tr>
<tr>
<td>987-65-4321</td>
<td>JULIA</td>
<td>MILLS</td>
<td>15-MAR-05</td>
<td>ASSC</td>
<td>95000</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>777-11-4321</td>
<td>765-43-2109</td>
<td>AIMEE</td>
<td>MANNING</td>
<td>15-MAR-06</td>
<td>ASST</td>
<td>85000</td>
<td>2</td>
</tr>
<tr>
<td>111-22-3333</td>
<td>543-21-0987</td>
<td>JOHN</td>
<td>MILLSON</td>
<td>01-MAY-05</td>
<td>PROF</td>
<td>110000</td>
<td>2</td>
</tr>
<tr>
<td>789-12-3210</td>
<td>987-65-4321</td>
<td>JAIME</td>
<td>SANCHEZ</td>
<td>10-MAY-09</td>
<td>PROF</td>
<td>107000</td>
<td>3</td>
</tr>
<tr>
<td>876-54-3210</td>
<td>654-32-1098</td>
<td>CRISTOPHER</td>
<td>COLAN</td>
<td>01-MAR-04</td>
<td>ASST</td>
<td>90000</td>
<td>3</td>
</tr>
<tr>
<td>888-33-1111</td>
<td>987-65-4321</td>
<td>JAMES</td>
<td>BLOKE</td>
<td>15-MAR-09</td>
<td>ASST</td>
<td>85000</td>
<td>3</td>
</tr>
<tr>
<td>555-66-7777</td>
<td>111-22-3333</td>
<td>SUSAN</td>
<td>JONES</td>
<td>01-MAY-07</td>
<td>ASSC</td>
<td>125000</td>
<td>3</td>
</tr>
<tr>
<td>333-22-4444</td>
<td>111-22-3333</td>
<td>SALLY</td>
<td>SCOTT</td>
<td>01-MAY-06</td>
<td>ASST</td>
<td>90000</td>
<td>3</td>
</tr>
<tr>
<td>098-76-5432</td>
<td>654-32-1098</td>
<td>LEONARD</td>
<td>VINCE</td>
<td>10-APR-00</td>
<td>ASST</td>
<td>55000</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LastName</th>
<th>FacHireDate</th>
<th>FacRank</th>
<th>FacSalary</th>
<th>LEVEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMMANUEL</td>
<td>15-APR-01</td>
<td>PROF</td>
<td>120000</td>
<td>1</td>
</tr>
<tr>
<td>FIBON</td>
<td>01-MAY-99</td>
<td>ASSC</td>
<td>70000</td>
<td>2</td>
</tr>
<tr>
<td>COLAN</td>
<td>01-MAR-04</td>
<td>ASST</td>
<td>90000</td>
<td>3</td>
</tr>
<tr>
<td>VINCE</td>
<td>10-APR-00</td>
<td>ASST</td>
<td>55000</td>
<td>3</td>
</tr>
<tr>
<td>MILLSON</td>
<td>01-MAY-05</td>
<td>PROF</td>
<td>110000</td>
<td>2</td>
</tr>
<tr>
<td>JONES</td>
<td>01-MAY-07</td>
<td>ASSC</td>
<td>125000</td>
<td>3</td>
</tr>
<tr>
<td>SCOTT</td>
<td>01-MAY-06</td>
<td>ASST</td>
<td>90000</td>
<td>3</td>
</tr>
<tr>
<td>MACON</td>
<td>11-APR-02</td>
<td>ASSC</td>
<td>105000</td>
<td>1</td>
</tr>
<tr>
<td>MANNING</td>
<td>15-MAR-06</td>
<td>ASST</td>
<td>85000</td>
<td>2</td>
</tr>
<tr>
<td>MILLS</td>
<td>15-MAR-05</td>
<td>ASSC</td>
<td>95000</td>
<td>2</td>
</tr>
<tr>
<td>BLOKE</td>
<td>15-APR-08</td>
<td>ASST</td>
<td>85000</td>
<td>3</td>
</tr>
<tr>
<td>SANCHEZ</td>
<td>10-MAY-09</td>
<td>PROF</td>
<td>107000</td>
<td>3</td>
</tr>
</tbody>
</table>
Instead of indenting to show the hierarchical structure, the complete path can be shown. Oracle provides the SYS_CONNECT_BY_PATH function to show the complete path with a column name and a separator character as parameters.

Example 9.44: Hierarchical Query using the SYS_CONNECT_BY_PATH function (Oracle)

The SYS_CONNECT_BY_PATH function uses the FacLastName column to identify rows and the / as the separator between rows on a path.

```sql
SELECT SYS_CONNECT_BY_PATH(FacLastName,'/') AS Path,
       FacHireDate, FacSalary, FacRank, LEVEL
FROM Faculty2
START WITH FacSupervisor IS NULL
CONNECT BY PRIOR FacNo = FacSupervisor
ORDER SIBLINGS BY FacLastName;
```

<table>
<thead>
<tr>
<th>Path</th>
<th>FacHireDate</th>
<th>FacRank</th>
<th>FacSalary</th>
<th>LEVEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>/EMMANUEL</td>
<td>15-APR-01</td>
<td>PROF</td>
<td>120000</td>
<td>1</td>
</tr>
<tr>
<td>/EMMANUEL/FIBON</td>
<td>01-MAY-99</td>
<td>ASSC</td>
<td>70000</td>
<td>2</td>
</tr>
<tr>
<td>/EMMANUEL/FIBON/COLAN</td>
<td>01-MAR-04</td>
<td>ASST</td>
<td>90000</td>
<td>3</td>
</tr>
<tr>
<td>/EMMANUEL/FIBON/VINCE</td>
<td>10-APR-00</td>
<td>ASST</td>
<td>55000</td>
<td>3</td>
</tr>
<tr>
<td>/EMMANUEL/MILLSON</td>
<td>01-MAY-05</td>
<td>PROF</td>
<td>110000</td>
<td>2</td>
</tr>
<tr>
<td>/EMMANUEL/MILLSON/JONES</td>
<td>01-MAY-07</td>
<td>ASSC</td>
<td>125000</td>
<td>3</td>
</tr>
<tr>
<td>/EMMANUEL/MILLSON/SCOTT</td>
<td>01-MAY-06</td>
<td>ASST</td>
<td>90000</td>
<td>3</td>
</tr>
<tr>
<td>/MACON</td>
<td>11-APR-02</td>
<td>ASSC</td>
<td>105000</td>
<td>1</td>
</tr>
<tr>
<td>/MACON/MANNING</td>
<td>15-MAR-06</td>
<td>ASST</td>
<td>85000</td>
<td>2</td>
</tr>
<tr>
<td>/MACON/MILLS</td>
<td>15-MAR-05</td>
<td>ASST</td>
<td>95000</td>
<td>2</td>
</tr>
<tr>
<td>/MACON/MILLS/BLOKE</td>
<td>15-APR-08</td>
<td>ASST</td>
<td>85000</td>
<td>3</td>
</tr>
<tr>
<td>/MACON/MILLS/SANCHEZ</td>
<td>10-MAY-09</td>
<td>PROF</td>
<td>107000</td>
<td>3</td>
</tr>
</tbody>
</table>

In addition to the SYS_CONNECT_BY_PATH function, Oracle provides other syntax elements for hierarchical queries as shown in Example 9.45. The CONNECT_BY_ROOT operator retrieves a column value from a root row. The CONNECT_BY_LEAF pseudo column provides a row's leaf status. A row is a leaf if it has no children.

Example 9.45: Hierarchical Query using CONNECT_BY_ROOT and CONNECT_BY_ISLEAF (Oracle)

The CONNECT_BY_ROOT operator uses a column name (FacLastName). The CONNECT_BY_ISLEAF pseudo column returns 1 if the row is a leaf and 0 otherwise.

```sql
SELECT SYS_CONNECT_BY_PATH(FacLastName,'/') AS Path,
       CONNECT_BY_ROOT FacLastName AS Root,
       CONNECT_BY_ISLEAF AS IsLeaf,
       FacHireDate, FacSalary, FacRank, LEVEL
FROM Faculty2
START WITH FacSupervisor IS NULL
CONNECT BY PRIOR FacNo = FacSupervisor
ORDER SIBLINGS BY FacLastName;
```
The CONNECT_BY_ROOT operator can be used indirectly for grouping so that summary totals can be calculated for paths in a hierarchy. To calculate summary totals, the root should not be restricted by the START WITH clause. Without a root restriction, every row will be considered a root so that summary totals are calculated for each row, not just the root rows. Example 9.46 shows summary salary totals and subordinate counts for each faculty member and subordinates.

**Example 9.46: Summary Totals for a Hierarchical Query using the CONNECT_BY_ROOT function (Oracle)**

A nested query in the FROM clause is necessary because the CONNECT_BY_ROOT operator cannot be used as a grouping column. The result is sorted by the number of faculty members in the group including the root row and subordinates (COUNT(*)).

```sql
SELECT Root, COUNT(*)-1 AS NumSubordinates,
       SUM(FacSalary) AS FacSalarySum
FROM
   ( SELECT CONNECT_BY_ROOT FacLastName AS Root, FacSalary
       FROM Faculty2
       CONNECT BY PRIOR FacNo = FacSupervisor )
GROUP BY Root
ORDER BY COUNT(*) DESC;
```

<table>
<thead>
<tr>
<th>Path</th>
<th>Root</th>
<th>IsLeaf</th>
<th>FacHireDate</th>
<th>FacRank</th>
<th>FacSalary</th>
<th>LEVEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>/EMMANUEL</td>
<td>EMMANUEL</td>
<td>0</td>
<td>15-APR-01</td>
<td>PROF</td>
<td>120000</td>
<td>1</td>
</tr>
<tr>
<td>/EMMANUEL/FIBON</td>
<td>EMMANUEL</td>
<td>0</td>
<td>01-MAY-99</td>
<td>ASSC</td>
<td>70000</td>
<td>2</td>
</tr>
<tr>
<td>/EMMANUEL/FIBON/COLAN</td>
<td>EMMANUEL</td>
<td>1</td>
<td>01-MAR-04</td>
<td>ASST</td>
<td>90000</td>
<td>3</td>
</tr>
<tr>
<td>/EMMANUEL/FIBON/VINCE</td>
<td>EMMANUEL</td>
<td>1</td>
<td>10-APR-00</td>
<td>ASST</td>
<td>55000</td>
<td>3</td>
</tr>
<tr>
<td>/EMMANUEL/MILLSON</td>
<td>EMMANUEL</td>
<td>0</td>
<td>01-MAY-05</td>
<td>PROF</td>
<td>110000</td>
<td>2</td>
</tr>
<tr>
<td>/EMMANUEL/MILLSON/JONES</td>
<td>EMMANUEL</td>
<td>1</td>
<td>01-MAY-07</td>
<td>ASSC</td>
<td>125000</td>
<td>3</td>
</tr>
<tr>
<td>/EMMANUEL/MILLSON/SCOTT</td>
<td>EMMANUEL</td>
<td>1</td>
<td>01-MAY-06</td>
<td>ASST</td>
<td>90000</td>
<td>3</td>
</tr>
<tr>
<td>/MACON</td>
<td>MACON</td>
<td>0</td>
<td>11-APR-02</td>
<td>ASSC</td>
<td>105000</td>
<td>1</td>
</tr>
<tr>
<td>/MACON/MANNING</td>
<td>MACON</td>
<td>1</td>
<td>15-MAR-06</td>
<td>ASST</td>
<td>85000</td>
<td>2</td>
</tr>
<tr>
<td>/MACON/MILLS</td>
<td>MACON</td>
<td>0</td>
<td>15-MAR-05</td>
<td>ASSC</td>
<td>95000</td>
<td>2</td>
</tr>
<tr>
<td>/MACON/MILLS/BLOKE</td>
<td>MACON</td>
<td>1</td>
<td>15-APR-08</td>
<td>ASST</td>
<td>85000</td>
<td>3</td>
</tr>
<tr>
<td>/MACON/MILLS/SANCHEZ</td>
<td>MACON</td>
<td>1</td>
<td>10-MAY-09</td>
<td>PROF</td>
<td>107000</td>
<td>3</td>
</tr>
</tbody>
</table>

The last problems involve path exceptions. Many hierarchies show consistency of values known as monotonicity on paths. Monotonicity means that the column values of subordinates change in the same direction (usually smaller) from ancestors. For example in an organization hierarchy, monotonicity indicates that a manager has larger compensation...
than direct and indirect subordinates. Path exception queries involve violations about the expected monotonicity of values. Typical path exception queries involve managers making less than subordinates, assemblies weighing less than constituent components, and parent accounts with smaller balances than related subaccounts.

**Path Exception Query:** a hierarchical query listing violations of monotonicity in path relationships in hierarchical data. Monotonicity means that the column values of subordinates change in the same direction (usually smaller) from ancestors.

The first step to formulate path exception queries involves the closure of the hierarchy. The closure of a hierarchy shows all pairs in which a child can be reached from a parent. In an organization chart, the closure shows a manager paired with each direct and indirect subordinate. In Figure 9.2, the closure for Macon contains Macon paired with each subordinate, Mills, Manning, Bloke, and Sanchez. Example 9.47 demonstrates the SELECT statement to derive the closure for the Faculty2 table. Each row in the hierarchy is paired with each direct and indirect subordinate as shown in query result.

**Example 9.47: Closure of the Faculty2 Table (Oracle)**

The WHERE clause removes redundant rows in which the faculty number matches the root's faculty number. The nested query in the FROM clause is necessary because the CONNECT BY_ROOT operator cannot be used in the WHERE clause preceding the CONNECT WITH clause. The unary PRIOR operator retrieves the value of the column name argument from the parent row. Note that the PRIOR operator involves the immediate parent row while the CONNECT BY_ROOT operator involves the root row. The SYS_CONNECT_BY_PATH function and sort order improve the readability of the result.

```
SELECT * 
FROM
  ( SELECT FacNo, PRIOR FacNo AS PriorFacNo, 
    CONNECT_BY_ROOT FacNo AS FacSupNo, FacLastName, 
    FacSalary, FacRank, 
    SYS_CONNECT_BY_PATH(FacLastName,'/') AS Path 
  FROM Faculty2 
  CONNECT BY PRIOR FacNo = FacSupervisor ) 
WHERE FacSupNo <> FacNo 
ORDER BY FacLastName;
```

<table>
<thead>
<tr>
<th>FacNo</th>
<th>PriorFacNo</th>
<th>FacSupNo</th>
<th>FacLastName</th>
<th>FacRank</th>
<th>FacSalary</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>888-33-1111</td>
<td>987-65-4321</td>
<td>987-65-4321</td>
<td>BLOKE</td>
<td>ASST</td>
<td>85000</td>
<td>/MILLS/BLOKE</td>
</tr>
<tr>
<td>888-33-1111</td>
<td>987-65-4321</td>
<td>765-43-2109</td>
<td>BLOKE</td>
<td>ASST</td>
<td>85000</td>
<td>/MACON/MILLS/BLOKE</td>
</tr>
<tr>
<td>876-54-3210</td>
<td>654-32-1098</td>
<td>543-21-0987</td>
<td>COLAN</td>
<td>ASST</td>
<td>90000</td>
<td>/EMMANUEL/FIBON/COLAN</td>
</tr>
<tr>
<td>876-54-3210</td>
<td>654-32-1098</td>
<td>654-32-1098</td>
<td>COLAN</td>
<td>ASST</td>
<td>90000</td>
<td>/FIBON/COLAN</td>
</tr>
<tr>
<td>654-32-1098</td>
<td>543-21-0987</td>
<td>543-21-0987</td>
<td>FIBON</td>
<td>ASSC</td>
<td>70000</td>
<td>/EMMANUEL/FIBON</td>
</tr>
<tr>
<td>555-66-7777</td>
<td>111-22-3333</td>
<td>111-22-3333</td>
<td>JONES</td>
<td>ASSC</td>
<td>125000</td>
<td>/EMMANUEL/MILLSON/JONES</td>
</tr>
<tr>
<td>555-66-7777</td>
<td>111-22-3333</td>
<td>543-21-0987</td>
<td>JONES</td>
<td>ASSC</td>
<td>125000</td>
<td>/MILLSON/JONES</td>
</tr>
<tr>
<td>777-11-4321</td>
<td>765-43-2109</td>
<td>765-43-2109</td>
<td>MANNING</td>
<td>ASST</td>
<td>85000</td>
<td>/MACON/MANNING</td>
</tr>
<tr>
<td>987-65-4321</td>
<td>765-43-2109</td>
<td>765-43-2109</td>
<td>MILLS</td>
<td>ASSC</td>
<td>95000</td>
<td>/MACON/MILLS</td>
</tr>
<tr>
<td>111-22-3333</td>
<td>543-21-0987</td>
<td>543-21-0987</td>
<td>MILLSON</td>
<td>PROF</td>
<td>110000</td>
<td>/EMMANUEL/MILLSON</td>
</tr>
<tr>
<td>789-12-3210</td>
<td>987-65-4321</td>
<td>987-65-4321</td>
<td>SANCHEZ</td>
<td>PROF</td>
<td>107000</td>
<td>/MILLS/SANCHEZ</td>
</tr>
<tr>
<td>789-12-3210</td>
<td>987-65-4321</td>
<td>765-43-2109</td>
<td>SANCHEZ</td>
<td>PROF</td>
<td>107000</td>
<td>/MACON/MILLS/SANCHEZ</td>
</tr>
<tr>
<td>333-22-4444</td>
<td>111-22-3333</td>
<td>111-22-3333</td>
<td>SCOTT</td>
<td>ASST</td>
<td>90000</td>
<td>/MILLSON/SCOTT</td>
</tr>
<tr>
<td>333-22-4444</td>
<td>111-22-3333</td>
<td>543-21-0987</td>
<td>SCOTT</td>
<td>ASST</td>
<td>90000</td>
<td>/EMMANUEL/MILLSON/SCOTT</td>
</tr>
<tr>
<td>098-76-5432</td>
<td>654-32-1098</td>
<td>543-21-0987</td>
<td>VINCE</td>
<td>ASST</td>
<td>55000</td>
<td>/EMMANUEL/FIBON/VINCE</td>
</tr>
<tr>
<td>098-76-5432</td>
<td>654-32-1098</td>
<td>654-32-1098</td>
<td>VINCE</td>
<td>ASST</td>
<td>55000</td>
<td>/FIBON/VINCE</td>
</tr>
</tbody>
</table>
The last two examples use the closure query block to retrieve path exceptions. Example 9.48 retrieves subordinate faculty members with larger salaries than their ancestors (either direct or indirect supervisors). Example 9.49 retrieves subordinate faculty members with a higher rank than their supervisors. The rank order is PROF (full professor), ASSC (associate professor), and ASST (assistant professor). The results for both examples show the values for the subordinate and ancestor faculty member for ease of comparison.

**Example 9.48: Path Exception Query to Retrieve Faculty Earning more than a Supervisor, either Direct or Indirect (Oracle)**

The path exception condition (comparison of faculty salary values) is added to the WHERE condition of the outer query. The nested query uses the `CONNECT_BY_ROOT` operator to retrieve values from the root row.

```sql
SELECT * 
FROM  
  ( SELECT FacNo, CONNECT_BY_ROOT FacNo AS FacSupNo, FacLastName, CONNECT_BY_ROOT FacLastName AS FacSupLastName, FacSalary, CONNECT_BY_ROOT FacSalary AS FacSupSalary 
    FROM Faculty2 
    CONNECT BY PRIOR FacNo = FacSupervisor ) 
WHERE FacSupNo <> FacNo AND FacSalary > FacSupSalary;
```

<table>
<thead>
<tr>
<th>FacNo</th>
<th>FacSupNo</th>
<th>FacLastName</th>
<th>FacSupLastName</th>
<th>FacSalary</th>
<th>FacSupSalary</th>
</tr>
</thead>
<tbody>
<tr>
<td>876-54-3210</td>
<td>654-32-1098</td>
<td>COLAN</td>
<td>FIBON</td>
<td>90000</td>
<td>70000</td>
</tr>
<tr>
<td>555-66-7777</td>
<td>543-21-0987</td>
<td>JONES</td>
<td>MILLSON</td>
<td>125000</td>
<td>110000</td>
</tr>
<tr>
<td>789-12-3210</td>
<td>987-65-4321</td>
<td>SANCHEZ</td>
<td>MILLS</td>
<td>107000</td>
<td>95000</td>
</tr>
<tr>
<td>555-66-7777</td>
<td>111-22-3333</td>
<td>JONES</td>
<td>EMMANUEL</td>
<td>125000</td>
<td>120000</td>
</tr>
<tr>
<td>789-12-3210</td>
<td>785-43-2109</td>
<td>SANCHEZ</td>
<td>MACON</td>
<td>107000</td>
<td>105000</td>
</tr>
</tbody>
</table>

**Example 9.49: Path Exception Query to Retrieve Faculty with a Higher Rank than a Supervisor, either Direct or Indirect (Oracle)**

The path exception condition (comparison of faculty rank values) in the outer query involves three combinations of inconsistent values for a subordinate (FacRank) and an ancestor (FacSupRank). The order among ranks is PROF > ASSC > ASST.

```sql
SELECT * 
FROM  
  ( SELECT FacNo, CONNECT_BY_ROOT FacNo AS FacSupNo, FacLastName, CONNECT_BY_ROOT FacLastName AS FacSupLastName, FacRank, CONNECT_BY_ROOT FacRank AS FacSupRank 
    FROM Faculty2 
    CONNECT BY PRIOR FacNo = FacSupervisor ) 
WHERE FacSupNo <> FacNo AND ( FacRank = 'PROF' AND FacSupRank = 'ASSC' ) 
  OR ( FacRank = 'PROF' AND FacSupRank = 'ASST' ) 
  OR ( FacRank = 'ASSC' AND FacSupRank = 'ASST' ) ;
```

<table>
<thead>
<tr>
<th>FacNo</th>
<th>FacSupNo</th>
<th>FacLastName</th>
<th>FacSupLastName</th>
<th>FacRank</th>
<th>FacSupRank</th>
</tr>
</thead>
<tbody>
<tr>
<td>789-12-3210</td>
<td>987-65-4321</td>
<td>SANCHEZ</td>
<td>MILLS</td>
<td>PROF</td>
<td>ASSC</td>
</tr>
<tr>
<td>789-12-3210</td>
<td>765-43-2109</td>
<td>SANCHEZ</td>
<td>MACON</td>
<td>PROF</td>
<td>ASSC</td>
</tr>
</tbody>
</table>

The Oracle notation presented in this section has a number of syntax elements. To help you use the notation, Table 9-16 presents a convenient summary.
Table 9-16: Summary of Proprietary Oracle Notation for Hierarchical Queries

<table>
<thead>
<tr>
<th>Syntax Element</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONNECT BY PRIOR</td>
<td>Clause to specify a condition that establishes the link between parent and child rows</td>
</tr>
<tr>
<td>START WITH</td>
<td>Clause to specify a condition to identify the root rows in a hierarchical query</td>
</tr>
<tr>
<td>SIBLINGS</td>
<td>Keyword to indicate a sort order for siblings, rows with the same parent. Used in the ORDER BY clause.</td>
</tr>
<tr>
<td>LEVEL</td>
<td>Pseudo column to determine the hierarchical level of a row beginning with 1 for root rows</td>
</tr>
<tr>
<td>CONNECT_BY_ISLEAF</td>
<td>Pseudo column to determine leaf status of a row, 1 if a row has no related child rows, 0 otherwise</td>
</tr>
<tr>
<td>SYS_CONNECT_BY_PATH</td>
<td>Function to retrieve the path for a row using a column and separator character</td>
</tr>
<tr>
<td>CONNECT_BY_ROOT</td>
<td>Unary operator in the SELECT clause to retrieve the value of a specified column from a root row</td>
</tr>
<tr>
<td>PRIOR</td>
<td>Unary operator in the SELECT clause to reference the value of a specified column in the parent row</td>
</tr>
</tbody>
</table>

9.5.3 Extensions in the SQL Standard for Hierarchical Queries

The SQL standard, starting with SQL:1999, provided recursive common table expressions (CTE) to formulate hierarchical queries. CTEs can be used for other purposes besides hierarchical queries although they are only necessary for hierarchical queries so they were not previously introduced. Recursion means self-reference so a recursive CTE references itself. Recursive CTEs are supported by most major enterprise DBMSs including Oracle so recursive CTEs provide a reasonably portable notation compared to the proprietary nature of the Oracle CONNECT BY PRIOR clause.

The recursive CTE notation is more verbose than the Oracle notation although the recursive CTE notation uses only one syntax element compared to many syntax elements in the Oracle notation. The CTE notation involves two query blocks connected by a union operation followed by a second SELECT statement. Example 9.50 shows the basic pattern for a hierarchical query using a recursive CTE. Note the WITH keyword begins the CTE.

Example 9.50: Query pattern for Hierarchical Query using a Recursive CTE

The WITH keyword identifies the CTE name and column names. The first SELECT block (<CTEQuery1>), known as the anchor member, references the table with hierarchical data. The second SELECT block (<CTEQuery2>), known as the recursive member, references the CTE name. The SELECT statement after the WITH clause uses the CTE to generate the result. The semicolon terminates the entire statement including the WITH clause and SELECT statement.

```
WITH CTEName ( ColumnName* )
AS
  -- Anchor member (AM) referencing the hierarchical table.
  ( <CTEQuery1>
    UNION ALL
    -- Recursive member (RM) referencing the CTEName.
    <CTEQuery2> )
  -- Statement using CTEName
SELECT * FROM CTEName;
```

Example 9.51 shows a SQL statement conforming to the pattern in Example 9.50. Example 9.51 begins with the definition of the CTE following the WITH keyword. Faculty2CTE contains the columns to identify a row as well as a column to identify the hierarchical level of a column. The CTE definition contains the query blocks specifying the anchor and recursive members following the AS keyword. The connection between the query blocks occurs in the join condition and the LevelNo calculation in the recursive member.

Example 9.51: Basic Hierarchical Query using a Recursive CTE (Oracle)

Example 9.51 generates a result equivalent to Example 9.42. The anchor member (AM) retrieves the two roots of
the hierarchical table (FacSupervisor IS NULL). The recursive member (RM) repeatedly executes through each level below the roots. The LevelNo column in the CTE is 1 for the root rows in the anchor member. The LevelNo value is incremented by 1 for each level below the root in the recursive member. Note that LevelNo is a computed column, not the pseudo column used in the proprietary Oracle notation.

WITH Faculty2CTE ( FacNo, FacSupervisor, FacFirstName, FacLastName, FacHireDate, FacRank, FacSalary, LevelNo )
AS
( SELECT FacNo, FacSupervisor, FacFirstName, FacLastName, FacHireDate, FacRank, FacSalary, 1
  FROM Faculty2
  WHERE FacSupervisor IS NULL
  -- AM referencing Faculty2, the hierarchical table.
  UNION ALL
  SELECT F2.FacNo, F2.FacSupervisor, F2.FacFirstName, F2.FacLastName, F2.FacHireDate, F2.FacRank, F2.FacSalary, F2CTE.LevelNo + 1
  FROM Faculty2 F2 INNER JOIN Faculty2CTE F2CTE
  ON F2.FacSupervisor = F2CTE.FacNo
)
-- Statement using the CTE
SELECT * FROM Faculty2CTE
ORDER BY LevelNo, FacNo;

<table>
<thead>
<tr>
<th>FacNo</th>
<th>FacSupervisor</th>
<th>FacFirstName</th>
<th>FacLastName</th>
<th>FacHireDate</th>
<th>FacRank</th>
<th>FacSalary</th>
<th>LevelNo</th>
</tr>
</thead>
<tbody>
<tr>
<td>543-21-0987</td>
<td>VICTORIA</td>
<td>EMMANUEL</td>
<td></td>
<td>15-APR-01</td>
<td>PROF</td>
<td>120000</td>
<td>1</td>
</tr>
<tr>
<td>765-43-2109</td>
<td>NICKI</td>
<td>MACON</td>
<td></td>
<td>11-APR-02</td>
<td>ASSC</td>
<td>105000</td>
<td>1</td>
</tr>
<tr>
<td>111-22-3333</td>
<td>JOHN</td>
<td>MILLSION</td>
<td></td>
<td>01-MAY-05</td>
<td>PROF</td>
<td>110000</td>
<td>2</td>
</tr>
<tr>
<td>654-32-1098</td>
<td>LEONARD</td>
<td>FIBON</td>
<td></td>
<td>01-MAY-99</td>
<td>ASSC</td>
<td>70000</td>
<td>2</td>
</tr>
<tr>
<td>777-11-4321</td>
<td>AIMEE</td>
<td>MANNING</td>
<td></td>
<td>15-MAR-06</td>
<td>ASST</td>
<td>85000</td>
<td>2</td>
</tr>
<tr>
<td>987-65-4321</td>
<td>JULIA</td>
<td>MILLS</td>
<td></td>
<td>15-MAR-05</td>
<td>ASSC</td>
<td>95000</td>
<td>2</td>
</tr>
<tr>
<td>098-76-5432</td>
<td>LEONARD</td>
<td>VINCE</td>
<td></td>
<td>10-APR-00</td>
<td>ASST</td>
<td>55000</td>
<td>3</td>
</tr>
<tr>
<td>333-22-4444</td>
<td>SALLY</td>
<td>SCOTT</td>
<td></td>
<td>01-MAY-06</td>
<td>ASST</td>
<td>90000</td>
<td>3</td>
</tr>
<tr>
<td>555-66-7777</td>
<td>SUSAN</td>
<td>JONES</td>
<td></td>
<td>01-MAY-07</td>
<td>ASSC</td>
<td>125000</td>
<td>3</td>
</tr>
<tr>
<td>789-12-3210</td>
<td>JAIME</td>
<td>SANCHEZ</td>
<td></td>
<td>10-MAY-09</td>
<td>PROF</td>
<td>107000</td>
<td>3</td>
</tr>
<tr>
<td>876-54-3210</td>
<td>CRISTOPHER</td>
<td>COLAN</td>
<td></td>
<td>01-MAR-04</td>
<td>ASST</td>
<td>90000</td>
<td>3</td>
</tr>
<tr>
<td>888-33-1111</td>
<td>JAMES</td>
<td>BLOKE</td>
<td></td>
<td>15-APR-08</td>
<td>ASST</td>
<td>85000</td>
<td>3</td>
</tr>
</tbody>
</table>

Recursive Common Table Expression (CTE): Recursive CTEs are the SQL standard notation for hierarchical queries. A recursive CTE involves two query blocks connected by a union operation and a second SELECT statement. The second query block references the CTE, a self-reference. The second SELECT statement uses the CTE to generate the results.

This subsection finishes with two path exception query examples to demonstrate the recursive CTE notation on useful problems. Example 9.52 lists details about faculty earning more than a supervisor at any level. Example 9.53 lists details about faculty with a higher rank than a supervising faculty member.

Example 9.52: Path Exception Query using the Recursive CTE Notation (Oracle)

Example 9.52 retrieves faculty earning more than a supervisor at any level, generating the same result as Example 9.48. The join operations in each query block retrieve columns from the supervisor's row.

WITH Faculty2CTE ( FacNo, FacSupNo, FacLastName, FacSupLastName, FacSalary, FacSupSalary )
AS
( SELECT F1.FacNo, F1.FacSupervisor, F1.FacLastName, FacSalary, FacSupSalary )
F1Sup.FacLastName, F1.FacSalary, F1Sup.FacSalary
FROM Faculty2 F1 INNER JOIN Faculty2 F1Sup
ON F1.FacSupervisor = F1Sup.FacNo
UNION ALL
SELECT F2.FacNo, F2CTE.FacSupNo, F2.FacLastName,
F2CTE.FacSupLastName, F2.FacSalary, F2CTE.FacSupSalary
FROM Faculty2 F2 INNER JOIN Faculty2CTE F2CTE
ON F2.FacSupervisor = F2CTE.FacNo )
-- Statement using the CTE
SELECT *
FROM Faculty2CTE
WHERE FacSupNo <> FacNo AND FacSalary > FacSupSalary;

Example 9.53: Path Exception Query using the Recursive CTE notation (Oracle)

Example 9.52 retrieves faculty with a higher rank than a supervisor at any level, generating the same result as Example 9.49. The join operations in each query block retrieve columns from the supervisor's row.

WITH Faculty2CTE ( FacNo, FacSupNo, FacLastName,
FacSupLastName, FacRank, FacSupRank )
AS
( SELECT F1.FacNo, F1.FacSupervisor, F1.FacLastName,
F1Sup.FacLastName, F1.FacRank, F1Sup.FacRank
FROM Faculty2 F1 INNER JOIN Faculty2 F1Sup
ON F1.FacSupervisor = F1Sup.FacNo
UNION ALL
SELECT F2.FacNo, F2CTE.FacSupNo, F2.FacLastName,
F2CTE.FacSupLastName, F2.FacRank, F2CTE.FacSupRank
FROM Faculty2 F2 INNER JOIN Faculty2CTE F2CTE
ON F2.FacSupervisor = F2CTE.FacNo )
-- Statement using the CTE
SELECT *
FROM Faculty2CTE
WHERE FacSupNo <> FacNo
AND ( ( FacRank = 'PROF' AND FacSupRank = 'ASSC' )
OR ( FacRank = 'PROF' AND FacSupRank = 'ASST' )
OR ( FacRank = 'ASSC' AND FacSupRank = 'ASST' ) );

Closing Thoughts

Chapter 9 has presented advanced query formulation skills with an emphasis on complex matching problems and additional parts of the SQL SELECT statement. Complex matching problems involve the outer join operator with its variations (one-sided and full), the difference operator, and the division operator. In addition to more complex matching problems, this chapter explained the subtle effects of null values to provide a deeper understanding of query results.
and presented hierarchical queries that support retrieval from hierarchically-structured tables.

Two new parts of the SELECT statement were covered for complex matching problems. The keywords LEFT, RIGHT, and FULL as part of the join operator style support outer join operations. Nested queries are a query inside another query. To understand the effect of a nested query, you should look for tables used in both an outer and an inner query. If there are no common tables, the nested query executes one time (Type I nested query). Otherwise, the nested query executes one time for each row of the outer query (Type II nested query). Type I nested queries are typically used to formulate joins as part of the SELECT and DELETE statements. Type I nested queries with the NOT IN operator and Type II nested queries with the NOT EXISTS operator are useful for problems involving the difference operator. Type I nested queries in the HAVING clause are useful for problems involving the division operator.

For hierarchical queries, two SQL extensions were covered. Oracle provides proprietary notation including the CONNECT BY PRIOR clause, START WITH clause, SIBLINGS sort specification, LEVEL pseudo column, and several functions. The SQL standard notation involves recursive common table expressions involving the WITH statement containing two query blocks connected by a union operation and a SELECT statement to generate the hierarchical query results. The proprietary Oracle notation is more succinct but not portable to other DBMSs. Somewhat surprisingly, Oracle supports both its proprietary notation and the SQL standard notation.

Although advanced query skills are not as widely applied as the fundamental skills covered in Chapter 4, they are important when required. You may gain a competitive advantage by mastering these advanced query formulation skills.

Chapters 4 and 9 have covered important query formulation skills and a large part of the SELECT statement of SQL. Despite this significant coverage, there is still much left to learn. You need lots of practice to confidently formulate complex matching problems and hierarchical queries. In addition, you have not learned how to apply your query formulation skills to building applications. Chapter 10 applies your skills to building applications with views, while Chapter 11 applies your skills to stored procedures and triggers.

**Review Concepts**

- **Formulating one-sided outer joins with Access and Oracle**

  ```sql
  SELECT OfferNo, CourseNo, Offering.FacNo, Faculty.FacNo, FacFirstName, FacLastName
  FROM Offering LEFT JOIN Faculty
  ON Offering.FacNo = Faculty.FacNo
  WHERE CourseNo = 'IS480'
  ```

- **Formulating full outer joins using the FULL JOIN keyword (SQL:2011 and Oracle)**

  ```sql
  SELECT FacNo, FacFirstName, FacLastName, FacSalary, StdNo, StdFirstName, StdLastName, StdGPA
  FROM Faculty FULL JOIN Student
  ON Student.StdNo = Faculty.FacNo
  ```

- **Formulating full outer joins by combining two one-sided outer joins in Access**

  ```sql
  SELECT FacNo, FacFirstName, FacLastName, FacSalary, StdNo, StdFirstName, StdLastName, StdGPA
  FROM Faculty RIGHT JOIN Student
  ON Student.StdNo = Faculty.FacNo
  UNION
  SELECT FacNo, FacFirstName, FacLastName, FacSalary, StdNo, StdFirstName, StdLastName, StdGPA
  FROM Faculty LEFT JOIN Student
  ON Student.StdNo = Faculty.FacNo
  ```

- **Mixing inner and outer joins (Access and Oracle)**

  ```sql
  SELECT OfferNo, Offering.CourseNo, OffTerm, CrsDesc, Faculty.FacNo, FacFirstName, FacLastName
  FROM ( Faculty RIGHT JOIN Offering
  ON Offering.FacNo = Faculty.FacNo )
INNER JOIN Course
ON Course.CourseNo = Offering.CourseNo
WHERE OffYear = 2013

- Ambiguous query containing a non-preserved table (table with only matching rows in the result) in a one-sided outer join involved in another join or outer join operation
- Understanding that conditions in the WHERE or HAVING clause can use SELECT statements in addition to scalar (individual) values
- Identifying Type I nested queries by the IN keyword and the lack of a reference to a table used in an outer query
- Using a Type I nested query to formulate a join

```
SELECT DISTINCT StdNo, StdFirstName, StdLastName, StdMajor
FROM Student
WHERE Student.StdNo IN
  ( SELECT StdNo FROM Enrollment
    WHERE EnrGrade >= 3.5 )
```

- Using a Type I nested query inside a DELETE statement to test conditions on a related table

```
DELETE FROM Offering
WHERE Offering.FacNo IN
  ( SELECT FacNo FROM Faculty
    WHERE FacFirstName = 'LEONARD'
    AND FacLastName = 'VINCE' )
```

- Not using a Type I nested query for a join when a column from the nested query is needed in the final query result
- Identifying problem statements involving the difference operator: the words not or only relating two nouns in a sentence
- Limited SQL formulations for difference problems: Type I nested queries with the NOT IN operator, one-sided outer join with an IS NULL condition, and difference operation using the EXCEPT or MINUS keywords
- Using a Type I nested query with the NOT IN operator for difference problems involving a comparison of a single column

```
SELECT FacNo, FacFirstName, FacLastName, FacDept, FacSalary
FROM Faculty
WHERE FacNo NOT IN
  ( SELECT StdNo FROM Student )
```

- Identifying Type II nested queries by a reference to a table used in an outer query
- Using Type II nested queries with the NOT EXISTS operator for complex difference problems

```
SELECT FacNo, FacFirstName, FacLastName, FacDept, FacSalary
FROM Faculty
WHERE NOT EXISTS
  ( SELECT * FROM Student
    WHERE Student.StdNo = Faculty.FacNo )
```

- Using a nested query in the FROM clause to compute nested aggregates or aggregates for more than one grouping

```
SELECT T.CourseNo, T.CrsDesc, COUNT(*) AS NumOfferings,
       Avg(T.EnrollCount) AS AvgEnroll
FROM
  ( SELECT Course.CourseNo, CrsDesc,
        Offering.OfferNo, COUNT(*) AS EnrollCount
  FROM Offering, Enrollment, Course
  WHERE Offering.OfferNo = Enrollment.OfferNo
  AND Course.CourseNo = Offering.CourseNo
  GROUP BY Course.CourseNo, CrsDesc, Offering.OfferNo
```
Identifying problem statements involving the division operator: the word *every* or *all* connecting different parts of a sentence

Using the count method to formulate division problems

```sql
SELECT StdNo
FROM StdClub
GROUP BY StdNo
HAVING COUNT(*) = ( SELECT COUNT(*) FROM Club )
```

Evaluating a simple condition containing a null value in a column expression

Using three-valued logic and truth tables to evaluate compound conditions with null values

Understanding the result of aggregate calculations with null values

Understanding the result of grouping on a column with null values

Recognizing the need to formulate hierarchical queries for tables with hierarchical data

Using the CONNECT BY PRIOR and START WITH clauses to formulate basic hierarchical queries

```sql
SELECT FacNo, FacSupervisor, FacFirstName, FacLastName,
FacHireDate, FacSalary, FacRank, LEVEL
FROM Faculty2
START WITH FacSupervisor IS NULL
CONNECT BY PRIOR FacNo = FacSupervisor
ORDER BY LEVEL;
```

Applying the proprietary Oracle syntax elements including the LEVEL pseudo column, CONNECT_BY_ROOT operator, SYS_CONNECT_BY_PATH function, CONNECT_BY_ISLEAF pseudo column, and SIBLINGS keyword to formulate more complex hierarchical queries

Formulating path exception queries listing violations of monotonicity in path relationships such as subordinates earning more than their direct or indirect supervisors. Path exception queries use the closure of the hierarchy.

Recognizing recursive common table expressions, the SQL standard notation for formulating hierarchical queries on enterprise DBMSs

Questions

1. Explain a situation when a one-sided outer join is useful.
2. Explain a situation when a full outer join is useful.
3. How do you interpret the meaning of the LEFT and RIGHT JOIN keywords in the FROM clause?
4. What is the interpretation of the FULL JOIN keywords in the FROM clause?
5. How do you perform a full outer join in SQL implementations (such as Microsoft Access) that do not support the FULL JOIN keywords?
6. What is a nested query?
7. What is the distinguishing feature about the appearance of Type I nested queries?
8. What is the distinguishing feature about the appearance of Type II nested queries?
9. How many times is a Type I nested query executed as part of an outer query?
10. How is a Type I nested query like a procedure in a program?
11. How many times is a Type II nested query executed as part of an outer query?
12. How is a Type II nested query like a nested loop in a program?
13. What is the meaning of the IN comparison operator?
14. What is the meaning of the EXISTS comparison operator?
15. What is the meaning of the NOT EXISTS comparison operator?
16. When can you not use a Type I nested query to perform a join?
17. Why is a Type I nested query a good join method when you need a join in a DELETE statement?
18. Why does SQL:2011 permit nested queries in the FROM clause?
19. Identify two situations in which nested queries in the FROM clause are necessary.
20. How do you detect that a problem involves a division operation?
21. Explain the “count” method for formulating division problems.
22. Why is it sometimes necessary to use the DISTINCT keyword inside the COUNT function for division problems?
23. What is the result of a simple condition when a column expression in the condition evaluates to null?
24. What is a truth table?
25. How many values do truth tables have in the SQL:2011 standard?
26. How do you use truth tables to evaluate compound conditions?
27. How do null values affect aggregate calculations?
28. Explain why the following equation may not be true if Column1 or Column2 contains null values: $\text{SUM(Column1)} - \text{SUM(Column2)} = \text{SUM(Column1 - Column2)}$
29. How are null values handled in a grouping column?
30. In Access, how do you compensate for the lack of the DISTINCT keyword inside the COUNT function?
31. When can you use a Type I nested query with the NOT IN operator to formulate a difference operation in SQL?
32. When can you use a one-sided outer join with an IS NULL condition to formulate a difference operation in SQL?
33. When can you use a MINUS operation in SQL to formulate a difference operation in SQL?
34. What is the most general way to formulate difference operations in SQL statements?
35. Is the one-sided outer join operator associative?
36. What makes a query ambiguous?
37. What is the difference between Microsoft Access and Oracle in handling ambiguous queries?
38. What is a hierarchical query?
39. What is the difference between a self-join and a hierarchical query?
40. What is an important advantage for query language support for hierarchical queries?
41. What is a path exception query?
42. Explain the usage of the CONNECT BY PRIOR and START WITH clauses.
43. Explain the usage of the CONNECT_BY_ROOT operator and the SYS_CONNECT_BY_PATH function.
44. Explain the usage of the LEVEL pseudo column and the SIBLINGS keyword.
45. What is a recursive common table expression?
46. Briefly compare the proprietary Oracle notation for hierarchical queries to recursive common table expressions.
47. Explain the difference between the PRIOR operator and the CONNECT_BY_ROOT operator.

Problems

The problems use the tables of the Order Entry database introduced in the Problems section of Chapter 4. When formulating the problems, remember that the EmpNo foreign key in the OrderTbl table allows null values. An order does not have an associated employee if taken over the Internet.

1. Using a Type I nested query, list the customer number, name (first and last), and city of each customer who has a balance greater than $150 and placed an order in February 2013.
2. Using a Type II nested query, list the customer number, name (first and last), and city of each customer who has a balance greater than $150 and placed an order in February 2013.
3. Using two Type I nested queries, list the product number, the name, and the price of products with a price greater than $150 that were ordered on January 23, 2013.
4. Using two Type I nested queries and another join style, list the product number, name, and price of products with a price greater than $150 that were ordered in January 2013 by customers with balances greater than $400.
5. List the order number, order date, employee number, and employee name (first and last) of orders placed on January 23, 2013. List the order even if there is not an associated employee.
6. List the order number, order date, employee number, employee name (first and last), customer number, and customer name (first and last) of orders placed on January 23, 2013. List the order even if there is not an associated employee.
7. List all the people in the database. The resulting table should have all columns of the Customer and Employee tables. Match the Customer and Employee tables on first and last names. If a customer does not match any employees, the columns pertaining to the Employee table will be blank. Similarly for an employee who does not match any customers, the columns pertaining to the Customer table will be blank.
8. For each Ink Jet product ordered in January 2013, list the order number, order date, customer number, name (first and last), employee number (if present), employee name (first and last), quantity ordered, product number, and product name. Include products containing Ink Jet in the product name. Include both Internet (no employee) and phone orders (taken by an employee).
9. Using a Type II nested query, list the customer number and name of Colorado customers who have not placed orders in February 2013.
10. Repeat problem 9 using a Type I nested query with a NOT IN condition instead of a nested query. If the problem cannot be formulated in this manner, provide an explanation indicating the reason.
11. Repeat problem 9 using the MINUS keyword. Note that Access does not support the MINUS keyword. If the problem cannot be formulated in this manner, provide an explanation indicating the reason.
12. Repeat problem 9 using a one-sided outer join and an IS NULL condition. If the problem cannot be formulated in this manner, provide an explanation indicating the reason.
13. Using a Type II nested query, list the employee number, first name, and last name of employees in the (720) area code who have not taken orders. An employee is in the (720) area code if the employee phone number contains the string (720) in the beginning of the column value.
14. Repeat problem 13 using a Type I nested query with a NOT IN condition instead of a nested query. If the problem cannot be formulated in this manner, provide an explanation indicating the reason. (Hint: you need to think carefully about the effect of null values in the OrderTbl.EmpNo column.)
15. Repeat problem 9 using a one-sided outer join and an IS NULL condition. If the problem cannot be formulated in this manner, provide an explanation indicating the reason.
16. Repeat problem 9 using the MINUS keyword. Note that Access does not support the MINUS keyword. If the problem cannot be formulated in this manner, provide an explanation indicating the reason.
17. List the order number and order date of orders containing only one product with the words Ink Jet in the product description.
18. List the customer number and name (first and last) of customers who have ordered products only manufactured by Connex. Include only customers who have ordered at least one product manufactured by Connex. Remove duplicate rows from the result.
19. List the order number and order date of orders containing every product with the words Ink Jet in the product description.
20. List the product number and name of products contained on every order placed on January 7, 2013 through January 9, 2013.
21. List the customer number and name (first and last) of customers who have ordered every product manufactured by ColorMeg, Inc. in January 2013.
22. Using a Type I nested query, delete orders placed by customer Betty Wise in January 2013. The CASCADE DELETE action will delete related rows in the OrdLine table.

23. Using a Type I nested query, delete orders placed by Colorado customers that were taken by Landi Santos in January 2013. The CASCADE DELETE action will delete related rows in the OrdLine table.

24. List the order number and order date of orders in which any part of the shipping address (street, city, state, and zip) differs from the customer's address.

25. List the employee number and employee name (first and last) of employees who have taken orders in January 2013 from every Seattle customer.

26. For Colorado customers, compute the average amount of their orders. The average amount of a customer's orders is the sum of the amount (quantity ordered times the product price) on each order divided by the number of orders. The result should include the customer number, customer last name, and average order amount.

27. For Colorado customers, compute the average amount of their orders and the number of orders placed. The result should include the customer number, customer last name, average order amount, and number of orders placed. In Access, this problem is especially difficult to formulate.

28. For Colorado customers, compute the number of unique products ordered. If a product is purchased on multiple orders, it should be counted only one time. The result should include the customer number, customer last name, and number of unique products ordered.

29. For each employee with a commission less than 0.04, compute the number of orders taken and the average number of products per order. The result should include the employee number, employee last name, number of orders taken, and the average number of products per order. In Access, this problem is especially difficult to formulate as a single SELECT statement.

30. For each Connex product, compute the number of unique customers who ordered the product in January 2013. The result should include the product number, product name, and number of unique customers.

31. Please explain if the following SELECT statement is ambiguous. If it is ambiguous, provide a variation of this statement with possibly different results. The variation should be the identical SQL statement except for the order of operations in the FROM clause.

```sql
SELECT OrderTbl.OrdNo, OrdDate, Employee.EmpNo,
    EmpFirstName, EmpLastName, Customer.CustNo,
    CustFirstName, CustLastName, OrdLine.Qty,
    Product.ProdNo, ProdName
FROM ( ( ( OrderTbl LEFT JOIN Employee
    ON OrderTbl.EmpNo = Employee.EmpNo )
    INNER JOIN Customer
    ON Customer.CustNo = OrderTbl.CustNo )
    INNER JOIN OrdLine
    ON OrderTbl.OrdNo = OrdLine.OrdNo )
    INNER JOIN Product
    ON OrdLine.ProdNo = Product.ProdNo
```

32. Please explain if the following SELECT statement is ambiguous. If it is ambiguous, provide a variation of this statement with possibly different results. The variation should be the identical SQL statement except for the order of operations in the FROM clause.

```sql
SELECT OrderTbl.OrdNo, OrdDate, Employee.EmpNo,
    EmpFirstName, EmpLastName, Customer.CustNo,
    CustFirstName, CustLastName, OrdLine.Qty,
    Product.ProdNo, ProdName
FROM ( ( ( OrderTbl RIGHT JOIN Employee
    ON OrderTbl.EmpNo = Employee.EmpNo )
    INNER JOIN Customer
    ON Customer.CustNo = OrderTbl.CustNo )
    INNER JOIN OrdLine
    ON OrderTbl.OrdNo = OrdLine.OrdNo )
    INNER JOIN Product
    ON OrdLine.ProdNo = Product.ProdNo
```
33. Add 1 to the order quantity of each product ordered by customer number C9943201 on January 23, 2013. In the UPDATE statement, you should not use an order number constant. You need to reference the related tables in the UPDATE statement. Write the UPDATE statement in both Access and Oracle using a Type I nested query.

34. Revise problem 33 to add 1 to the order quantity of each product ordered by Harry Sanders on January 23, 2013. In the UPDATE statement, you should not use an order number or customer number constant. You need to reference the related tables in the UPDATE statement. Write the UPDATE statement in both Access and Oracle using a Type I nested query.

35. Revise problem 33 to add 1 to the order quantity of each product with product name containing the string “Color Inkjet” ordered by Harry Sanders on January 23, 2013. In the UPDATE statement, you should not use an order number, customer number, or product number constant. You need to reference the related tables in the UPDATE statement. Write the UPDATE statement in both Access and Oracle using a Type I nested query.

Null Value Problems

The following problems are based on the Product and Employee tables of the Order Entry database. The tables are repeated below for your convenience. The ProdNextShipDate column contains the next expected shipment date for the product. If the value is null, a new shipment has not been arranged. A shipment may not be scheduled for a variety of reasons, such as the large quantity on hand or unavailability of the product from the manufacturer. In the Employee table, the commission rate can be null indicating a commission rate has not been assigned. A null value for SupEmpNo indicates that the employee has no supervisor.

### Product

<table>
<thead>
<tr>
<th>ProdNo</th>
<th>ProdName</th>
<th>ProdMfg</th>
<th>ProdQOH</th>
<th>ProdPrice</th>
<th>ProdNextShipDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0036566</td>
<td>17 inch Color Monitor</td>
<td>ColorMeg, Inc.</td>
<td>12</td>
<td>$169.00</td>
<td>2/20/2013</td>
</tr>
<tr>
<td>P0036577</td>
<td>19 inch Color Monitor</td>
<td>ColorMeg, Inc.</td>
<td>10</td>
<td>$319.00</td>
<td>2/20/2013</td>
</tr>
<tr>
<td>P1114590</td>
<td>R3000 Color Laser Printer</td>
<td>Connex</td>
<td>5</td>
<td>$699.00</td>
<td>1/22/2013</td>
</tr>
<tr>
<td>P1412138</td>
<td>10 Foot Printer Cable</td>
<td>Ethlite</td>
<td>100</td>
<td>$12.00</td>
<td></td>
</tr>
<tr>
<td>P1445671</td>
<td>8-Outlet Surge Protector</td>
<td>Intersafe</td>
<td>33</td>
<td>$14.99</td>
<td></td>
</tr>
<tr>
<td>P1556678</td>
<td>CVP Ink Jet Color Printer</td>
<td>Connex</td>
<td>8</td>
<td>$99.00</td>
<td>1/22/2013</td>
</tr>
<tr>
<td>P3455443</td>
<td>Color Ink Jet Cartridge</td>
<td>Connex</td>
<td>24</td>
<td>$38.00</td>
<td>1/22/2013</td>
</tr>
<tr>
<td>P4200344</td>
<td>36-Bit Color Scanner</td>
<td>UV Components</td>
<td>16</td>
<td>$199.99</td>
<td>1/29/2013</td>
</tr>
<tr>
<td>P6677900</td>
<td>Black Ink Jet Cartridge</td>
<td>Connex</td>
<td>44</td>
<td>$25.69</td>
<td></td>
</tr>
<tr>
<td>P9995676</td>
<td>Battery Back-up System</td>
<td>Cybercx</td>
<td>12</td>
<td>$89.00</td>
<td>2/1/2013</td>
</tr>
</tbody>
</table>

### Employee

<table>
<thead>
<tr>
<th>EmpNo</th>
<th>EmpFirstName</th>
<th>EmpLastName</th>
<th>EmpPhone</th>
<th>EmpEMail</th>
<th>SupEmpNo</th>
<th>EmpCommRate</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1329594</td>
<td>Landi Santos</td>
<td>(303) 789-1234</td>
<td><a href="mailto:LSantos@bigco.com">LSantos@bigco.com</a></td>
<td>E8843211</td>
<td>0.02</td>
<td></td>
</tr>
<tr>
<td>E8544399</td>
<td>Joe Jenkins</td>
<td>(303) 221-9875</td>
<td><a href="mailto:JJenkins@bigco.com">JJenkins@bigco.com</a></td>
<td>E8843211</td>
<td>0.02</td>
<td></td>
</tr>
<tr>
<td>E8843211</td>
<td>Amy Tang</td>
<td>(303) 556-4321</td>
<td><a href="mailto:ATang@bigco.com">ATang@bigco.com</a></td>
<td>E9884325</td>
<td>0.04</td>
<td></td>
</tr>
<tr>
<td>E9345771</td>
<td>Colin White</td>
<td>(303) 221-4453</td>
<td><a href="mailto:CWhite@bigco.com">CWhite@bigco.com</a></td>
<td>E9884325</td>
<td>0.04</td>
<td></td>
</tr>
<tr>
<td>E9884325</td>
<td>Thomas Johnson</td>
<td>(303) 556-9987</td>
<td><a href="mailto:TJohnson@bigco.com">TJohnson@bigco.com</a></td>
<td></td>
<td>0.05</td>
<td></td>
</tr>
<tr>
<td>E9954302</td>
<td>Mary Hill</td>
<td>(303) 556-9871</td>
<td><a href="mailto:MHill@bigco.com">MHill@bigco.com</a></td>
<td>E8843211</td>
<td>0.02</td>
<td></td>
</tr>
<tr>
<td>E9973110</td>
<td>Theresa Beck</td>
<td>(720) 320-2234</td>
<td><a href="mailto:TBeck@bigco.com">TBeck@bigco.com</a></td>
<td>E9884325</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. Identify the result rows in the following SELECT statement. Both Access and Oracle versions of the statement are shown.

   **Access:**
   ```sql
   SELECT * 
   FROM Product 
   WHERE ProdNextShipDate = #1/22/2013#
   ```
Oracle:
SELECT *
FROM Product
WHERE ProdNextShipDate = '22-Jan-2013';

2. Identify the result rows in the following SELECT statement:
   Access:
   SELECT *
   FROM Product
   WHERE ProdNextShipDate = #1/22/2013#
       AND ProdPrice < 100

   Oracle:
   SELECT *
   FROM Product
   WHERE ProdNextShipDate = '22-Jan-2013'
       AND ProdPrice < 100;

3. Identify the result rows in the following SELECT statement:
   Access:
   SELECT *
   FROM Product
   WHERE ProdNextShipDate = #1/22/2013#
       OR ProdPrice < 100

   Oracle:
   SELECT *
   FROM Product
   WHERE ProdNextShipDate = '22-Jan-2013'
       OR ProdPrice < 100;

4. Determine the result of the following SELECT statement:
   SELECT COUNT(*) AS NumRows,
       COUNT(ProdNextShipDate) AS NumShipDates
   FROM Product

5. Determine the result of the following SELECT statement:
   SELECT ProdNextShipDate, COUNT(*) AS NumRows
   FROM Product
   GROUP BY ProdNextShipDate

6. Determine the result of the following SELECT statement:
   SELECT ProdMfg, ProdNextShipDate, COUNT(*) AS NumRows
   FROM Product
   GROUP BY ProdMfg, ProdNextShipDate

7. Determine the result of the following SELECT statement:
   SELECT ProdNextShipDate, ProdMfg, COUNT(*) AS NumRows
   FROM Product
   GROUP BY ProdNextShipDate, ProdMfg

8. Identify the result rows in the following SELECT statement:
   SELECT EmpFirstName, EmpLastName
   FROM Employee
   WHERE EmpCommRate > 0.02

9. Determine the result of the following SELECT statement:
   SELECT SupEmpNo, AvG(EmpCommRate) AS AvgCommRate
   FROM Employee
   GROUP BY SupEmpNo
10. Determine the result of the following SELECT statement. The statement computes the average commission rate of subordinate employees. The result includes the employee number, first name, and last name of the supervising employee as well as the average commission amount of the subordinate employees.

```sql
SELECT Emp.SupEmpNo, Sup.EmpFirstName, Sup.EmpLastName,
       AVG(Emp.EmpCommRate) AS AvgCommRate
FROM Employee Emp, Employee Sup
WHERE Emp.SupEmpNo = Sup.EmpNo
GROUP BY Emp.SupEmpNo, Sup.EmpFirstName, Sup.EmpLastName
```

11. Using your knowledge of null value evaluation, explain why these two SQL statements generate different results for the Order Entry Database. You should remember that null values are allowed for `OrderTbl.EmpNo`.

```sql
SELECT EmpNo, EmpLastName, EmpFirstName
FROM Employee
WHERE EmpNo NOT IN
  ( SELECT EmpNo FROM OrderTbl WHERE EmpNo IS NOT NULL )

SELECT EmpNo, EmpLastName, EmpFirstName
FROM Employee
WHERE EmpNo NOT IN
  ( SELECT EmpNo FROM OrderTbl )
```

Hierarchical Query Problems

The following problems use the `Employee2` table, extended from the `Employee` table of the Order Entry database. Some of the columns have been dropped and others added for these problems. Here are comments about the extensions to the `Employee` table:

- As in the `Employee` table, `EmpNo` is the primary key.
- As in the `Employee` table, the `SupEmpNo` column is a foreign key referencing the `EmpNo` column.
- The `EmpSalary` column should be smaller for subordinates than supervisors, both direct and indirect.
- The `EmpGrade` column should be larger for subordinates than supervisors, both direct and indirect.
- The `EmpCommRate` should be larger for subordinates than supervisors, both direct and indirect.

<table>
<thead>
<tr>
<th>EmpNo</th>
<th>EmpFirstName</th>
<th>EmpLastName</th>
<th>EmpSalary</th>
<th>EmpGrade</th>
<th>SupEmpNo</th>
<th>EmpCommRate</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1329594</td>
<td>Landi</td>
<td>Santos</td>
<td>36000</td>
<td>2</td>
<td>E8843211</td>
<td>0.050</td>
</tr>
<tr>
<td>E8543999</td>
<td>Joe</td>
<td>Jenkins</td>
<td>30000</td>
<td>4</td>
<td>E8843211</td>
<td>0.040</td>
</tr>
<tr>
<td>E8843211</td>
<td>Amy</td>
<td>Tang</td>
<td>35000</td>
<td>3</td>
<td>E9884325</td>
<td>0.030</td>
</tr>
<tr>
<td>E9345771</td>
<td>Colin</td>
<td>White</td>
<td>40000</td>
<td>2</td>
<td>E9884325</td>
<td>0.040</td>
</tr>
<tr>
<td>E9884325</td>
<td>Thomas</td>
<td>Johnson</td>
<td>60000</td>
<td>2</td>
<td>E9884325</td>
<td>0.035</td>
</tr>
<tr>
<td>E9954302</td>
<td>Mary</td>
<td>Hill</td>
<td>37000</td>
<td>3</td>
<td>E8843211</td>
<td>0.050</td>
</tr>
<tr>
<td>E9973110</td>
<td>Theresa</td>
<td>Beck</td>
<td>42000</td>
<td>1</td>
<td>E9884325</td>
<td>0.033</td>
</tr>
<tr>
<td>E1234567</td>
<td>Claire</td>
<td>Adams</td>
<td>50000</td>
<td>1</td>
<td>E1234567</td>
<td>0.025</td>
</tr>
<tr>
<td>E7654321</td>
<td>Yanjuan</td>
<td>Pong</td>
<td>40000</td>
<td>3</td>
<td>E1234567</td>
<td>0.030</td>
</tr>
<tr>
<td>E4321098</td>
<td>Miguel</td>
<td>Sanchez</td>
<td>52000</td>
<td>2</td>
<td>E1234567</td>
<td>0.033</td>
</tr>
<tr>
<td>E6543210</td>
<td>Bradley</td>
<td>Smith</td>
<td>35000</td>
<td>3</td>
<td>E7654321</td>
<td>0.045</td>
</tr>
<tr>
<td>E5432109</td>
<td>Susan</td>
<td>Henry</td>
<td>41000</td>
<td>2</td>
<td>E7654321</td>
<td>0.050</td>
</tr>
<tr>
<td>E9876543</td>
<td>Michael</td>
<td>Roberts</td>
<td>55000</td>
<td>2</td>
<td>E4321098</td>
<td>0.040</td>
</tr>
<tr>
<td>E8765432</td>
<td>Melissa</td>
<td>Cole</td>
<td>42000</td>
<td>3</td>
<td>E4321098</td>
<td>0.033</td>
</tr>
</tbody>
</table>

1. Draw organizational charts, similar to Figures 9.2 and 9.3, to depict the hierarchical organization among the rows in the `Employee2` table.
2. Using the Oracle proprietary notation, write a SELECT statement to retrieve the closure (combinations of employee and supervisor, direct or indirect) starting with the root employees having null values for SupEmpNo. The result should contain EmpNo, EmpLastName, EmpSalary, EmpGrade, SupEmpNo, EmpCommRate, root employee number, and the LEVEL pseudo column. Order the result by EmpLastName and the LEVEL pseudo column. Note that you will need to rename the LEVEL pseudo column in the output list to reference it in the ORDER BY clause.

3. Using the Oracle proprietary notation, write a SELECT statement to retrieve the closure (combinations of employee and supervisor, direct or indirect) starting with the root employees having null values for SupEmpNo. The result should contain EmpNo, EmpLastName, root employee number, and path using last name as the row identifier and / as the separator. Sort the siblings by employee last name.

4. Using the Oracle proprietary notation, write a SELECT statement to retrieve the closure (combinations of employee and supervisor, direct or indirect) starting with the root employees having null values for SupEmpNo. The result should contain EmpLastName arranged to depict the hierarchical structure using the LPAD function (See Example 9.43.), EmpNo, EmpSalary, EmpGrade, and EmpCommRate. Sort the siblings by employee last name.

5. Using the Oracle proprietary notation, summarize each supervisor (non-leaf row) on the number of subordinates (direct and indirect) and sum of the salary of the subordinates. The result should include the employee last name, sum of the salary, and number (count) of subordinates (both direct and indirect). Only include non-leaf nodes in the final result.

6. Using the Oracle proprietary notation, list details about employees with a larger salary than a supervisor, direct or indirect. The result should include the employee number, last name, and salary of both the employee and supervisor as well as the path from the supervisor to the employee using the last name to identify rows on the path and / as the separator character.

7. Using the Oracle proprietary notation, list details about employees with a smaller grade than a supervisor, direct or indirect. The result should include the employee number, last name, and grade of both the employee and supervisor as well as the path from the supervisor to the employee using the last name to identify rows on the path and / as the separator character.

8. Using the Oracle proprietary notation, list details about employees with a smaller commission rate than a supervisor, direct or indirect. The result should include the employee number, last name, and commission rates of the employee and supervisor as well as the path from the supervisor to the employee using the last name to identify rows on the path and / as the separator character.

9. Using the Oracle proprietary notation, summarize the commission amounts earned on January 2013 sales for each employee supervised by Johnson either directly or indirectly. The earned commission is calculated by the employee's commission rate times the amount of sales on orders taken by the employee. The amount of sales on an order is calculated by summing the quantity ordered times price for each product on an order. You should combine the Employee2 and OrderTbl tables on employee number to link employees with orders. The result should include the employee number, employee last name, last name of the employee's direct supervisor, hierarchical level, and sum of the earned commission.

10. Using the recursive CTE notation, list details about employees with a larger salary than a supervisor, direct or indirect. The result should include the employee number, last name, and salary of both the employee and supervisor.

11. Using the recursive CTE notation list details about employees with a smaller grade than a supervisor, direct or indirect. The result should include the employee number, last name, and grade of both the employee and supervisor.

12. Using the recursive CTE notation, list details about employees with a smaller commission than a supervisor, direct or indirect. The result should include the employee number, last name, and commission of both the employee and supervisor.

References for Further Study

Most textbooks for the business student do not cover query formulation and SQL in as much detail as here. For advanced SQL coverage beyond the coverage in this chapter, you should consult the summary of SQL books at www.ocelot.ca/books.htm. For new features in SQL:1999, you should read Melton and Simon (2001). Groff and Weinberg (1999) cover the various notations for outer joins available in commercial DBMSs. For product-specific SQL advice,
the sqlblog.com site features forums about a number of DBMSs including Microsoft SQL Server and open source products. The Database Journal (www.databasejournal.com) provides articles, tutorials, and resources about many DBMS products. Oracle documentation can be found at the Oracle Technet site (www.oracle.com/technetwork). The Mimer Developer website has validators (http://developer.mimer.se/validator) for the SQL standards as aids to writing portable SQL statements.

Appendix 9.A: Usage of Multiple Statements in Microsoft Access

In Microsoft Access, you can use multiple SELECT statements instead of nested queries in the FROM clause. Using multiple statements can provide simpler formulation in some cases than using nested queries in the FROM clause. For example, instead of using DISTINCT inside COUNT as in Example 9.29, you can use a stored query with the DISTINCT keyword following the SELECT keyword. In Example 9A.1, the first stored query (Temp9A-1) finds the unique combinations of faculty name and course number. Note the use of the DISTINCT keyword to eliminate duplicates. The second stored query (Temp9A-2) finds the unique course numbers in the Offering table. The final query combines the two stored queries. Note that you can use stored queries similar to the way tables are used. Simply use the stored query name in the FROM clause.

Example 9A.1: Using Stored Queries Instead of Nested Queries in the FROM Clause

List the name of faculty who teach in at least one section of all fall 2012 information systems courses. The result is identical to that in Example 9.29.

Temp9A-1:

```
SELECT DISTINCT Faculty.FacNo, FacFirstName, FacLastName, CourseNo
FROM Faculty, Offering
WHERE Faculty.FacNo = Offering.FacNo
AND OffTerm = 'FALL' AND OffYear = 2012
AND CourseNo LIKE 'IS*'
```

Temp9A-2:

```
SELECT DISTINCT CourseNo
FROM Offering
WHERE OffTerm = 'FALL' AND OffYear = 2012
AND CourseNo LIKE 'IS*'
```

```
SELECT FacNo, FacFirstName, FacLastName
FROM [Temp9A-1]
GROUP BY FacNo, FacFirstName, FacLastName
HAVING COUNT(*) = ( SELECT COUNT(*) FROM [Temp9A-2] )
```


This appendix summarizes the SQL:2011 syntax for nested SELECT statements (subqueries) and outer join operations presented in Chapter 9. For the syntax of other variations of the nested SELECT and outer join operations not presented in Chapter 9, consult an SQL reference book. Nested SELECT statements can be used in the FROM clause and the WHERE clause of the SELECT, UPDATE, and DELETE statements. The conventions used in the syntax notation are identical to those used at the end of Chapter 3.

Expanded Syntax for Nested Queries in the FROM Clause

```
<Table-Specification>:
   { <Simple-Table> | -- defined in Chapter 4
      <Join-Operation> | -- defined in Chapter 4
      <Simple-Select> [ [ AS ] AliasName ] }
   -- <Simple-Select> is defined in Chapter 4
```
Expanded Syntax for Row Conditions

<Row-Condition>:
{ <Simple-Condition> | -- defined in Chapter 4
  <Compound-Condition> | -- defined in Chapter 4
  <Exists-Condition> |
  <Element-Condition> }

<Exists-Condition>: [ NOT ] EXISTS <Simple-Select>

<Simple-Select>: -- defined in Chapter 4

<Element-Condition>:
  <Scalar-Expression> <Element-Operator>( <Simple-Select> )

<Element-Operator>:
{ = | < | > | >= | <= | <> | [ NOT ] IN }

<Scalar-Expression>: -- defined in Chapter 4

Expanded Syntax for Group Conditions

<Simple-Group-Condition>: -- Last choice is new
{ <Column-Expression> ComparisonOperator
  <Column-Expression> |
  <Column-Expression> [ NOT ] IN ( Constant* ) |
  <Column-Expression> BETWEEN <Column-Expression>
  AND <Column-Expression> |
  <Column-Expression> IS [NOT] NULL |
  ColumnName [ NOT ] LIKE StringPattern |
  <Exists-Condition> |
  <Column-Expression> <Element-Operator> <Simple-Select> }

(Column-Expression>: -- defined in Chapter 4

Expanded Syntax for Outer Join Operations

<Join-Operation>:
{ <Simple-Table> <Join-Operator> <Simple-Table>
  ON <Join-Condition> |
  { <Simple-Table> | <Join-Operation> } <Join-Operator>
  { <Simple-Table> | <Join-Operation> }
  ON <Join-Condition> |
  ( <Join-Operation> ) } }

<Join-Operator>:
{ [ INNER ] JOIN |
  LEFT [ OUTER ] JOIN |
  RIGHT [ OUTER ] JOIN |
  FULL [ OUTER ] JOIN }
Expanded Syntax for Recursive Common Table Expressions

WITH CTEName ( ColumnName* )
AS
-- Anchor member (AM) referencing the hierarchical table.
( <Simple-Select> -- Using expanded <Table-Specification> above
UNION ALL
-- Recursive member (RM) referencing CTEName.
<Simple-Select> ) -- Using expanded <Table-Specification> above
-- Statement using CTEName
<Select-Statement> ; -- Using expanded <Table-Specification> above

Appendix 9.C: Oracle 8i Notation for Outer Joins

Until the Oracle 9i release, Oracle used a proprietary extension for one-sided outer joins. To express a one-sided outer join in Oracle 8i SQL, you need to use the notation (+) as part of a join condition in the WHERE clause. You place the (+) notation just after the join column of the null table, that is, the table with null values in the result. In contrast, the SQL:2011 LEFT and RIGHT keywords are placed after the table in which nonmatching rows are preserved in the result. The Oracle 8i formulations of Examples 9.1, 9.2, 9.4, 9.5, and 9.6 demonstrate the (+) notation.

Example 9.1 (Oracle 8i): One-Sided Outer Join with Outer Join Symbol on the Right Side of a Join Condition

The (+) notation is placed after the Faculty.FacNo column in the join condition because Faculty is the null table in the result.

```sql
SELECT OfferNo, CourseNo, Offering.FacNo, Faculty.FacNo,
       FacFirstName, FacLastName
FROM Faculty, Offering
WHERE Offering.FacNo = Faculty.FacNo (+)
AND CourseNo LIKE 'IS%'
```

Example 9.2 (Oracle 8i): One-Sided Outer Join with Outer Join Symbol on the Left Side of a Join Condition

The (+) notation is placed after the Faculty.FacNo column in the join condition because Faculty is the null table in the result.

```sql
SELECT OfferNo, CourseNo, Offering.FacNo, Faculty.FacNo,
       FacFirstName, FacLastName
FROM Faculty, Offering
WHERE Faculty.FacNo (+) = Offering.FacNo
AND CourseNo LIKE 'IS%'
```

Example 9.4 (Oracle 8i): Full Outer Join Using a Union of Two One-Sided Outer Joins

Combine the Faculty and Student tables using a full outer join. List the Social Security number, the name (first and last), the salary (faculty only), and the GPA (students only) in the result.

```sql
SELECT FacNo, FacFirstName, FacLastName, FacSalary,
       StdNo, StdFirstName, StdLastName, StdGPA
FROM Faculty, Student
WHERE Student.StdNo = Faculty.FacNo (+)
UNION
SELECT FacNo, FacFirstName, FacLastName, FacSalary,
       StdNo, StdFirstName, StdLastName, StdGPA
FROM Faculty, Student
WHERE Student.StdNo (+) = Faculty.FacNo
```
Example 9.5 (Oracle 8i): Mixing a One-Sided Outer Join and an Inner Join

Combine columns from the Faculty, Offering, and Course tables for IS courses offered in 2013. Include a row in the result even if there is not an assigned instructor.

```
SELECT OfferNo, Offering.CourseNo, OffTerm, CrsDesc,
       Faculty.FacNo, FacFirstName, FacLastName
FROM Faculty, Offering, Course
WHERE Offering.FacNo = Faculty.FacNo (+)
  AND Course.CourseNo = Offering.CourseNo
  AND Course.CourseNo LIKE 'IS%' AND OffYear = 2013
```

Example 9.6 (Oracle 8i): Mixing a One-Sided Outer Join and Two Inner Joins

List the rows of the Offering table where there is at least one student enrolled, in addition to the requirements of Example 9.6. Remove duplicate rows when there is more than one student enrolled in an offering.

```
SELECT DISTINCT Offering.OfferNo, Offering.CourseNo,
               OffTerm, CrsDesc, Faculty.FacNo, FacFirstName,
               FacLastName
FROM Faculty, Offering, Course, Enrollment
WHERE Offering.FacNo = Faculty.FacNo (+)
  AND Course.CourseNo = Offering.CourseNo
  AND Offering.OfferNo = Enrollment.OfferNo
  AND Course.CourseNo LIKE 'IS%' AND OffYear = 2013
```

It should be noted that the proprietary extension of Oracle is inferior to the SQL:2011 notation. The proprietary extension does not allow specification of the order of performing outer joins. This limitation can be problematic on difficult problems involving more than one outer join. Thus, you should use the SQL:2011 outer join syntax although later Oracle versions (9i and beyond) still support the proprietary extension using the (+) symbol.