John L. Viescas
Michael J. Hernandez

Foreword by Keith W. Hare
Convenor, International SQL Standards Committee

SQL Queries for Mere Mortals®

Second Edition

A Hands-On Guide to Data Manipulation in SQL

Software-independent approach!
If you work with database software such as

- **Access**
- **MS SQL Server**
- **Oracle**
- **DB2**
- **MySQL**
- **Ingres**

or any other SQL-based program, this book could save you hours of time and aggravation—before you write a single query!

Praise for *SQL Queries for Mere Mortals®, Second Edition*

Unless you are working at a very advanced level, this is the only SQL book you will ever need. The authors have taken the mystery out of complex queries and explained principles and techniques with such clarity that a "Mere Mortal" will indeed be empowered to perform the superhuman. Do not walk past this book!

— Graham Mandeno, Database Consultant

I learned SQL primarily from the first edition of this book, and I am pleased to see a second edition of this book so that others can continue to benefit from its organized presentation of the language. Starting from how to design your tables so that SQL can be effective (a common problem for database beginners), and then continuing through the various aspects of SQL construction and capabilities, the reader can become a moderate expert upon completing the book and its samples. Learning how to convert a question in English into a meaningful SQL statement will greatly facilitate your mastery of the language. Numerous examples from real life will help you visualize how to use SQL to answer the questions about the data in your database. Just one of the "watch out for this trap" items will save you more than the cost of the book when you avoid that problem when writing your queries. I highly recommend this book if you want to tap the full potential of your database.

— Kenneth D. Snell, Ph.D., Database Designer/Programmer

I don't think they do this in public schools any more, and it is a shame, but do you remember in the seventh and eighth grades when you learned to diagram a sentence? Those of you who do may no longer remember how you did it, but all of you do write better sentences because of it. John Viescas and Mike Hernandez must have remembered because they take everyday English queries and literally translate them into SQL. This is an important book for all database designers. It takes the complexity of mathematical Set Theory and of First Order Predicate Logic, as outlined in E. F. Codd's original treatise on relational database design, and makes it easy for anyone to understand. If you want an elementary- through intermediate-level course on SQL, this is the one book that is a requirement, no matter how many others you buy.

— Arvin Meyer, MCP, MVP

SQL Queries for Mere Mortals, Second Edition, provides a step-by-step, easy-to-read introduction to writing SQL queries. It includes hundreds of examples with detailed explanations. This book provides the tools you need to understand, modify, and create SQL queries.

— Keith W. Hare, Convenor, ISO/IEC JTC1 SC32 WG3—
the International SQL Standards Committee

Even in this day of wizards and code generators, successful database developers still require a sound knowledge of Structured Query Language (SQL, the standard language for communicating with most database systems). In this book, John and Mike do a marvelous job of making what's usually a dry and difficult subject come alive, presenting the material with humor in a logical manner, with plenty of relevant examples. I would say that this book should feature prominently in the collection on the bookshelf of all serious developers, except that I'm sure it'll get so much use that it won't spend much time on the shelf!

— Doug Steele, Microsoft Access Developer and author

SQL Queries *for* Mere Mortals®
Second Edition

Addison-Wesley presents the
For Mere Mortals® Series

Series Editor: Michael J. Hernandez

The goal of the *For Mere Mortals® Series* is to present you with information on important technology topics in an easily accessible, common-sense manner. The primary audience for *Mere Mortals* books is that of readers who have little or no background or formal training in the subject matter. Books in the Series avoid dwelling on the theoretical and instead take you right into the heart of the topic with a matter-of-fact, hands-on approach. The books are not designed to address all the intricacies of a given technology, but they do not avoid or gloss over complex, essential issues either. Instead, they focus on providing core, foundational knowledge in a way that is easy to understand and that will properly ground you in the topic. This practical approach provides you with a smooth learning curve and helps you to begin to solve your real-world problems immediately. It also prepares you for more advanced treatments of the subject matter, should you decide to pursue them, and even enables the books to serve as solid reference material for those of you with more experience. The software-independent approach taken in most books within the Series also teaches the concepts in such a way that they can be applied to whatever particular application or system you may need to use.

Titles in the Series:

Project Management for Mere Mortals®
Claudia M. Baca. ISBN: 0321423453

User Interface Design for Mere Mortals™
Eric Butow. ISBN: 0321447735

Database Design for Mere Mortals®, Second Edition: A Hands-On Guide to Relational Database Design
Michael J. Hernandez. ISBN: 0201752840

Microsoft Office Project for Mere Mortals®: Solving the Mysteries of Microsoft Office Project
Patti Jansen. ISBN: 0321423429

UML for Mere Mortals®
Robert A. Maksimchuk and Eric J. Naiburg. ISBN: 0321246241

VSTO for Mere Mortals™
Kathleen McGrath and Paul Stubbs. ISBN: 0321426711

SQL Queries for Mere Mortals®: A Hands-On Guide to Data Manipulation in SQL, Second Edition
John L. Viescas and Michael J. Hernandez. ISBN: 0321444434

SQL Queries *for* Mere Mortals®
Second Edition

A Hands-On Guide to Data Manipulation in SQL

John L. Viescas
Michael J. Hernandez

Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco •
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

Library of Congress Cataloging-in-Publication Data
Viescas, John L., 1947-
SQL queries for mere mortals : a hands-on guide to data manipulation in SQL / John L. Viescas and Michael J. Hernandez. Ñ 2nd ed.
p. cm.
On t.p. of previous ed. Michael J. HernandezÕs name appeared first.
Includes index.
ISBN 0-321-44443-4 (pbk. : alk. paper)
1. SQL (Computer program language) 2. Database searching. I. Hernandez, Michael J. (Michael James), 1955- II. Viescas, John L., 1947- SQL queries for mere mortals. III. Title.

QA76.73.S67H48 2007
005.75Õ85Ñdc22 2007026881

ISBN-13: 978-0-321-44443-1
ISBN-10: 0-321-44443-4

Text printed in the United States on recycled paper at Courier in Westford, Massachusetts.
Eighth printing, September 2013

Editor-in-Chief: Karen Gettman
Acquisitions Editor: Chuck Toporek
Managing Editor: John Fuller
Project Editor: Elizabeth Ryan
Copy Editor: Chrysta Meadowbrooke
Indexer: Coughlin Indexing
Proofreader: Mike Shelton
Technical Reviewers: Keith Hare, Stephen Forte
Cover Designer: Alan Clements
Composition: Pine Tree Composition

Contents

Foreword **xvii**

Preface **xix**

About the Authors **xxi**

Introduction **xxiii**

Are You a Mere Mortal? xxiii
About This Book xxiv
What This Book Is Not xxv
How to Use This Book xxvi
Reading the Diagrams Used in This Book xxvii
Sample Databases Used in This Book xxx
"Follow the Yellow Brick Road" xxxii

PART I Relational Databases and SQL 1

CHAPTER 1 What Is Relational? 3

Topics Covered in This Chapter 3
Types of Databases 3
A Brief History of the Relational Model 4
  In the Beginning . . . 4
  Relational Database Software 5
Anatomy of a Relational Database 6
  Tables 6
  Fields 7
  Records 8
  Keys 8

Views 9
Relationships 11
What's in It for You? 15
Where Do You Go from Here? 16
Summary 17

CHAPTER 2 Ensuring Your Database Structure Is Sound 19

Topics Covered in This Chapter 19
Why Is This Chapter Here? 19
Why Worry about Sound Structures? 20
Fine-Tuning Fields 21
What's in a Name? (Part One) 21
Smoothing Out the Rough Edges 23
Resolving Multipart Fields 25
Resolving Multivalued Fields 27
Fine-Tuning Tables 30
What's in a Name? (Part Two) 30
Ensuring a Sound Structure 32
Resolving Unnecessary Duplicate Fields 33
Identification Is the Key 39
Establishing Solid Relationships 42
Establishing a Deletion Rule 44
Setting the Type of Participation 46
Setting the Degree of Participation 48
Is That All? 50
Summary 51

CHAPTER 3 A Concise History of SQL 53

Topics Covered in This Chapter 53
The Origins of SQL 54
Early Vendor Implementations 55
". . . And Then There Was a Standard" 56
Evolution of the ANSI/ISO Standard 58
Other SQL Standards 61
Commercial Implementations 64
What the Future Holds 65
Why Should You Learn SQL? 65
Summary 66

PART II SQL Basics 69

CHAPTER 4 Creating a Simple Query 71

Topics Covered in This Chapter 71
Introducing SELECT 72
The SELECT Statement 73
A Quick Aside: Data versus Information 75
Translating Your Request into SQL 77
    Expanding the Field of Vision 81
    Using a Shortcut to Request All Columns 83
Eliminating Duplicate Rows 84
Sorting Information 87
    First Things First: Collating Sequences 88
    Let's Now Come to Order 89
Saving Your Work 92
Sample Statements 93
Summary 102
Problems for You to Solve 103

CHAPTER 5 Getting More Than Simple Columns 105

Topics Covered in This Chapter 105
What Is an Expression? 106
What Type of Data Are You Trying to Express? 107
Changing Data Types: The CAST Function 110
Specifying Explicit Values 112
    Character String Literals 112
    Numeric Literals 114
    Datetime Literals 115
Types of Expressions 117
    Concatenation 117
    Mathematical Expressions 121
    Date and Time Arithmetic 124
Using Expressions in a SELECT Clause 128
    Working with a Concatenation Expression 128
    Naming the Expression 129
    Working with a Mathematical Expression 131

Working with a Date Expression 132
A Brief Digression: Value Expressions 133
That "Nothing" Value: Null 135
Introducing Null 136
The Problem with Nulls 138
Sample Statements 139
Summary 147
Problems for You to Solve 149

CHAPTER 6 Filtering Your Data 151

Topics Covered in This Chapter 151
Refining What You See Using WHERE 151
The WHERE Clause 152
Using a WHERE Clause 154
Defining Search Conditions 156
Comparison 156
Range 164
Set Membership 167
Pattern Match 169
Null 173
Excluding Rows with NOT 175
Using Multiple Conditions 178
Introducing AND and OR 179
Excluding Rows: Take Two 184
Order of Precedence 187
Checking for Overlapping Ranges 191
Nulls Revisited: A Cautionary Note 193
Expressing Conditions in Different Ways 197
Sample Statements 198
Summary 206
Problems for You to Solve 207

PART III Working with Multiple Tables 211

CHAPTER 7 Thinking in Sets 213

Topics Covered in This Chapter 213
What Is a Set, Anyway? 214

Operations on Sets 215
Intersection 216
Intersection in Set Theory 216
Intersection between Result Sets 217
Problems You Can Solve with an Intersection 221
Difference 222
Difference in Set Theory 222
Difference between Result Sets 224
Problems You Can Solve with Difference 227
Union 228
Union in Set Theory 228
Combining Result Sets Using a Union 230
Problems You Can Solve with Union 232
SQL Set Operations 233
Classic Set Operations versus SQL 233
Finding Common Values: INTERSECT 234
Finding Missing Values: EXCEPT (Difference) 236
Combining Sets: UNION 239
Summary 242

CHAPTER 8 INNER JOINs 243

Topics Covered in This Chapter 243
What Is a JOIN? 243
The INNER JOIN 244
What's "Legal" to JOIN? 244
Column References 245
Syntax 246
Check Those Relationships! 261
Uses for INNER JOINs 262
Find Related Rows 262
Find Matching Values 263
Sample Statements 263
Two Tables 264
More Than Two Tables 270
Looking for Matching Values 277
Summary 288
Problems for You to Solve 289

CHAPTER 9 OUTER JOINs 293

Topics Covered in This Chapter 293
What Is an OUTER JOIN? 293
The LEFT/RIGHT OUTER JOIN 295
Syntax 296
The FULL OUTER JOIN 314
Syntax 314
FULL OUTER JOIN on Non-Key Values 317
UNION JOIN 317
Uses for OUTER JOINs 318
Find Missing Values 318
Find Partially Matched Information 319
Sample Statements 319
Summary 335
Problems for You to Solve 335

CHAPTER 10 UNIONs 339

Topics Covered in This Chapter 339
What Is a UNION? 339
Writing Requests with UNION 342
Using Simple SELECT Statements 342
Combining Complex SELECT Statements 345
Using UNION More Than Once 349
Sorting a UNION 351
Uses for UNION 352
Sample Statements 353
Summary 365
Problems for You to Solve 366

CHAPTER 11 Subqueries 369

Topics Covered in This Chapter 369
What Is a Subquery? 370
Row Subqueries 370
Table Subqueries 371
Scalar Subqueries 372

Subqueries as Column Expressions 372
Syntax 372
An Introduction to Aggregate Functions: COUNT and MAX 375
Subqueries as Filters 377
Syntax 378
Special Predicate Keywords for Subqueries 379
Uses for Subqueries 392
Build Subqueries as Column Expressions 392
Use Subqueries as Filters 393
Sample Statements 394
Subqueries in Expressions 395
Subqueries in Filters 400
Summary 409
Problems for You to Solve 410

PART IV Summarizing and Grouping Data 413

CHAPTER 12 Simple Totals 415

Topics Covered in This Chapter 415
Aggregate Functions 416
Counting Rows and Values with COUNT 418
Computing a Total with SUM 421
Calculating a Mean Value with AVG 423
Finding the Largest Value with MAX 424
Finding the Smallest Value with MIN 426
Using More Than One Function 427
Using Aggregate Functions in Filters 428
Sample Statements 431
Summary 438
Problems for You to Solve 439

CHAPTER 13 Grouping Data 441

Topics Covered in This Chapter 441
Why Group Data? 442
The GROUP BY Clause 444
Syntax 445
Mixing Columns and Expressions 450

Using GROUP BY in a Subquery in a WHERE Clause 452
Simulating a SELECT DISTINCT Statement 453
"Some Restrictions Apply" 454
Column Restrictions 455
Grouping on Expressions 457
Uses for GROUP BY 458
Sample Statements 459
Summary 470
Problems for You to Solve 471

CHAPTER 14 Filtering Grouped Data 473

Topics Covered in This Chapter 473
A New Meaning of "Focus Groups" 474
When You Filter Makes a Difference 478
Should You Filter in WHERE or in HAVING? 478
Avoiding the HAVING COUNT Trap 481
Uses for HAVING 486
Sample Statements 487
Summary 496
Problems for You to Solve 496

PART V Modifying Sets of Data 499

CHAPTER 15 Updating Sets of Data 501

Topics Covered in This Chapter 501
What Is an UPDATE? 501
The UPDATE Statement 502
Using a Simple UPDATE Expression 503
A Brief Aside: Transactions 506
Updating Multiple Columns 507
Using a Subquery to Filter Rows 508
Using a Subquery UPDATE Expression 514
Uses for UPDATE 516
Sample Statements 517
Summary 533
Problems for You to Solve 534

CHAPTER 16 Inserting Sets of Data 537

Topics Covered in This Chapter 537
What Is an INSERT? 537
The INSERT Statement 539
Inserting Values 539
Generating the Next Primary Key Value 542
Inserting Data by Using SELECT 544
Uses for INSERT 550
Sample Statements 552
Summary 562
Problems for You to Solve 563

CHAPTER 17 Deleting Sets of Data 567

Topics Covered in This Chapter 567
What Is a DELETE? 567
The DELETE Statement 568
Deleting All Rows 569
Deleting Some Rows 571
Uses for DELETE 575
Sample Statements 576
Summary 583
Problems for You to Solve 584

In Closing 587

APPENDICES 589

A SQL Standard Diagrams 591

B Schema for the Sample Databases 601

C Date and Time Functions 607

D Suggested Reading 615

Index 617

This page intentionally left blank

Foreword

In the 20 years since the database language SQL was adopted as an international standard, and the 25 years since SQL database products appeared on the market, SQL has become the predominant language for storing, modifying, retrieving, and deleting data. Today, a significant portion of the world's data—and the world's economy—is tracked using SQL databases.

SQL is everywhere because it is a very powerful tool for manipulating data. It is in high-performance transaction processing systems. It is behind Web interfaces. I've even found SQL in network monitoring tools and spam firewalls.

Today, SQL can be executed directly, embedded in programming languages, and accessed through call interfaces. It is hidden inside GUI development tools, code generators, and report writers. However visible or hidden, the underlying queries are SQL. Therefore, to understand existing applications and to create new ones, you need to understand SQL.

SQL Queries for Mere Mortals, Second Edition, provides a step-by-step, easy-to-read introduction to writing SQL queries. It includes hundreds of examples with detailed explanations. This book provides the tools you need to understand, modify, and create SQL queries.

As a database consultant and a participant in both the U.S. and international SQL standards committees, I spend a lot of time working with SQL. So, it is with a certain amount of authority that I state, "The authors of this book not only understand SQL, they also understand how to explain it." Both qualities make this book a valuable resource.

Keith W. Hare
Senior Consultant, JCC Consulting, Inc.
Vice Chair, INCITS H2—the USA SQL Standards Committee
Convenor, ISO/IEC JTC1 SC32 WG3—the International
SQL Standards Committee

This page intentionally left blank

Preface

"Language is by its very nature a communal thing;
that is, it expresses never the exact thing but a
compromise—that which is common to you, me, and everybody."
—Thomas Earnest Hulme, *Speculations*

Learning how to retrieve information from or manipulate information in a database is commonly a perplexing exercise. However, it can be a relatively easy task as long as you understand the question you're asking or the change you're trying to make to the database. After you understand the problem, you can translate it into the language used by any database system, which in most cases is Structured Query Language (SQL). You have to translate your request into an SQL statement so that your database system knows what information you want to retrieve or change. SQL provides the means for you and your database system to communicate.

Throughout our many years as database consultants, we've found that the number of people who merely need to retrieve information from a database or perform simple data modifications in a database far outnumber those who are charged with the task of creating programs and applications for a database. Unfortunately, no books focus solely on this subject, particularly from a "mere mortals" viewpoint. There are numerous good books on SQL, to be sure, but most are targeted to database programming and development.

With this in mind, we decided it was time to write a book that would help people learn how to query a database properly and effectively. We produced the first edition of this book in 2000. With this new edition, we also wanted to introduce you to the basic ways to change data in your database using SQL. The result of our decision is in your hands. This book is unique among SQL books in that it focuses on SQL with little regard to any one specific database system implementation. This second edition includes hundreds of new examples, and we included versions of the sample databases using the popular open-source MySQL database system. When you finish reading this book, you'll have the skills you need to retrieve or modify any information you require.

Acknowledgments

Writing a book such as this is always a cooperative effort. There are always editors, colleagues, friends, and relatives willing to lend their support and provide valuable advice when we need it the most. These people continually provide us with encouragement, help us to remain focused, and motivate us to see this project through to the end.

First and foremost, we want to thank our acquisitions editor, Elizabeth Peterson, for prodding us to produce this second edition. Thanks also to Kristin Weinberger for shepherding us along the way. And we can't forget our final acquisitions editor, Chuck Toporek, as well as Romny French and the production staff—they're a great team! Special thanks to Chrysta Meadowbrooke, who did a fabulous job copyediting the final manuscript. She cleaned up lots of inconsistencies and even pointed out some SQL examples that needed fixing! Finally, thanks to editor-in-chief Karen Gettman, who put this team together and kept a watchful eye over the entire process.

Next, we'd like to acknowledge our technical editors, particularly Stephen Forte and Keith Hare. Keith especially spent time working through all the examples, pointing out a few errors, and making suggestions to enhance the text. Thanks once again to all of you for your time and input and for helping us to make this a solid treatise on SQL queries.

Finally, another very special thanks to Keith Hare for providing the Foreword. As the Convenor of the International SQL Standards Committee, Keith is an SQL expert par excellence. We have a lot of respect for Keith's knowledge and expertise on the subject, and we're pleased to have his thoughts and comments at the beginning of our book.

About the Authors

John L. Viescas is an independent database consultant with more than 40 years of experience. He began his career as a systems analyst, designing large database applications for IBM mainframe systems. He spent six years at Applied Data Research in Dallas, Texas, where he directed a staff of more than 30 people and was responsible for research, product development, and customer support of database products for IBM mainframe computers. While working at Applied Data Research, John completed a degree in business finance at the University of Texas at Dallas, graduating cum laude.

John joined Tandem Computers, Inc., in 1988, where he was responsible for the development and implementation of database marketing programs in TandemÕs U.S. Western Sales region. He developed and delivered technical seminars on TandemÕs relational database management system, NonStop SQL. John wrote his first book, A Quick Reference Guide to SQL (Microsoft Press, 1989), as a research project to document the similarities in the syntax among the ANSI-86 SQL standard, IBMÕs DB2, MicrosoftÕs SQL Server, Oracle CorporationÕs Oracle, and TandemÕs NonStop SQL. He wrote the first edition of Running Microsoft Access (Microsoft Press, 1992) while on sabbatical from Tandem. He has since written four editions of Running, two editions of Microsoft Office Access Inside Out (Microsoft Press, 2003 and 2007Ñthe successor to the Running series), and Building Microsoft Access Applications (Microsoft Press, 2005).

John formed his own company in 1993. He provides information systems management consulting for a variety of small to large businesses around the world, with a specialty in the Microsoft Access and SQL Server database management products. He maintains offices in Nashua, New Hampshire, and Paris, France. He has been recognized as a ÒMost Valuable ProfessionalÓ (MVP) since 1993 by Microsoft Product Support Services for his assistance with technical questions on public support forums. He set a landmark 20 consecutive years as an MVP in 2013.

You can visit JohnÕs Web site at www.viescas.com or contact him by e-mail at john@viescas.com.

Michael J. Hernandez has been an independent relational database consultant specializing relational database design. He has more than 20 years of experience in the technology industry, developing database application for a wide variety of clients. HeÕs been a contributing author to wide variety of magazine columns, white papers, books and periodicals, and is coauthor of the best-sellingSQL Queries for Mere Mortals. Mike has been a top-rated and noted technical trainer for the government, the military, the private sector and companies throughout the United States. He has spoken at numerous national and international conferences, and has consistently been a top-rated speaker and presenter.

Aside from his technical background, Mike has a diverse set of skills and interests that he also pursues, ranging from the artistic to the metaphysical. His greatest interest is still the guitar, as heÕs been a practicing guitarist for more than 40 years and once played professionally for 15 years. HeÕs also a working actor, a great cook, loves to teach (writing, public speaking, music), has a gift for bad puns, and even reads Tarot cards.

He says heÕs never going to retire, per se, but rather just change whatever it is heÕs doing whenever he finally gets tired of it and move on to something else that interests him.

Introduction

"I presume you're mortal, and may err."
—James Shirley
The Lady of Pleasure

If you've used a computer more than casually, you have probably used Structured Query Language, or SQL—perhaps without even knowing it. SQL is *the* standard language for communicating with most database systems. Any time you import data into a spreadsheet or perform a merge into a word processing program, you're most likely using SQL in some form or another. Every time you go online to an e-commerce site on the Web and place an order for a book, a recording, a movie, or any of the dozens of other products you can order, there's a very high probability that the code behind the Web page you're using is accessing its databases with SQL. If you need to get information from a database system that uses SQL, you can enhance your understanding of the language by reading this book.

Are You a Mere Mortal?

You might ask, "Who is a *mere mortal*? Me?" The answer is not simple. When we started to write this book, we thought we were experts in the database language called SQL. Along the way, we discovered we were mere mortals too, in several areas. We understood a few specific implementations of SQL very well, but we unraveled many of the complex intricacies of the language as we studied how it is used in many commercial products. So, if you fit any of the following descriptions, you're a mere mortal too!

- If you use computer applications that let you access information from a database system, you're probably a mere mortal. The first time you don't get the information you expected using the query tools built in to your application, you'll need to explore the underlying SQL statements to find out why.

- If you have recently discovered one of the many available desktop database applications but are struggling with defining and querying the data you need, you're a mere mortal.
- If you're a database programmer who needs to "think out of the box" to solve some complex problems, you're a mere mortal.
- If you're a database guru in one product but are now faced with integrating the data from your existing system into another system that supports SQL, you're a mere mortal.

In short, *anyone* who has to use a database system that supports SQL can use this book. As a beginning database user who has just discovered that the data you need can be fetched using SQL, you will find that this book teaches you all the basics and more. For an expert user who is suddenly faced with solving complex problems or integrating multiple systems that support SQL, this book will provide insights into leveraging the complex abilities of the SQL database language.

About This Book

Everything you read in this book is based on the current International Organization for Standardization (ISO) Standard for the SQL database language (document ISO/IEC 9075-2:2003), as currently implemented in most of the popular commercial database systems. The ISO document was also adopted by the American National Standards Institute (ANSI), so this is truly an international standard. The SQL you'll learn here *is not* specific to any particular software product.

As you'll learn in more detail in Chapter 3, A Concise History of SQL, the SQL Standard defines both more and less than you'll find implemented in most commercial database products. Most database vendors have yet to implement many of the more advanced features, but most do support the core of the standard.

We researched a wide range of popular products to make sure that you can use what we're teaching in this book. When we found parts of the core of the language not supported by some major products, we warned you in the text and showed you alternate ways to state your database requests in standard SQL. When we found significant parts of the SQL Standard supported by only a few vendors, we introduced you to the syntax and then suggested alternatives.

We have organized this book into five major sections.

- Part I, Relational Databases and SQL, explains how modern database systems are based on a rigorous mathematical model and provides a brief history of the database query language that has evolved into what we know as SQL. We also discuss some simple rules that you can use to make sure your database design is sound.
- Part II, SQL Basics, introduces you to using the SELECT statement, creating expressions, and sorting information with an ORDER BY clause. You'll also learn how to filter data by using a WHERE clause.
- Part III, Working with Multiple Tables, shows you how to formulate queries that draw data from more than one table. Here we show you how to link tables in a query using the INNER JOIN, OUTER JOIN, and UNION operators, and how to work with subqueries.
- Part IV, Summarizing and Grouping Data, discusses how to obtain summary information and group and filter summarized data. Here is where you'll learn about the GROUP BY and HAVING clauses.
- Part V, Modifying Sets of Data, explains how to write queries that modify a set of rows in your tables. In the chapters in this section, you'll learn how to use the UPDATE, INSERT, and DELETE statements.

At the end of the book in the appendices, you'll find syntax diagrams for all the SQL elements you've learned, layouts of the sample databases, a list of date and time manipulation functions implemented in five of the major database systems, and book recommendations to further your study of SQL. There is also a CD containing all the sample databases used throughout the book in several different formats.

What This Book Is Not

Although this book is based on the 2003 SQL Standard that was current at the time of this writing (a 2007/2008 draft standard is in the works), it does not cover every aspect of the standard. In truth, many features in the 2003 SQL Standard won't be implemented for many years—if at all—in the major database system implementations. The fundamental purpose of this book is to give you a solid grounding in writing queries in SQL. Throughout the book, you'll find us recommending that you "consult your database documentation" for how a specific feature might or might not work. That's not to say we covered only the lowest common denominator for any feature among the major database systems. We do try to caution you when some systems implement a feature differently or not at all.

You'll find it difficult to create other than simple queries using a single table if your database design is flawed. We included a chapter on database design to help you identify when you will have problems, but that one chapter includes only the basic principles. A thorough discussion of database design principles and how to implement a design in a specific database system is beyond the scope of this book.

This book is also not about how to solve a problem in the most efficient way. As you work through many of the later chapters, you'll find we suggest more than one way to solve a particular problem. In some cases where writing a query in a particular way is likely to have performance problems on any system, we try to warn you about it. But each database system has its own strengths and weaknesses. After you learn the basics, you'll be ready to move on to digging into the particular database system you use to learn how to formulate your query solutions so that they run in a more optimal manner.

How to Use This Book

We have designed the chapters in this book to be read in sequence. Each succeeding chapter builds on concepts taught in earlier chapters. However, you can jump into the middle of the book without getting lost. For example, if you are already familiar with the basic clauses in a SELECT statement and want to learn more about JOINs, you can jump right in to Chapters 7 Thinking in Sets, 8 INNER JOINS, and 9 OUTER JOINS.

At the end of many of the chapters you'll find an extensive set of sample problems, their solutions, and sample result sets. We recommend that you study several of the samples to gain a better understanding of the techniques involved and then try solving some of the later samples yourself without looking at the solutions we propose.

Note that where a particular query returns dozens of rows in the result set, we show you only the first few rows in this book to give you an idea of how the answer should look. You might not see the exact same result on your system, however, because each database system that supports SQL has its own optimizer that figures out the fastest way to solve the query. Also, the first few rows you see returned by your database system might not exactly match the first few we show you unless the query contains an ORDER BY clause that requires the rows to be returned in a specific sequence.

We've also included a complete set of problems for you to solve on your own, which you'll find at the end of most chapters. This gives you the opportunity to really practice what you've just learned in the chapter. Don't worry—the solutions are included in the sample databases on the CD. We've also included hints on those problems that might be a little tricky.

After you have worked your way through the entire book, you'll find the complete SQL diagrams in Appendix A to be an invaluable reference for all the SQL techniques we showed you. You will also be able to use the sample database layouts in Appendix B to help you design your own databases.

Reading the Diagrams Used in This Book

The numerous diagrams throughout the book illustrate the proper syntax for the statements, terms, and phrases you'll use when you work with SQL. Each diagram provides a clear picture of the overall construction of the SQL element currently being discussed. You can also use any of these diagrams as templates to create your own SQL statements or to help you acquire a clearer understanding of a specific example.

All the diagrams are built from a set of core elements and can be divided into two categories: *statements* and *defined terms*. A statement is always a major SQL operation, such as the SELECT statement we discuss in this book, while a defined term is always a component used to build part of a statement, such as a *value expression, a search condition,* or a *conditional expression.* (Don't worry—we'll explain all these terms later in the book.) The only difference between a syntax diagram for a statement and a syntax diagram for a defined term is the manner in which the main syntax line begins and ends. We designed the diagrams with these differences so that you can clearly see whether you're looking at the diagram for an entire statement or a diagram for a term that you might use within a statement. Figure 1 (on page xxviii) shows the beginning and end points for both diagram categories. Aside from this difference, the diagrams are built from the same elements. Figure 2 (on page xxviii) shows an example of each type of syntax diagram and is followed by a brief explanation of each diagram element.

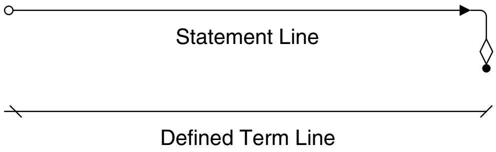

Figure 1 *Syntax line end points for statements and defined terms*

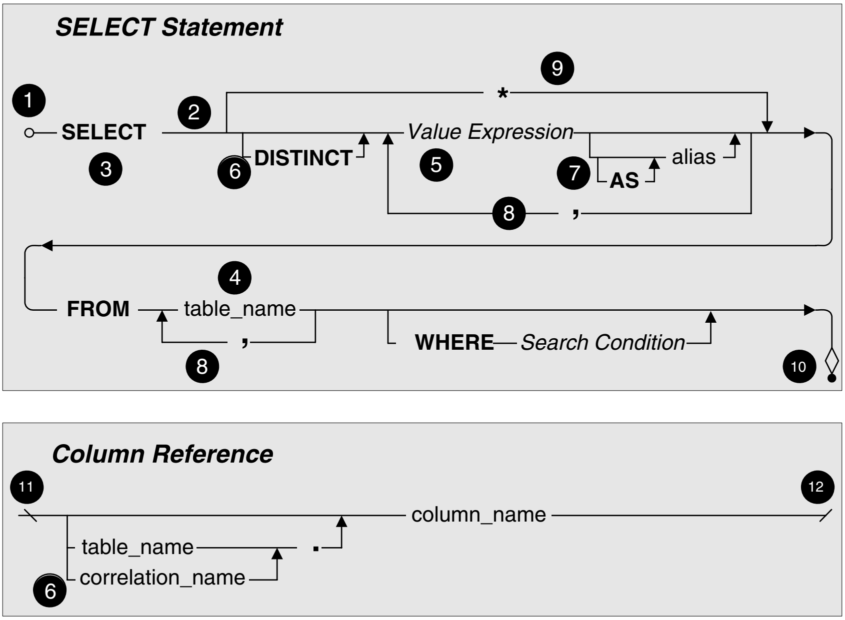

Figure 2 *Sample statement and defined term diagrams*

1. ***Statement start point***—denotes the beginning of the main syntax line for a statement. Any element that appears *directly on* the main syntax line is a *required element*, and any element that appears *below* it is an *optional element*.
2. ***Main syntax line***—determines the order of all required and optional elements for the statement or defined term. Follow this line from left to right (or in the direction of the arrows) to build the syntax for the statement or defined term.
3. ***Keyword(s)***—indicates a major word in SQL grammar that is a required part of the syntax for a statement or defined term. In a diagram, keywords are formatted in capital letters and bold font. (You don't have to worry about typing a keyword in capital letters when you actually write the statement in your database program, but it does make the statement easier to read.)

4. ***Literal entry***—specifies the name of a value you explicitly supply to the statement. A literal entry is represented by a word or phrase that indicates the type of value you need to supply. Literal entries in a diagram are formatted in all lowercase letters.
5. ***Defined term***—denotes a word or phrase that represents some operation that returns a final value to be used in this statement. We'll explain and diagram every defined term you need to know as you work through the book. Defined terms are always formatted in italic letters.
6. ***Optional element***—indicates any element or group of elements that appears below the main syntax line. An optional element can be a statement, keyword, defined term, or literal value and, for purposes of clarity, is placed on its own line. In some cases, you can specify a set of values for a given option, with each value separated by a comma (see number 8). Also, several optional elements have a set of sub-optional elements (see number 7). In general, you read the syntax line for an optional element from left to right, in the same manner that you read the main syntax line. Always follow the directional arrows and you'll be in good shape. Note that some options allow you to specify multiple values or choices, so the arrow will flow from right to left. After you've entered all the items you need, however, the flow will return to normal from left to right. Fortunately, all optional elements work the same way. After we show you how to use an optional element later in the book, you'll know how to use any other optional element you encounter in a syntax diagram.
7. ***Sub-optional element***—denotes any element or group of elements that appears below an optional element. Sub-optional elements allow you to fine-tune your statements so that you can work with more complex problems.
8. ***Option list separator***—indicates that you can specify more than one value for this option and that each value must be separated with a comma.
9. ***Alternate option***—denotes a keyword or defined term that can be used as an alternative to one or more optional elements. The syntax line for an alternate option will bypass the syntax lines of the optional elements it is meant to replace.
10. ***Statement end point***—denotes the end of the main syntax line for a statement.
11. ***Defined term start point***—denotes the beginning of the main syntax line for a defined term.
12. ***Defined term end point***—denotes the end of the main syntax line for a defined term.

Now that you're familiar with these elements, you'll be able to read all the syntax diagrams in the book. And on those occasions when a diagram requires further explanation, we provide you with the information you need to read the diagram clearly and easily. To help you better understand how the diagrams work, here's a sample SELECT statement that we built using Figure 2.

```
SELECT FirstName, LastName, City, DOB AS DateOfBirth
FROM Students
WHERE City = 'El Paso'
```

This SELECT statement retrieves four columns from the Students table, as we've indicated in the SELECT and FROM clauses. As you follow the main syntax line from left to right, you see that you have to indicate at least one *value expression*. A value expression can be a column name, an expression created using column names, or simply a constant (literal) value that you want to display. You can indicate as many columns as you need with the value expression's *option list separator* (a comma). This is how we were able to use four column names from the Student table. We were concerned that some people viewing the information returned by this SELECT statement might not know what DOB means, so we assigned an *alias* to the DOB column with the value expression's AS sub-option. Finally, we used the WHERE clause to make certain the SELECT statement shows only those students who live in El Paso. (If this doesn't quite make sense to you just now, there's no cause for alarm. You'll learn all this in great detail throughout the remainder of the book.)

You'll find a full set of syntax diagrams in Appendix A. They show the complete and proper syntax for all the statements and defined terms we discuss in the book. If you happen to refer to these diagrams as you work through each chapter, you'll notice a slight disparity between some of the diagrams in a given chapter and the corresponding diagrams in the appendix. The diagrams in the chapters are just simplified versions of the diagrams in the appendix. These simplified versions allow us to explain complex statements and defined terms more easily and give us the ability to focus on particular elements as needed. But don't worry—all the diagrams in the appendix will make perfect sense after you work through the material in the book.

Sample Databases Used in This Book

Bound into the back of the book, you'll find a CD-ROM containing five sample databases that we use for the example queries throughout the book. We've also included diagrams of the database structures in Appendix B: Schema for the Sample Databases.

1. Sales Orders. This is a typical order entry database for a store that sells bicycles and accessories. (Every database book needs at least one order entry example, right?)
2. Entertainment Agency. We structured this database to manage entertainers, agents, customers, and bookings. You would use a similar design to handle event bookings or hotel reservations.
3. School Scheduling. You might use this database design to register students at a high school or community college. This database tracks not only class registrations but also which instructors are assigned to each class and what grades the students received.
4. Bowling League. This database tracks bowling teams, team members, the matches they played, and the results.
5. Recipes. You can use this database to save and manage all your favorite recipes. We even added a few that you might want to try.

On the sample CD or ftp site (ftp://ftp.viescas.com/Download/SQLQFMM/SQLQFMM2.zip), you can find all five databases in four different formats.

- Because of the great popularity of the Microsoft Office Access desktop database, we created one set of databases (.mdb file extension) using Microsoft Access 2000 (Version 9.0). We chose Version 9 of this product because it closely supports the current ISO/IEC SQL Standard, and you can open database files in this format using Access 2000, 2002 (XP), 2003, and 2007. You can find these files in the MSAccess subfolder.
- The second format consists of database files (.mdf file extension) created using Microsoft SQL Server 2000. We have also included SQL command files (.sql file extension) and batch files (.bat file extension) that you can use to attach the samples to a Microsoft SQL Server catalog. You can also attach these files to a Microsoft SQL Server 2005 server. You can find these files in the MSSQLServer subfolder. You can obtain a free copy of Microsoft SQL Server 2005 Express Edition at http://msdn.microsoft.com/vstudio/express/sql/download/default.aspx.
- We created the third set of databases using the popular open-source MySQL version 5 database system. You can either point your InnoDB data directory to the MySQL subfolder or use the scripts (.sql file extension) you can also find in that folder to create the database structure, load the data, and create the sample views in your own MySQL data folder. You can obtain a free copy of the community edition of the MySQL database system at http://www.mysql.com/.

- The fourth format is a series of SQL scripts that you can modify and use with any major database system that supports SQL. You can find scripts to define the schema (the tables) of each database, to load the data using INSERT statements, and to create the queries using CREATE VIEW statements in the SQLScripts subfolder. Although we created these scripts using utilities in Microsoft SQL Server, we simplified them to make them generic for use with most database systems.

To install the sample files, see the file ReadMe.txt in the root folder of the sample CD. If you mount the sample CD on an Apple Macintosh system, you will find only the sample files for MySQL and the SQL scripts.

> ❖ **Note** Although we were very careful to use the most common and simplest syntax for the CREATE TABLE, CREATE INDEX, CREATE CONSTRAINT, and INSERT commands in the sample SQL scripts, you (or your database administrator) might need to modify these files slightly to work with your database system. If you're working with a database system on a remote server, you might need to gain permission from your database administrator to build the samples from the SQL commands we supplied.

For the chapters in Parts II, III, and IV that focus on the SELECT statement, you'll find all the example statements and solutions in the "example" version of each sample database (e.g., SalesOrdersExample, EntertainmentAgency Example). Because the examples in Part V modify the sample data, we created "modify" versions of each of the sample databases (e.g., SalesOrdersModify, EntertainmentAgencyModify). The sample databases for Part V also include additional columns and tables not found in the SELECT examples that enable us to demonstrate certain features of UPDATE, INSERT, and DELETE queries.

"Follow the Yellow Brick Road"

—Munchkin to Dorothy in *The Wizard of Oz*

Now that you've read through the Introduction, you're ready to start learning SQL, right? Well, maybe. At this point, you're still in the house, it's still being tossed about by the tornado, and you haven't left Kansas.

Before you make that jump to Chapter 4, Creating a Sample Query, take our advice and read through the first three chapters. Chapter 1, What Is Relational?, will give you an idea of how the relational database was conceived and how it has grown to be the most widely used type of database in the industry today. We hope this will give you some amount of insight into the database system you're currently using. In Chapter 2, Ensuring Your Database Structure Is Sound, you'll learn how to fine-tune your data structures so that your data is reliable and, above all, accurate. You're going to have a tough time working with some of the SQL statements if you have poorly designed data structures, so we suggest you read this chapter carefully.

Chapter 3 is literally the beginning of the "yellow brick road." Here you'll learn the origins of SQL and how it evolved into its current form. You'll also learn about some of the people and companies who helped pioneer the language and why there are so many varieties of SQL. Finally, you'll learn how SQL came to be a national and international standard and what the outlook for SQL will be in the years to come.

After you've read these chapters, consider yourself well on your way to Oz. Just follow the road we've laid out through each of the remaining chapters. When you've finished the book, you'll find that you've found the wizard—and he is you.

This page intentionally left blank

7

Thinking in Sets

"Small cheer and a great welcome makes a merry feast."
—William Shakespeare
Comedy of Errors, Act 3, scene 1

Topics Covered in This Chapter

What Is a Set, Anyway?

Operations on Sets

Intersection

Difference

Union

SQL Set Operations

Summary

By now, you know how to create a set of information by asking for specific columns or expressions on columns (SELECT), how to sort the rows (ORDER BY), and how to restrict the rows returned (WHERE). Up to this point, we've been focusing on basic exercises involving a single table. But what if you want to know something about information contained in multiple tables? What if you want to compare or contrast sets of information from the same or different tables?

Creating a meal by peeling, slicing, and dicing a single pile of potatoes or a single bunch of carrots is easy. From here on out, most of the problems we're going to show you how to solve will involve getting data from *multiple* tables. We're not only going to show you how to put together a good stew—we're going to teach you how to be a chef!

Before digging into this chapter, you need to know that it's all about the *concepts* you must understand in order to successfully link two or more sets of information. We're also going to give you a brief overview of some specific syntax defined in the SQL Standard that directly supports the pure definition of these concepts. Be forewarned, however, that many current commercial implementations of SQL do not yet support this "pure" syntax. In later chapters, we'll show you how to implement the concepts you'll learn here using SQL syntax that is commonly supported by most major database systems. What we're after here is not the letter of the law but rather the spirit of the law.

What Is a Set, Anyway?

If you were a teenager any time from the mid-1960s onward, you might have studied set theory in a mathematics course. (Remember New Math?) If you were introduced to set algebra, you probably wondered why any of it would ever be useful.

Now you're trying to learn about relational databases and this quirky language called SQL to build applications, solve problems, or just get answers to your questions. Were you paying attention in algebra class? If so, solving problems—particularly complex ones—in SQL will be much easier.

Actually, you've been working with sets from the beginning of this book. In Chapter 1, What Is Relational?, you learned about the basic structure of a relational database—tables containing records that are made up of one or more fields. (Remember that in SQL, records are known as rows, and fields are known as columns.) Each table in your database is a *set* of information about one subject. In Chapter 2, Ensuring Your Database Structure Is Sound, you learned how to verify that the structure of your database is sound. Each table should contain the *set* of information related to one and only one subject or action.

In Chapter 4, Creating a Simple Query, you learned how to build a basic SELECT statement in SQL to retrieve a result *set* of information that contains specific columns from a single table and how to sort those result sets. In Chapter 5, Getting More Than Simple Columns, you learned how to glean a new *set* of information from a table by writing expressions that operate on one or more columns. In Chapter 6, Filtering Your Data, you learned how to restrict further the *set* of information you retrieve from your tables by adding a filter (WHERE clause) to your query.

As you can see, a set can be as little as the data from one column from one row in one table. Actually, you can construct a request in SQL that returns no rows—an empty set. Sometimes it's useful to discover that something does *not* exist. A set can also be multiple columns (including columns you create with expressions) from multiple rows fetched from multiple tables. Each row in a result set is a *member* of the set. The values in the columns are specific *attributes* of each member—data items that describe the member of the set. In the next several chapters, we'll show how to ask for information from multiple sets of data and link these sets together to get answers to more complex questions. First, however, you need to understand more about sets and the logical ways to combine them.

Operations on Sets

In Chapter 1, we discussed how Dr. E. F. Codd invented the relational model on which most modern databases and SQL are based. Two branches of mathematics—set theory and first-order predicate logic—formed the foundation of his new model.

After you graduate beyond getting answers from only a single table, you need to learn how to use result sets of information to solve more complex problems. These complex problems usually require using one of the common set operations to link data from two or more tables. Sometimes, you'll need to get two different result sets from the same table and then combine them to get your answer.

The three most common set operations are as follows.

- **Intersection**—You use this to find the common elements in two or more different sets: "Show me the recipes that contain *both* lamb *and* rice." "Show me the customers who ordered *both* bicycles *and* helmets."
- **Difference**—You use this to find items that are in one set but not another: "Show me the recipes that contain lamb but *do not* contain rice." "Show me the customers who ordered a bicycle but *not* a helmet."
- **Union**—You use this to combine two or more similar sets: "Show me all the recipes that contain *either* lamb *or* rice." "Show me the customers who ordered *either* a bicycle *or* a helmet."

In the following three sections, we'll explain these basic set operations—the ones you should have learned in high school algebra. The SQL Set Operations section later in this chapter gives an overview of how these operations are implemented in "pure" SQL.

Intersection

No, it's not your local street corner. An *intersection* of two sets contains the common elements of two sets. Let's first take a look at an intersection from the pure perspective of set theory and then see how you can use an intersection to solve business problems.

Intersection in Set Theory

An intersection is a very powerful mathematical tool often used by scientists and engineers. As a scientist, you might be interested in finding common points between two sets of chemical or physical sample data. For example, a pharmaceutical research chemist might have two compounds that seem to provide a certain beneficial effect. Finding the commonality (the intersection) between the two compounds might help discover what it is that makes the two compounds effective. Or, an engineer might be interested in finding the intersection between one alloy that is hard but brittle and another alloy that is soft but resilient.

Let's take a look at intersection in action by examining two sets of numbers. In this example, each single number is a member of the set. The first set of numbers is as follows.

1, 5, 8, 9, 32, 55, 78

The second set of numbers is as follows.

3, 7, 8, 22, 55, 71, 99

The intersection of these two sets of numbers is the numbers common to both sets.

8, 55

The individual entries—the members—of each set don't have to be just single values. In fact, when solving problems with SQL, you'll probably deal with sets of rows.

According to set theory, when a member of a set is something more than a single number or value, each member (or object) of the set has multiple attributes or bits of data that describe the properties of each member. For example, your favorite stew recipe is a complex member of the set of all recipes that contains many different ingredients. Each ingredient is an attribute of your complex stew member.

To find the intersection between two sets of complex set members, you have to find the members that match on all the attributes. Also, all the members in each set you're trying to compare must have the same number and type of attributes. For example, suppose you have a complex set like the one below, in which each row represents a member of the set (a stew recipe), and each column denotes a particular attribute (an ingredient).

Potatoes	Water	Lamb	Peas
Rice	Chicken Stock	Chicken	Carrots
Pasta	Water	Tofu	Snap Peas
Potatoes	Beef Stock	Beef	Cabbage
Pasta	Water	Pork	Onions

A second set might look like the following.

Potatoes	Water	Lamb	Onions
Rice	Chicken Stock	Turkey	Carrots
Pasta	Vegetable Stock	Tofu	Snap Peas
Potatoes	Beef Stock	Beef	Cabbage
Beans	Water	Pork	Onions

The intersection of these two sets is the one member whose attributes all match in both sets.

Potatoes	Beef Stock	Beef	Cabbage

Intersection between Result Sets

If the previous examples look like rows in a table or a result set to you, you're on the right track! When you're dealing with rows in a set of data that you fetch with SQL, the attributes are the individual columns. For example, suppose you have a set of rows returned by a query like the following one. (These are recipes from John's cookbook.)

Recipe	Starch	Stock	Meat	Vegetable
Lamb Stew	Potatoes	Water	Lamb	Peas
Chicken Stew	Rice	Chicken Stock	Chicken	Carrots
Veggie Stew	Pasta	Water	Tofu	Snap Peas
Irish Stew	Potatoes	Beef Stock	Beef	Cabbage
Pork Stew	Pasta	Water	Pork	Onions

A second query result set might look like the following. (These are recipes from Mike's cookbook.)

Recipe	Starch	Stock	Meat	Vegetable
Lamb Stew	Potatoes	Water	Lamb	Peas
Turkey Stew	Rice	Chicken Stock	Turkey	Carrots
Veggie Stew	Pasta	Vegetable Stock	Tofu	Snap Peas
Irish Stew	Potatoes	Beef Stock	Beef	Cabbage
Pork Stew	Beans	Water	Pork	Onions

The intersection of these two sets is the two members whose attributes all match in both sets—that is, the two recipes that Mike and John have in common.

Recipe	Starch	Stock	Meat	Vegetable
Lamb Stew	Potatoes	Water	Lamb	Peas
Irish Stew	Potatoes	Beef Stock	Beef	Cabbage

Sometimes it's easier to see how intersection works using a set diagram. A *set diagram* is an elegant yet simple way to diagram sets of information and graphically represent how the sets intersect or overlap. You might also have heard this sort of diagram called a Euler or Venn diagram. (By the way, Leonard Euler was an eighteenth-century Swiss mathematician, and John Venn used this particular type of logic diagram in 1880 in a paper he wrote while a Fellow at Cambridge University. So you can see that "thinking in sets" is not a particularly modern concept!)

Let's assume you have a nice database containing all your favorite recipes. You really like the way onions enhance the flavor of beef, so you're interested in finding all recipes that contain both beef and onions. Figure 7–1 shows the set diagram that helps you visualize how to solve this problem.

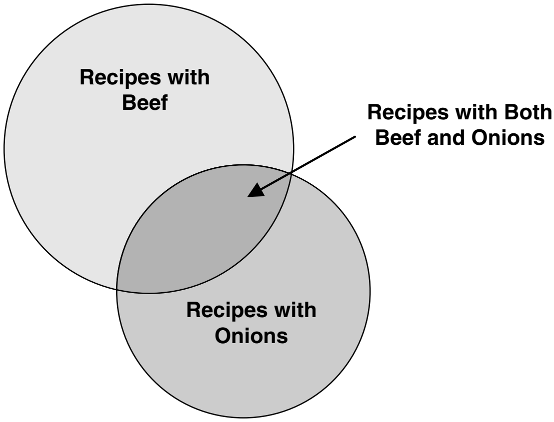

Figure 7–1 *Finding out which recipes have both beef and onions*

The upper circle represents the set of recipes that contain beef. The lower circle represents the set of recipes that contain onions. Where the two circles overlap is where you'll find the recipes that contain both—the intersection of the two sets. As you can imagine, you first ask SQL to fetch all the recipes that have beef. In the second query, you ask SQL to fetch all the recipes that have onions. As you'll see later, you can use a special SQL keyword—INTERSECT—to link the two queries to get the final answer.

Yes, we know what you're thinking. If your recipe table looks like the samples above, you could simply say the following.

"Show me the recipes that have beef as the meat ingredient and onions as the vegetable ingredient."

Translation	Select the recipe name from the recipes table where meat ingredient is beef and vegetable ingredient is onions
Clean Up	Select ~~the~~ recipe name from ~~the~~ recipes ~~table~~ where meat ingredient ~~is~~ = beef and vegetable ingredient ~~is~~ = onions
SQL	see below

```
SELECT RecipeName
FROM Recipes
WHERE MeatIngredient = 'Beef'
   AND VegetableIngredient = 'Onions'
```

Hold on now! If you remember the lessons you learned in Chapter 2, you know that a single Recipes table probably won't cut it. (Pun intended!) What about recipes that have ingredients other than meat and vegetables? What about the fact that some recipes have many ingredients and others have only a few? A correctly designed recipes database will have a separate Recipe_Ingredients table with one row per recipe per ingredient. Each ingredient row will have only one ingredient, so no single row can be both beef and onions at the same time. You'll need to first find all the beef rows, then find all the onions rows, and then intersect them on RecipeID. (If you're confused about why we're criticizing the previous table design, be sure to go back and read Chapter 2!)

How about a more complex problem? Let's say you want to add carrots to the mix. A set diagram to visualize the solution might look like Figure 7–2.

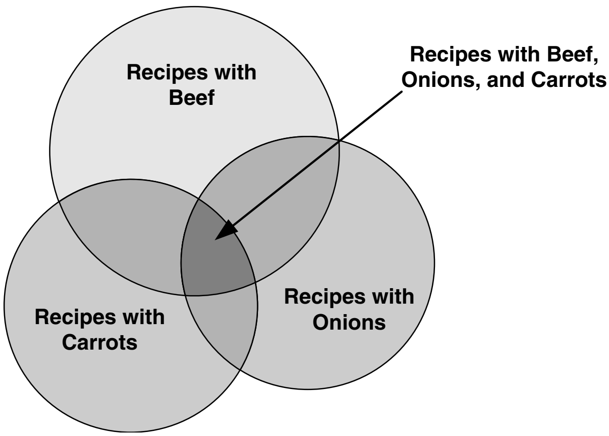

Figure 7–2 *Determining which recipes have beef, onions, and carrots*

Got the hang of it? The bottom line is that when you're faced with solving a problem involving complex criteria, a set diagram can be an invaluable way to see the solution expressed as the intersection of SQL result sets.

Problems You Can Solve with an Intersection

As you might guess, you can use an intersection to find the matches between two or more sets of information. Here's just a small sample of the problems you can solve using an intersection technique with data from the sample databases.

"Show me customers and employees who have the same name."

"Find all the customers who ordered a bicycle and also ordered a helmet."

"List the entertainers who played engagements for customers Bonnicksen and Rosales."

"Show me the students who have an average score of 85 or better in Art and who also have an average score of 85 or better in Computer Science."

"Find the bowlers who had a raw score of 155 or better at both Thunderbird Lanes and Bolero Lanes."

"Show me the recipes that have beef and garlic."

One of the limitations of using a pure intersection is that the values must match in all the columns in each result set. This works well if you're intersecting two or more sets from the same table—for example, customers who ordered bicycles and customers who ordered helmets. It also works well when you're intersecting sets from tables that have similar columns—for example, customer names and employee names. In many cases, however, you'll want to find solutions that require a match on only a few column values from each set. For this type of problem, SQL provides an operation called a JOIN—an intersection on key values. Here's a sample of problems you can solve with a JOIN.

"Show me customers and employees who live in the same city." (JOIN on the city name.)

"List customers and the entertainers they booked." (JOIN on the engagement number.)

"Find the agents and entertainers who live in the same ZIP Code." (JOIN on the ZIP Code.)

"Show me the students and their teachers who have the same first name." (JOIN on the first name.)

"Find the bowlers who are on the same team." (JOIN on the team ID.)

"Display all the ingredients for recipes that contain carrots." (JOIN on the ingredient ID.)

Never fear. In the next chapter we'll show you all about solving these problems (and more) by using JOINs. And because so few commercial implementations of SQL support INTERSECT, we'll show how to use a JOIN to solve many problems that might otherwise require an INTERSECT.

Difference

What's the difference between 21 and 10? If you answered 11, you're on the right track! A *difference* operation (sometimes also called subtract, minus, or except) takes one set of values and removes the set of values from a second set. What remains is the set of values in the first set that are *not* in the second set. (As you'll see later, EXCEPT is the keyword used in the SQL Standard.)

Difference in Set Theory

Difference is another very powerful mathematical tool. As a scientist, you might be interested in finding what's different about two sets of chemical or physical sample data. For example, a pharmaceutical research chemist might have two compounds that seem to be very similar, but one provides a certain beneficial effect and the other does not. Finding what's different about the two compounds might help uncover why one works and the other does not. As an engineer, you might have two similar designs, but one works better than the other. Finding the difference between the two designs could be crucial to eliminating structural flaws in future buildings.

Let's take a look at difference in action by examining two sets of numbers. The first set of numbers is as follows.

1, 5, 8, 9, 32, 55, 78

The second set of numbers is as follows.

3, 7, 8, 22, 55, 71, 99

The difference of the first set of numbers minus the second set of numbers is the numbers that exist in the first set but not the second.

1, 5, 9, 32, 78

Note that you can turn the previous difference operation around. Thus, the difference of the second set minus the first set is

3, 7, 22, 71, 99

The members of each set don't have to be single values. In fact, you'll most likely be dealing with sets of rows when trying to solve problems with SQL.

Earlier in this chapter we said that when a member of a set is something more than a single number or value, each member of the set has multiple attributes (bits of information that describe the properties of each member). For example, your favorite stew recipe is a complex member of the set of all recipes that contains many different ingredients. You can think of each ingredient as an attribute of your complex stew member.

To find the difference between two sets of complex set members, you have to find the members that match on all the attributes in the second set with members in the first set. Don't forget that all of the members in each set you're trying to compare must have the same number and type of attributes. Remove from the first set all the matching members you find in the second set, and the result is the difference. For example, suppose you have a complex set like the one below. Each row represents a member of the set (a stew recipe), and each column denotes a particular attribute (an ingredient).

Potatoes	Water	Lamb	Peas
Rice	Chicken Stock	Chicken	Carrots
Pasta	Water	Tofu	Snap Peas
Potatoes	Beef Stock	Beef	Cabbage
Pasta	Water	Pork	Onions

A second set might look like this.

Potatoes	Water	Lamb	Onions
Rice	Chicken Stock	Turkey	Carrots
Pasta	Vegetable Stock	Tofu	Snap Peas
Potatoes	Beef Stock	Beef	Cabbage
Beans	Water	Pork	Onions

The difference of the first set minus the second set is the objects in the first set that don't exist in the second set.

Potatoes	Water	Lamb	Peas
Rice	Chicken Stock	Chicken	Carrots
Pasta	Water	Tofu	Snap Peas
Pasta	Water	Pork	Onions

Difference between Result Sets

When you're dealing with rows in a set of data fetched with SQL, the attributes are the individual columns. For example, suppose you have a set of rows returned by a query like the following one. (These are recipes from John's cookbook.)

Recipe	Starch	Stock	Meat	Vegetable
Lamb Stew	Potatoes	Water	Lamb	Peas
Chicken Stew	Rice	Chicken Stock	Chicken	Carrots
Veggie Stew	Pasta	Water	Tofu	Snap Peas
Irish Stew	Potatoes	Beef Stock	Beef	Cabbage
Pork Stew	Pasta	Water	Pork	Onions

A second query result set might look like the following. (These are recipes from Mike's cookbook.)

Recipe	Starch	Stock	Meat	Vegetable
Lamb Stew	Potatoes	Water	Lamb	Peas
Turkey Stew	Rice	Chicken Stock	Turkey	Carrots
Veggie Stew	Pasta	Vegetable Stock	Tofu	Snap Peas
Irish Stew	Potatoes	Beef Stock	Beef	Cabbage
Pork Stew	Beans	Water	Pork	Onions

The difference between John's recipes and Mike's recipes (John's minus Mike's) is all the recipes in John's cookbook that *do not* appear in Mike's cookbook.

Recipe	Starch	Stock	Meat	Vegetable
Chicken Stew	Rice	Chicken Stock	Chicken	Carrots
Veggie Stew	Pasta	Water	Tofu	Snap Peas
Pork Stew	Pasta	Water	Pork	Onions

You can also turn this problem around. Suppose you want to find the recipes in Mike's cookbook that *are not* in John's cookbook. Here's the answer.

Recipe	Starch	Stock	Meat	Vegetable
Turkey Stew	Rice	Chicken Stock	Turkey	Carrots
Veggie Stew	Pasta	Vegetable Stock	Tofu	Snap Peas
Pork Stew	Beans	Water	Pork	Onions

Again, we can use a set diagram to help visualize how a difference operation works. Let's assume you have a nice database containing all your favorite recipes. You really do not like the way onions taste with beef, so you're interested in finding all recipes that contain beef but not onions. Figure 7–3 shows you the set diagram that helps you visualize how to solve this problem.

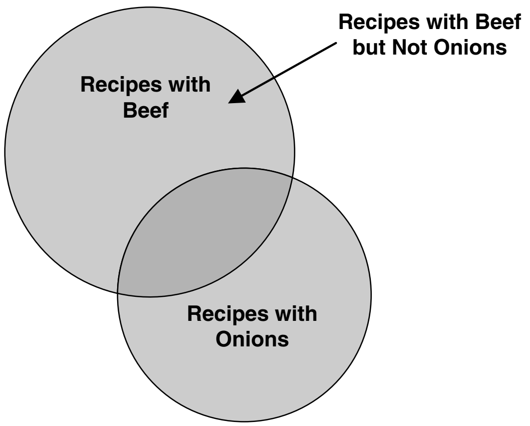

Figure 7–3 *Finding out which recipes have beef but not onions*

The upper full circle represents the set of recipes that contain beef. The lower full circle represents the set of recipes that contain onions. As you remember from the discussion about INTERSECT, where the two circles overlap is where you'll find the recipes that contain both. The dark-shaded part of the upper circle that's not part of the overlapping area represents the set of recipes that contain beef but do not contain onions. Likewise, the part of the lower circle that's not part of the overlapping area represents the set of recipes that contain onions but do not contain beef.

You probably know that you first ask SQL to fetch all the recipes that have beef. Next, you ask SQL to fetch all the recipes that have onions. (As you'll see later in this chapter, the special SQL keyword EXCEPT links the two queries to get the final answer.)

Are you falling into the trap again? (You *did* read Chapter 2, didn't you?) If your recipe table looks like the samples earlier, you might think that you could simply say the following.

> *"Show me the recipes that have beef as the meat ingredient and that do not have onions as the vegetable ingredient."*

Translation	Select the recipe name from the recipes table where meat ingredient is beef and vegetable ingredient is not onions
Clean Up	Select ~~the~~ recipe name from ~~the~~ recipes ~~table~~ where meat ingredient ~~is~~ = beef and vegetable ingredient ~~is not~~ <> onions

SQL	`SELECT RecipeName`
	`FROM Recipes`
	`WHERE MeatIngredient = 'Beef'`
	`   AND VegetableIngredient <> 'Onions'`

Again, as you learned in Chapter 2, a single Recipes table isn't such a hot idea. (Pun intended!) What about recipes that have ingredients other than meat and vegetables? What about the fact that some recipes have many ingredients and others have only a few? A correctly designed Recipes database will have a separate Recipe_Ingredients table with one row per recipe per ingredient. Each ingredient row will have only one ingredient, so no one row can be both beef and onions at the same time. You'll need to first find all the beef rows, then find all the onions rows, then difference them on RecipeID.

How about a more complex problem? Let's say you hate carrots, too. A set diagram to visualize the solution might look like Figure 7–4.

First you need to find the set of recipes that have beef, and then get the difference with either the set of recipes containing onions or the set containing

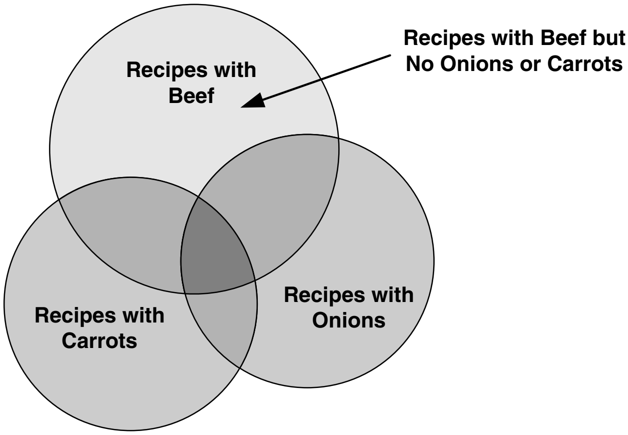

Figure 7–4 *Finding out which recipes have beef but no onions or carrots*

carrots. Take that result and get the difference again with the remaining set (onions or carrots) to leave only the recipes that have beef but no carrots or onions (the light-shaded area in the upper circle).

Problems You Can Solve with Difference

Unlike intersection (which looks for common members of two sets), difference looks for members that are in one set but *not* in another set. Here's just a small sample of the problems you can solve using a difference technique with data from the sample databases.

> *"Show me customers whose names are not the same as any employee."*
>
> *"Find all the customers who ordered a bicycle but did not order a helmet."*
>
> *"List the entertainers who played engagements for customer Bonnicksen but did not play any engagement for customer Rosales."*
>
> *"Show me the students who have an average score of 85 or better in Art but do not have an average score of 85 or better in Computer Science."*
>
> *"Find the bowlers who had a raw score of 155 or better at Thunderbird Lanes but not at Bolero Lanes."*
>
> *"Show me the recipes that have beef but not garlic."*

One of the limitations of using a pure difference is that the values must match in all the columns in each result set. This works well if you're finding the difference between two or more sets from the same table—for example, customers who ordered bicycles and customers who ordered helmets. It also works well when you're finding the difference between sets from tables that have similar columns—for example, customer names and employee names.

In many cases, however, you'll want to find solutions that require a match on only a few column values from each set. For this type of problem, SQL provides an OUTER JOIN operation, which is an intersection on key values that includes the unmatched values from one or both of the two sets. Here's a sample of problems you can solve with an OUTER JOIN.

> *"Show me customers who do not live in the same city as any employees." (OUTER JOIN on the city name.)*
>
> *"List customers and the entertainers they did not book." (OUTER JOIN on the engagement number.)*
>
> *"Find the agents who are not in the same ZIP Code as any entertainer." (OUTER JOIN on the ZIP Code.)*
>
> *"Show me the students who do not have the same first name as any teachers." (OUTER JOIN on the first name.)*
>
> *"Find the bowlers who have an average of 150 or higher who have never bowled a game below 125." (OUTER JOIN on the bowler ID from two different tables.)*
>
> *"Display all the ingredients for recipes that do not have carrots." (OUTER JOIN on the recipe ID.)*

Don't worry! We'll show you all about solving these problems (and more) using OUTER JOINs in Chapter 9. Also, because few commercial implementations of SQL support EXCEPT (the keyword for difference), we'll show how to use an OUTER JOIN to solve many problems that might otherwise require an EXCEPT.

Union

So far we've discussed finding the items that are common in two sets (intersection) and the items that are different (difference). The third type of set operation involves adding two sets (union).

Union in Set Theory

Union lets you combine two sets of similar information into one set. As a scientist, you might be interested in combining two sets of chemical or physical sample data. For example, a pharmaceutical research chemist might have two different sets of compounds that seem to provide a certain beneficial effect. The chemist can union the two sets to obtain a single list of all effective compounds.

Let's take a look at union in action by examining two sets of numbers. The first set of numbers is as follows.

1, 5, 8, 9, 32, 55, 78

The second set of numbers is as follows.

3, 7, 8, 22, 55, 71, 99

The union of these two sets of numbers is the numbers in both sets combined into one new set.

1, 5, 8, 9, 32, 55, 78, 3, 7, 22, 71, 99

Note that the values common to both sets, 8 and 55, appear only once in the answer. Also, the sequence of the numbers in the result set is not necessarily in any specific order. When you ask a database system to perform a UNION, the values returned won't necessarily be in sequence unless you explicitly include an ORDER BY clause. In SQL, you can also ask for a UNION ALL if you want to see the duplicate members.

The members of each set don't have to be just single values. In fact, you'll probably deal with sets of rows when working with SQL.

To find the union of two or more sets of complex members, all the members in each set you're trying to union must have the same number and type of attributes. For example, suppose you have a complex set like the one below. Each row represents a member of the set (a stew recipe), and each column denotes a particular attribute (an ingredient).

Potatoes	Water	Lamb	Peas
Rice	Chicken Stock	Chicken	Carrots
Pasta	Water	Tofu	Snap Peas
Potatoes	Beef Stock	Beef	Cabbage
Pasta	Water	Pork	Onions

A second set might look like the following.

Potatoes	Water	Lamb	Onions
Rice	Chicken Stock	Turkey	Carrots
Pasta	Vegetable Stock	Tofu	Snap Peas
Potatoes	Beef Stock	Beef	Cabbage
Beans	Water	Pork	Onions

The union of these two sets is the set of objects from both sets. Duplicates are eliminated.

Potatoes	Water	Lamb	Peas
Rice	Chicken Stock	Chicken	Carrots
Pasta	Water	Tofu	Snap Peas
Potatoes	Beef Stock	Beef	Cabbage
Pasta	Water	Pork	Onions
Potatoes	Water	Lamb	Onions
Rice	Chicken Stock	Turkey	Carrots
Pasta	Vegetable Stock	Tofu	Snap Peas
Beans	Water	Pork	Onions

Combining Result Sets Using a Union

It's a small leap from sets of complex objects to rows in SQL result sets. When you're dealing with rows in a set of data that you fetch with SQL, the attributes are the individual columns. For example, suppose you have a set of rows returned by a query like the following one. (These are recipes from John's cookbook.)

Recipe	Starch	Stock	Meat	Vegetable
Lamb Stew	Potatoes	Water	Lamb	Peas
Chicken Stew	Rice	Chicken Stock	Chicken	Carrots
Veggie Stew	Pasta	Water	Tofu	Snap Peas
Irish Stew	Potatoes	Beef Stock	Beef	Cabbage
Pork Stew	Pasta	Water	Pork	Onions

A second query result set might look like this one. (These are recipes from Mike's cookbook).

Recipe	**Starch**	**Stock**	**Meat**	**Vegetable**
Lamb Stew	Potatoes	Water	Lamb	Peas
Turkey Stew	Rice	Chicken Stock	Turkey	Carrots
Veggie Stew	Pasta	Vegetable Stock	Tofu	Snap Peas
Irish Stew	Potatoes	Beef Stock	Beef	Cabbage
Pork Stew	Beans	Water	Pork	Onions

The union of these two sets is all the rows in both sets. Maybe John and Mike decided to write a cookbook together, too!

Recipe	**Starch**	**Stock**	**Meat**	**Vegetable**
Lamb Stew	Potatoes	Water	Lamb	Peas
Chicken Stew	Rice	Chicken Stock	Chicken	Carrots
Veggie Stew	Pasta	Water	Tofu	Snap Peas
Irish Stew	Potatoes	Beef Stock	Beef	Cabbage
Pork Stew	Pasta	Water	Pork	Onions
Turkey Stew	Rice	Chicken Stock	Turkey	Carrots
Veggie Stew	Pasta	Vegetable Stock	Tofu	Snap Peas
Pork Stew	Beans	Water	Pork	Onions

Let's assume you have a nice database containing all your favorite recipes. You really like recipes with either beef or onions, so you want a list of recipes that contain either ingredient. Figure 7–5 (on page 232) shows you the set diagram that helps you visualize how to solve this problem.

The upper circle represents the set of recipes that contain beef. The lower circle represents the set of recipes that contain onions. The union of the two circles gives you all the recipes that contain either ingredient, with duplicates eliminated where the two sets overlap. As you probably know, you first ask SQL to fetch all the recipes that have beef. In the second query, you ask SQL

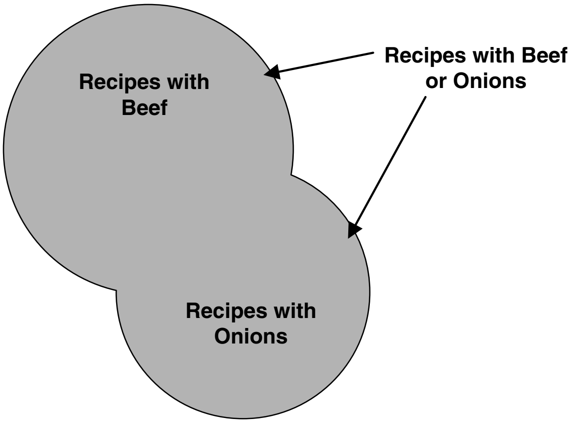

Figure 7–5 *Finding out which recipes have either beef or onions*

to fetch all the recipes that have onions. As you'll see later, the SQL keyword UNION links the two queries to get the final answer.

By now you know that it's not a good idea to design a recipes database with a single table. Instead, a correctly designed recipes database will have a separate Recipe_Ingredients table with one row per recipe per ingredient. Each ingredient row will have only one ingredient, so no one row can be both beef or onions at the same time. You'll need to first find all the recipes that have a beef row, then find all the recipes that have an onions row, and then union them.

Problems You Can Solve with Union

A union lets you "mush together" rows from two similar sets—with the added advantage of no duplicate rows. Here's a sample of the problems you can solve using a union technique with data from the sample databases.

"Show me all the customer and employee names and addresses."

"List all the customers who ordered a bicycle combined with all the customers who ordered a helmet."

"List the entertainers who played engagements for customer Bonnicksen combined with all the entertainers who played engagements for customer Rosales."

"Show me the students who have an average score of 85 or better in Art together with the students who have an average score of 85 or better in Computer Science."

> *"Find the bowlers who had a raw score of 155 or better at Thunderbird Lanes combined with bowlers who had a raw score of 140 or better at Bolero Lanes."*
>
> *"Show me the recipes that have beef together with the recipes that have garlic."*

As with other "pure" set operations, one of the limitations is that the values must match in all the columns in each result set. This works well if you're unioning two or more sets from the same table—for example, customers who ordered bicycles and customers who ordered helmets. It also works well when you're performing a union on sets from tables that have like columns—for example, customer names and addresses and employee names and addresses. We'll explore the uses of the SQL UNION operator in detail in Chapter 10.

In many cases where you would otherwise union rows from the same table, you'll find that using DISTINCT (to eliminate the duplicate rows) with complex criteria on joined tables will serve as well. We'll show you all about solving problems this way using JOINs in Chapter 8, INNER JOINs.

SQL Set Operations

Now that you have a basic understanding of set operations, let's look briefly at how they're implemented in SQL.

Classic Set Operations versus SQL

As noted earlier, not many commercial database systems yet support set intersection (INTERSECT) or set difference (EXCEPT) directly. The current SQL Standard, however, clearly defines how these operations should be implemented. We think that these set operations are important enough to at least warrant an overview of the syntax.

As promised, we'll show you alternative ways to solve an intersection or difference problem in later chapters using JOINs. Because most database systems do support UNION, Chapter 10 is devoted to its use. The remainder of this chapter gives you an overview of all three operations.

Finding Common Values: INTERSECT

Let's say you're trying to solve the following seemingly simple problem.

"Show me the orders that contain both a bike and a helmet."

Translation — Select the distinct order numbers from the order details table where the product number is in the list of bike and helmet product numbers

Clean Up — Select ~~the~~ distinct order numbers from ~~the~~ order details ~~table~~ where ~~the~~ product number ~~is~~ in ~~the list of~~ bike and helmet product numbers

SQL

```
SELECT DISTINCT OrderNumber
FROM Order_Details
WHERE ProductNumber IN (1, 2, 6, 10, 11, 25, 26)
```

> ❖ **Note** Readers familiar with SQL might ask why we didn't JOIN Order_Details to Products and look for bike or helmet product names. The simple answer is that we haven't introduced the concept of a JOIN yet, so we built this example on a single table using IN and a list of known bike and helmet product numbers.

That seems to do the trick at first, but the answer includes orders that contain either a bike *or* a helmet, and you really want to find ones that contain *both* a bike *and* a helmet! If you visualize orders with bicycles and orders with helmets as two distinct sets, it's easier to understand the problem. Figure 7-6 shows one possible relationship between the two sets of orders using a set diagram.

Actually, there's no way to predict in advance what the relationship between two sets of data might be. In Figure 7-6, some orders have a bicycle in the list of products ordered, but no helmet. Some have a helmet, but no bicycle. The overlapping area, or intersection, of the two sets is where you'll find orders that have both a bicycle and a helmet. Figure 7-7 shows another case where *all* orders that contain a helmet also contain a bicycle, but some orders that contain a bicycle do not contain a helmet.

Seeing "both" in your request suggests you're probably going to have to break the solution into separate sets of data and then link the two sets in some way. (Your request also needs to be broken into two parts.)

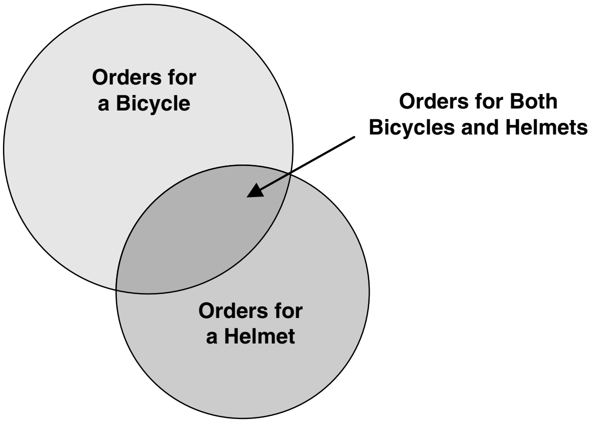

Figure 7–6 *One possible relationship between two sets of orders*

"Show me the orders that contain a bike."

Translation	Select the distinct order numbers from the order details table where the product number is in the list of bike product numbers
Clean Up	Select ~~the~~ distinct order numbers from ~~the~~ order details ~~table~~ where ~~the~~ product number ~~is~~ in ~~the list of~~ bike product numbers

SQL	`SELECT DISTINCT OrderNumber`
	 `FROM Order_Details`
	 `WHERE ProductNumber IN (1, 2, 6, 11)`

"Show me the orders that contain a helmet."

Translation	Select the distinct order numbers from the order details table where the product number is in the list of helmet product numbers

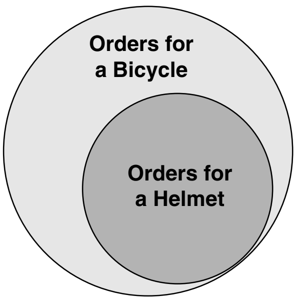

Figure 7–7 *All orders for a helmet also contain an order for a bicycle.*

Clean Up Select ~~the~~ distinct order numbers from ~~the~~ order details ~~table~~ where ~~the~~ product number ~~is~~ in ~~the list of~~ helmet product numbers

SQL
```
SELECT DISTINCT OrderNumber
FROM Order_Details
WHERE ProductNumber IN (10, 25, 26)
```

Now you're ready to get the final solution by using—you guessed it—an *intersection* of the two sets. Figure 7–8 shows the SQL syntax diagram that handles this problem. (Note that you can use INTERSECT more than once to combine multiple SELECT statements.)

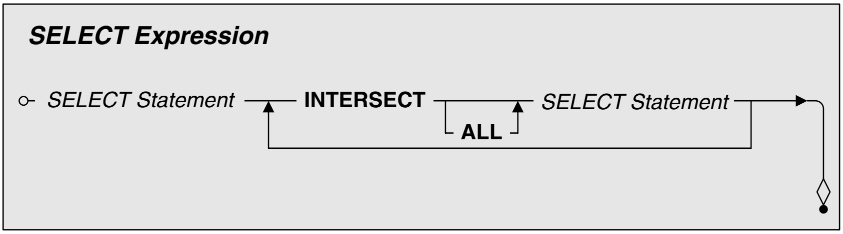

Figure 7–8 *Linking two SELECT statements with INTERSECT*

You can now take the two parts of your request and link them with an INTERSECT operator to get the correct answer.

SQL
```
SELECT DISTINCT OrderNumber
FROM Order_Details
WHERE ProductNumber IN (1, 2, 6, 11)
INTERSECT
SELECT DISTINCT OrderNumber
FROM Order_Details
WHERE ProductNumber IN (10, 25, 26)
```

The sad news is that not many commercial implementations of SQL yet support the INTERSECT operator. But all is not lost! Remember that the primary key of a table uniquely identifies each row. (You don't have to match on all the fields in a row—just the primary key—to find unique rows that intersect.) We'll show you an alternative method (JOIN) in Chapter 8 that can solve this type of problem in another way. The good news is that most commercial implementations of SQL *do* support JOIN.

Finding Missing Values: EXCEPT (DIFFERENCE)

Okay, let's go back to the bicycles and helmets problem again. Let's say you're trying to solve this seemingly simple request as follows.

"Show me the orders that contain a bike but not a helmet."

Translation Select the distinct order numbers from the order details table where the product number is in the list of bike product numbers and product number is not in the list of helmet product numbers

Clean Up Select ~~the~~ distinct order numbers from ~~the~~ order details ~~table~~ where ~~the~~ product number ~~is~~ in ~~the list of~~ bike product numbers and product number ~~is~~ not in ~~the list of~~ helmet product numbers

SQL
```
SELECT DISTINCT OrderNumber
FROM Order_Details
WHERE ProductNumber IN (1, 2, 6, 11)
   AND ProductNumber NOT IN (10, 25, 26)
```

Unfortunately, the answer shows you orders that contain only a bike! The problem is that the first IN clause finds detail rows containing a bicycle, but the second IN clause simply eliminates helmet rows. If you visualize orders with bicycles and orders with helmets as two distinct sets, you'll find this easier to understand. Figure 7–9 shows one possible relationship between the two sets of orders.

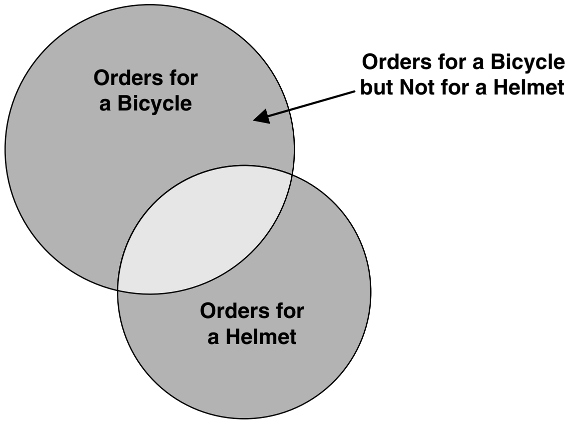

Figure 7–9 *Orders for a bicycle that do not also contain a helmet*

Seeing "except" or "but not" in your request suggests you're probably going to have to break the solution into separate sets of data and then link the two sets in some way. (Your request also needs to be broken into two parts.)

"Show me the orders that contain a bike."

Translation Select the distinct order numbers from the order details table where the product number is in the list of bike product numbers

Clean Up Select ~~the~~ distinct order numbers from ~~the~~ order details ~~table~~ where ~~the~~ product number ~~is~~ in ~~the list of~~ bike product numbers

SQL
```
SELECT DISTINCT OrderNumber
FROM Order_Details
WHERE ProductNumber IN (1, 2, 6, 11)
```

"Show me the orders that contain a helmet."

Translation Select the distinct order numbers from the order details table where the product number is in the list of helmet product numbers

Clean Up Select ~~the~~ distinct order numbers from ~~the~~ order details ~~table~~ where ~~the~~ product number ~~is~~ in ~~the list of~~ helmet product numbers

SQL
```
SELECT DISTINCT OrderNumber
FROM Order_Details
WHERE ProductNumber IN (10, 25, 26)
```

Now you're ready to get the final solution by using—you guessed it—a *difference* of the two sets. SQL uses the EXCEPT keyword to denote a difference operation. Figure 7-10 shows you the SQL syntax diagram that handles this problem.

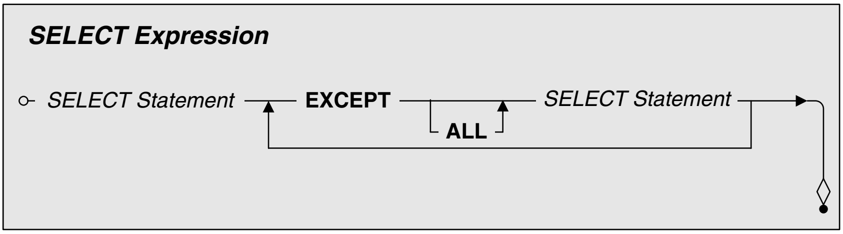

Figure 7–10 *Linking two SELECT statements with EXCEPT*

You can now take the two parts of your request and link them with an EXCEPT operator to get the correct answer.

SQL
```
SELECT DISTINCT OrderNumber
FROM Order_Details
WHERE ProductNumber IN (1, 2, 6, 11)
EXCEPT
SELECT DISTINCT OrderNumber
FROM Order_Details
WHERE ProductNumber IN (10, 25, 26)
```

Remember from our earlier discussion about the difference operation that the sequence of the sets matters. In this case you're asking for bikes "except" helmets. If you want to find out the opposite case—orders for helmets that do not include bikes—you can turn it around as follows.

SQL
```
SELECT DISTINCT OrderNumber
FROM Order_Details
WHERE ProductNumber IN (10, 25, 26)
EXCEPT
SELECT DISTINCT OrderNumber
FROM Order_Details
WHERE ProductNumber IN (1, 2, 6, 11)
```

The sad news is that not many commercial implementations of SQL yet support the EXCEPT operator. Hang on to your helmet! Remember that the primary key of a table uniquely identifies each row. (You don't have to match on all the fields in a row—just the primary key—to find unique rows that are different.) We'll show you an alternative method (OUTER JOIN) in Chapter 9 that can solve this type of problem in another way. The good news is that most commercial implementations of SQL *do* support OUTER JOIN.

Combining Sets: UNION

One more problem about bicycles and helmets, then we'll pedal on to the next chapter. Let's say you're trying to solve this request, which looks simple enough on the surface.

"Show me the orders that contain either a bike or a helmet."

Translation — Select the distinct order numbers from the order details table where the product number is in the list of bike and helmet product numbers

Clean Up — Select ~~the~~ distinct order numbers from ~~the~~ order details ~~table~~ where ~~the~~ product number ~~is~~ in ~~the list of~~ bike and helmet product numbers

SQL
```
SELECT DISTINCT OrderNumber
FROM Order_Details
WHERE ProductNumber IN (1, 2, 6, 10, 11, 25, 26)
```

Actually, that works just fine! So why use a UNION to solve this problem? The truth is, you probably would not. However, if we make the problem more complicated, a UNION would be useful.

"List the customers who ordered a bicycle together with the vendors who provide bicycles."

Unfortunately, answering this request involves creating a couple of queries using JOIN operations, then using UNION to get the final result. Because we haven't shown you how to do a JOIN yet, we'll save solving this problem for Chapter 10. Gives you something to look forward to, doesn't it?

Let's get back to the "bicycles or helmets" problem and solve it with a UNION. If you visualize orders with bicycles and orders with helmets as two distinct sets, then you'll find it easier to understand the problem. Figure 7–11 shows you one possible relationship between the two sets of orders.

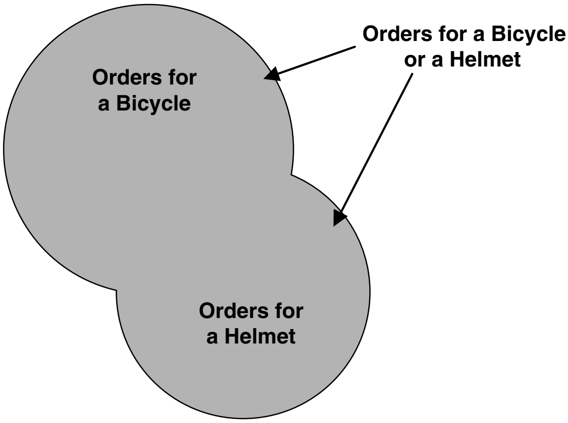

Figure 7–11 *Orders for bicycles or helmets*

Seeing "either," "or," or "together" in your request suggests that you'll need to break the solution into separate sets of data and then link the two sets with a UNION. This particular request can be broken into two parts.

"Show me the orders that contain a bike."

Translation	Select the distinct order numbers from the order details table where the product number is in the list of bike product numbers
Clean Up	Select ~~the~~ distinct order numbers from ~~the~~ order details ~~table~~ where ~~the~~ product number ~~is~~ in ~~the list of~~ bike product numbers

SQL	`SELECT DISTINCT OrderNumber`
	`FROM Order_Details`
	`WHERE ProductNumber IN (1, 2, 6, 11)`

"Show me the orders that contain a helmet."

Translation	Select the distinct order numbers from the order details table where the product number is in the list of helmet product numbers
Clean Up	Select ~~the~~ distinct order numbers from ~~the~~ order details ~~table~~ where ~~the~~ product number ~~is~~ in ~~the list of~~ helmet product numbers
SQL	`SELECT DISTINCT OrderNumber` `FROM Order_Details` `WHERE ProductNumber IN (10, 25, 26)`

Now you're ready to get the final solution by using—you guessed it—a *union* of the two sets. Figure 7-12 shows the SQL syntax diagram that handles this problem.

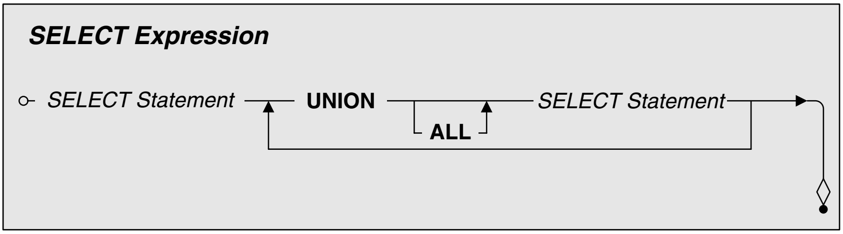

Figure 7–12 *Linking two SELECT statements with UNION*

You can now take the two parts of your request and link them with a UNION operator to get the correct answer.

SQL

```
SELECT DISTINCT OrderNumber
FROM Order_Details
WHERE ProductNumber IN (1, 2, 6, 11)
UNION
SELECT DISTINCT OrderNumber
FROM Order_Details
WHERE ProductNumber IN (10, 25, 26)
```

The good news is that most commercial implementations of SQL support the UNION operator. As is perhaps obvious from the examples, a UNION might be doing it the hard way when you want to get an "either-or" result from a single table. UNION is most useful for compiling a list from several similarly structured but different tables. We'll explore UNION in much more detail in Chapter 10.

SUMMARY

We began this chapter by discussing the concept of a set. Next, we discussed each of the major set operations implemented in SQL in detail—intersection, difference, and union. We showed how to use set diagrams to visualize the problem you're trying to solve. Finally, we introduced you to the basic SQL syntax and keywords (INTERSECT, EXCEPT, and UNION) for all three operations just to whet your appetite.

At this point you're probably saying, "Wait a minute, why did you show me three kinds of set operations—two of which I probably can't use?" Remember the title of the chapter: Thinking in Sets. If you're going to be at all successful solving complex problems, you'll need to break your problem into result sets of information that you then link back together.

So, if your problem involves "it must be this *and* it must be that," you might need to solve the "this" and then the "that" and then link them to get your final solution. The SQL Standard defines a handy INTERSECT operation—but an INNER JOIN might work just as well. Read on in Chapter 8.

Likewise, if your problem involves "it must be this *but it must not be* that," you might need to solve the "this" and then the "that" and then subtract the "that" from the "this" to get your answer. We showed you the SQL Standard EXCEPT operation, but an OUTER JOIN might also do the trick. Get the details in Chapter 9.

Finally, we showed you how to add sets of information using a UNION. As promised, we'll really get into UNION in Chapter 10.

This page intentionally left blank

Index

A
abbreviations
in field names, 22
in table names, 31
Access. See under Microsoft
acronyms
in field names, 22
in table names, 31
aggregate functions, 74, 375–377, 416–428. *See also* AVG; COUNT; COUNT(*); MAX; MIN; SUM
column name in, 419
in filters, 428–430
grouping of, 442–444 (*See also* GROUP BY)
imbedding of, 428
multiple, 427–428
Nulls and, 417, 420
overview, 416–418
sample statements, 431–438
syntax, 416, 416*f*, 595*f*
uses of, 416
WHERE clause in, 419, 430
aliases
for column names, 131
for tables, 252–254, 252*f*
ALL
in subqueries, 386–389, 387*f*, 393–394, 406–407
syntax, 487*f*, 598*f*
in UNIONs, 228–230, 341, 343, 345, 348
Allaire, *Cold Fusion,* 6
American National Standards Institute (ANSI), 57. *See also* SQL Standard
American Standard Code for Information interchange (ASCII), 107
analytic databases, defined, 4
AND, 179, 180*f*
null values and, 195–196
with OR, 183–184
sample statements, 204–205
Ansa Software, *Paradox,* 5
ANSI (American National Standards Institute), 57
ANSI/ISO standard. *See* SQL Standard
ANSI NCITS-H2, 62
"ANSI X3.168-1989 Database Language Embedded SQL," 58
"ANSI X3.135-1986 Database Language SQL" (SQL/86), 57–58
ANY, 386–389, 387*f*, 400, 598*f*
approximate numeric data type, 108–109
archiving of data, 547–549, 554–555
deleting data after archiving, 574–575, 579
artificial primary keys, 41–42, 41*f*
AS, 130–131, 252–253, 252*f*
ASC, 91, 96, 98–99, 100–101
ASCII (American Standard Code for Information interchange), 107, 157–159
Ashton Tate, *dBase,* 5
asterisk shortcut, 83–84, 83*f*
ASYMMETRIC BETWEEN, 165
attribute, defined, 6, 215. *See also* fields
AUTO_INCREMENT, 543
AutoNumber *(Access),* 543
AVG, 416*f*, 423–424, 432, 435, 595*f*. *See also* aggregate functions

B
base tables, defined, 9
Begg, Carolyn, 50
BETWEEN, 154, 164–167, 164*f*, 191–193
ASYMMETRIC, 165
SYMMETRIC, 165
BIGINT data type, 108
BINARY data type, 108
BINARY LARGE OBJECT data type, 108

BIT data type, 108, 109
BIT_LENGTH, 595*f*
BIT VARYING data type, 108
BLOB data type, 108
Boolean data type, 109

C
calculated columns, naming of, 129–131, 130*f*
calculated data. *See also* expressions
  in fields
    disadvantages of, 24, 25
    resolving of, 33
    uses of, 514–516
  in SELECT statements, 74
Call-Level Interface (CLI) specification, 61–62
Cartesian product, 248, 297
cascade deletion rule, 45, 46*f*
case sensitivity
  in collation of character string literals, 157–160
  in comparison of character string literals, 171
CAST, 110–112, 110*f*
  for datetime literal compatibility, 117, 119–120, 141, 142–143, 144–145
  for mathematical expression compatibility, 123–124
  syntax, 110*f*, 594*f*
Chamberlin, Donald, 54
changing data. *See* DELETE; INSERT; UPDATE
CHARACTER (CHAR) data type, 107
CHARACTER (CHAR) LARGE OBJECT data type, 107
CHARACTER (CHAR) VARYING data type, 107
character string functions, syntax, 594*f*
character string literals
  collating sequence of, 157–160
  concatenating, 118–119, 118*f*
  less than and greater than comparisons, 162
  LIKE and, 169–173, 170*f*
  MAX and, 424
  range condition predicates, 166
  specifying, 112–114, 113*f*
  syntax, 113*f*, 492*f*
CHAR_LENGTH, 595*f*
CLI (Call-Level Interface) specification, 61–62
client/server computing, introduction of, 5, 65
CLOB data type, 107
Codd, Edgar F., 4–5, 54
Cold Fusion, 6
collating sequence
  of character string literals, 157–160
  ORDER BY clause and, 88–89
columns
  added, SELECT statement and, 84
  aliases for, 131
  calculated, naming of, 129–131, 130*f*
  column references, 245–246, 246*f*, 249, 593*f*
  deleted, SELECT statement and, 84
  ordering of, 82–83
  as term, 72
COMMIT, 506
comparison predicates, 156–164
  comparison operators, 154
  equality and inequality, 160–161
  exclusive, 161
  inclusive, 161
  less than and greater than, 162–164
  NOT operator and, 185–186
  string values collating sequence, 157–160
  syntax, 157*f*, 597*f*
  types of, 156–157, 157*f*
composite primary keys (CPK)
  defined, 8, 39
  use of, 39
Computer Associates International, Inc., 56
concatenation expressions, 117–120, 118*f*
  length limits for returns, 119
  in SELECT clause, 128–129
  storing as data, 24
  syntax, 118*f*
concatenation operators, 117–118, 118*f*
Connolly, Thomas, 50
copying of data, 547–549
correlation names (aliases)
  for column names, 131
  for tables, 252–254, 252*f*
CORRESPONDING clause, 343
COUNT, 375–376, 420–421. *See also* aggregate functions

DISTINCT and, 420, 469
HAVING COUNT trap, 481–485
Null values and, 420
sample statements, 427–428, 431, 435
syntax, 376*f*, 416*f*, 595*f*
COUNT(*), 375–376, 418–420
DISTINCT and, 421
Null values and, 417
sample statements, 395, 397, 399, 437, 438
syntax, 376*f*, 416*f*, 595*f*
WHERE clause of, 419
CPK. *See* composite primary keys

D

data. *See also* data types
defined, 75
dynamic, defined, 4
static, defined, 4
vs. information, 75–77, 76*f*
Database 2 (DB2). *See under* IBM
database design. *See also* database structure
books on, 16–17
importance of understanding, 19–20
methodology, good
analysis of database with, 50
value of, 16
vs. database theory, 16
vs. implementation, 16
Database Design for Mere Mortals (Hernandez), 5, 16, 42, 50
database implementation, *vs.* database design, 16
database models
relational, history of, 4–6
types of, 4
databases
examples
downloadable version of, 93
schema of, 601–605
types of, 3–4
Web-centric, 6, 65
database structure
fine-tuning
fields, 21–29
tables, 30–42
thorough design process analysis, 50
sound, importance of, 20
Database Systems:A Practical Approach (Connolly and Begg), 50
database theory
importance of understanding, 16
vs. database design, 16
data types, 107–109
changing types, 110–112, 110*f*
extended, 109
Date, C. J., 16, 58
DATE data type, 109
date expressions, 124–126, 125*f*
leap year adjustments, 144–145
sample expressions, 141, 142–143, 144–145
in SELECT clause, 132–133
date functions
in specific software, 607–614
syntax, 594*f*
DATE keyword, 116, 117
date literals
BETWEEN and, 165–166, 167
specifying, 115–116
syntax, 115*f*, 592*f*
datetime arithmetic expressions, 124–127
DATETIME data type. *See also entries under* date; time; timestamp
concatenation of, 119–120
overview of, 109
datetime functions
in specific software, 607–614
syntax, 594*f*
datetime literals
specifying, 115–117, 115*f*
syntax, 115*f*, 592*f*
dBase, 5
DB2 (Database 2). *See under* IBM
DEC data type, 108
DECIMAL data type, 108
DEFAULT, 540
degree of participation
defined, 48
enforcement of, 49–50
setting, 48–50, 49*f*
DELETE
of all rows, 569–570
of data after archiving, 574–575, 579
overview, 567–568

DELETE (*cont.*)
recovering lost data, 506–507, 570
sample statements, 576–583
of some rows, 571–575
syntax, 568–569, 568*f,* 599*f*
transactions and, 506–507, 570
transforming SELECT statement into, 572, 572*f*
uses of, 568, 575–576
verifying accuracy of target materials, 571–572
WHERE clause in, 568, 568*f,* 571–575
sample statements, 576–583
subqueries, 573–575
deletion rule
defined, 44
establishing, 44–45, 46*f*
types of, 45, 46*f*
delimited identifiers, 22, 32
derived tables, in INNER JOIN, 254–256, 255*f*
DESC keyword, 90–91, 100
diagrams. *See* syntax diagrams
difference, 222–227. *See also* EXCEPT
commercial systems' support, 233
limitations of, 227–228
by OUTER JOIN with Null test, 295, 300, 318
between result sets, 224–227
set diagrams of, 225–227, 225*f,* 227*f*
in set theory, 222–224
uses of, 215, 227–228
DISTINCT
AVG and, 424
COUNT/COUNT(*) and, 376, 420, 469
in JOINs, 233, 247*f,* 252*f,* 255*f,* 296*f,* 302*f,* 305*f,* 358, 384
MAX and, 425–426
MIN and, 426–427
in SELECT statements, 84–86, 85*f,* 97, 102
in UNIONs, 348
"does not apply," *vs.* "is not applicable," 137–138
DOUBLE PRECISION data type, 108–109
duplicate fields
disadvantages of, 24–25, 36
identification of, 33
resolving of, 33–38
duplicate rows, eliminating, 84–86, 85*f*
dynamic data, defined, 4

E
EBCDIC (Extended Binary Coded Decimal Interchange Code), 158–159, 197
embedded SQL, specification for, 58–59
enforcement, of participation degree, 49–50
equality comparison predicates, 160–161
escape characters, 172
ESCAPE option, of LIKE predicate, 169*f,* 172–173
Euler, Leonard, 219
Euler diagrams, 219. *See also* set diagrams
events, defined, 7, 7*f,* 32–33
exact number data type, 108
EXCEPT, 222–227
alternatives to, 295
syntax, 236–239, 238*f*
execution, of query
defined, 93
methods, 93
EXISTS, 389–392, 393–394, 402–403, 598*f*
expressions. *See also* value expressions
column, subqueries as, 372–377
aggregate functions for, 375–377, 376*f*
GROUP BY in, 452–453, 461–463
sample statements, 395–499
syntax, 372–375
uses, 392
data types
changing of, 110–112, 110*f* (*See also* CAST)
overview of, 107–109
defined, 106
literal values, specifying, 112–117
character string literals, 112–114, 113*f*
datetime literals, 115–117, 115*f*
numeric literals, 114*f*
naming of, 129–131, 130*f*
overview of, 106
in SELECT clause, 128–135
concatenation expressions, 128–129
date expressions, 132–133
mathematical expressions, 131–132
sample statements, 139–147
value expressions, 135, 135*f*

types of, 117–127
concatenation, 117–120, 118*f*
length limits for returns, 119
in SELECT clause, 128–129
storing as data, 24
syntax, 118*f*
date and time, 124–127
leap year adjustments, 144–145
sample expressions, 141, 142–143, 144–145
in SELECT clause, 132–133
mathematical, 121–124, 121*f*
nulls in, 138–139, 139*f*
sample statements, 140, 143, 145, 147
in SELECT clause, 131–132
uses of, 128
Extended Binary Coded Decimal Interchange Code (EBCDIC), 107, 158–159
extended data types, 109
extensions to SQL standard, 60–61, 73
EXTRACT, 595*f*

F
false, value of, 507–508
Federal Information Processing Standard (FIPS), 61
fields
duplicate
disadvantages of, 24–25, 36
identification of, 33
resolving of, 33–38
fine-tuning of, 21–29
multipart
identification of, 24, 25, 25*f,* 26, 27*f*
resolving of, 25–27, 33
multivalued
identification of, 24, 27–28, 27*f*
resolving of, 27–29, 28*f,* 33
naming of, 21–22
overview of, 7–8
structure, fine-tuning of, 23–25, 23*f*
as term, 72
filters. *See also* HAVING clause; WHERE clause
aggregate functions in, 428–430
subqueries as, 377–392
predicate keywords, 380–392
sample statements, 400–409
syntax, 378–380, 378*f*
uses, 393–394
WHERE *vs.* HAVING, uses of, 478–485
FIPS (Federal Information Processing Standard), 61
FIPS PUB 127, 61
FK (foreign keys), 9, 9*f*
FLOAT data type, 108–109
foreign keys (FK), 9, 9*f*
FROM clause
in FULL OUTER JOIN, 314*f*
in INNER JOIN, 247, 247*f,* 249–250
correlation names for tables, 252–254, 252*f*
embedded JOIN in, 256–261, 257*f,* 260*f*
sample statements, 270–272, 276–277, 280–282, 284–288
embedded select statement in, 254–256, 255*f*
sample statements, 278–282, 284–288
in OUTER JOIN, 296–300, 296*f*
embedded JOIN in, 304–314, 305*f,* 306*f,* 308*f*
sample statements, 320–324, 326–331, 333–334
embedded select statements in, 301–304, 302*f*
sample statements, 320–322, 326–327, 329–331
in SELECT, 74, 74*f*
in UNION JOIN, 318*f*
FULL OUTER JOIN, 314–317, 314*f*
alternatives to, 316, 334
on non-key values, 317
function, defined, 73, 92

G
greater than comparison predicates, 162–164
GROUP BY, 444–458
aggregate functions in, 452, 455, 457–458
mixing columns and expressions, 450–452
nonaggregate column references, 450, 452, 455–456
sample statements, 459–470
as SELECT DISTINCT alternative, 453–454

GROUP BY (*cont.*)
  in SELECT statement, 74*f,* 75
  in subquery, 452–453, 461–463
  syntax, 445–450, 445*f*
  uses for, 458
  without aggregate functions, 453–454

H

HAVING clause, 74*f,* 75
  HAVING COUNT trap, 481–485
  restrictions on, 476
  sample statements, 487–496
  syntax, 476, 476*f*
  uses of, 474–478, 486–487
  vs. WHERE, uses of, 478–485
HAVING COUNT trap, 481–485
Hernandez, Michael J., 5, 16, 42, 50

I

IBM
  DB2 (Database 2)
    concatenation in, 118
    data entry in, 538
    date and time functions in, 607–608
    history of, 5, 56, 57
    not equal to operator, 161
  SAA (Systems Application Architecture), 61
  SQL/Data System (SQL/DS), 56
  SQL development, 54–55
  string values collating sequence, 158–159
  System R, 5, 54–55
identifiers
  delimited, 22, 32
  regular, 22, 32
inequality comparison predicates, 160–161
information
  defined, 75
  vs. data, 75–77, 76*f*
Informix-SE, 65, 118
INGRES, 5, 56, 57, 118
IN keyword, 380–386, 381*f,* 393–394, 404
INNER JOIN, 244–262
  ON clause, 247–250, 247*f*
  column references, 245–246, 246*f,* 249
  correlation names for tables, assigning, 252–254, 252*f*
  embedded JOINS in, 256–261, 257*f,* 260*f*
    sample statements, 270–272, 276–277, 280–282, 284–288
  embedded SELECT statements in, 254–256, 255*f*
    sample statements, 278–282, 284–288
  JOIN-eligible data types, 244–245
  knowledge of tables and, 261–262
  overview of, 244
  sample statements, 263–288
  syntax, 246–262, 247*f,* 257*f*
  of three or more tables, 256–261, 257*f,* 260*f,* 270–277, 347*f*
  of two tables, 247–251, 247*f,* 257*f,* 264–269, 339–340, 340*f*
  uses for, 262–263
  USING clause, 247, 247*f,* 250–251
IN predicate, 154, 167–169, 168*f,* 598*f*
INSERT
  column references in, 540
  individual values and rows, 539–544, 539*f*
  overview, 537–539
  primary keys, automatic generation of, 542–544
  sample statements, 552–563
  with SELECT expression, 544–550, 544*f,* 545*f,* 599*f*
    sample expressions, 554–555, 556–559, 561–562
  sets of data, 544–550, 544*f,* 545*f*
  transactions and, 506
  uses for, 550–551
  with VALUES keyword, 539–544
    sample statements, 552–553, 556, 559–560
    syntax, 539*f,* 599*f*
INTEGER (INT) data type, 108, 109
International Organization for Standardization (ISO), 58. *See also* SQL Standard
INTERSECT, syntax, 234–236, 236*f*
intersection, 216–222. *See also* INTERSECT
  commercial systems' support, 233
  limitations of, 221
  set diagrams of, 218–221, 219*f,* 220*f,* 235*f*
  of set results, 217–221
  in set theory, 216–217
  uses of, 215, 221–222
INTERVAL data type, 109, 125

interval literals, 116, 593*f*
An Introduction to Database Systems (Date), 16
"is not applicable," *vs.* "does not apply," 137–138
IS NULL predicate, 154, 173–175, 174*f,* 197, 205, 295, 300, 318
  syntax, 174*f,* 598*f*
"ISO 9075-1987 Database Language SQL," 58
"ISO 9075:1989 Database Language SQL with Integrity Enhancements" (SQL/89), 58
"ISO/IEC 9075:1992 Database Language SQL," 59–60
ISO (International Organization for Standardization), 58. *See also* SQL standard

J
JOIN. *See also* INNER JOIN; OUTER JOIN
  alternatives to, 250
  commercial systems' support, 250
  default mode, 247
  NATURAL, 251, 301
  overview of, 243–244
  UNION, 317, 318*f*
  uses of, 221–222

K
keys
  composite primary (CPK)
    defined, 8, 39
    use of, 39
  foreign (FK), 9, 9*f*
  overview of, 8–9
  primary (PK)
    artificial, 41–42, 41*f*
    automatic generation of, 542–544
    composite, 8, 39
    criteria for, 39–41, 40*f,* 41*f*
    defined, 6, 8, 9*f,* 39
    functions of, 8, 33
    simple, 39

L
largest value. *See* MAX
LEFT OUTER JOIN
  defined, 295–296
  sample statements, 319–334
  syntax, 296–314, 296*f*
less than comparison predicates, 162–164
LIKE, 154, 169–173, 169*f*
  ESCAPE option, 169*f,* 172–173
  sample pattern strings, 170*t*
  sample statements, 206
  syntax, 169*f,* 598*f*
  wildcard characters, 169–170
linking tables
  advantages of, 14–15
  defined, 13
  defining of, 13–14
  in resolving duplicate fields, 37–38, 37*f,* 38*f*
  in resolving multivalued fields, 28, 29*f*
literals. *See also* character string literals; date literals; datetime literals; numeric literals; time literals; timestamp literals
  interval, 116, 593*f*
  keywords for, 116, 117
  specifying values for
    character string literals, 112–114, 113*f*
    datetime literals, 115–117, 115*f*
    numeric literals, 114*f*
  types
    changing, 110–112, 110*f* (*See also* CAST)
    overview, 107–109
lost data, recovering, 570. *See also* transactions

M
mandatory participation, 46–47, 48*f*
many-to-many relationship, 13–15, 13*f,* 14*f,* 43–44, 44*f*
mathematical expressions, 121–124, 121*f*
  nulls in, 138–139, 139*f*
  sample statements, 140, 143, 145, 147
  in SELECT clause, 131–132
MAX, 376–377, 424–426. *See also* aggregate functions
  in multiple aggregate functions, 427
  sample statements, 396, 398, 434
  syntax, 376*f,* 416*f,* 595*f*
mean values. *See* AVG
members, of set. *See also* IN predicate
  characteristics of, 216–217
  defined, 215

MEMO data type, 107
Microrim, 5
Microsoft
  Access, 29, 65, 92
    AutoNumber in, 543
    concatenation in, 118
    COUNT DISTINCT and, 470
    date and time functions in, 609–610
    date entry in, 537–538
    DELETE in, 570, 570*f*
    GROUP BY in, 456
    JOINs in, 320, 334
    LIKE predicate in, 170
    subqueries, aggregate functions in, 521
    transactions in, 506
    UPDATE
      JOIN clause in, 511–512, 521
  subquery UPDATE expressions, 514
  ODBC (Open Database Connectivity) specification, 62
  SQL Server
    concatenation in, 118
    date and time functions in, 610
    FULL OUTER JOINs in, 316
    history of, 5–6, 65
    Identity data type, 543
    not equal to operator, 161
    OUTER JOIN in, 299
    TOP keyword in, 92
    UPDATE, JOIN clause in, 511–512
  Visual Studio, 6
MIN, 416*f,* 426–427, 433, 595*f*. *See also* aggregate functions
MONEY/CURRENCY data type, 109
multipart fields
  identification of, 24, 25, 25*f,* 26, 27*f*
  resolving of, 25–27, 33
multiple column retrieval, 81–83, 81*f*
multivalued fields
  identification of, 24, 27–28, 27*f*
  resolving of, 27–29, 28*f,* 33
MySQL
  AUTO_INCREMENT in, 543
  concatenation in, 118
  data entry in, 538
  date and time functions in, 611–613
  datetime specification in, 116, 117
  embedded SELECT statements in, 255
  history of, 6
  JOINs in, 334

N

names. *See also* aliases
  for columns, 130–131
  of fields, 21–22
  of tables, 30–32
NATIONAL CHARACTER (CHAR) data type, 107–108
NATIONAL CHARACTER (CHAR) LARGE OBJECT data type, 108
NATIONAL CHARACTER (CHAR) VARYING data type, 108
National Committee for Information Technology Standards (NCITS), 62
National Institute of Standards and Technology (NIST), 61
NATURAL JOIN, 251, 301
NCHAR data type, 107–108
NCHAR VARYING data type, 108
NCITS-H2, 62
NCITS (National Committee for Information Technology Standards), 62
NCLOB data type, 108
NEXTVAL, 543
NIST (National Institute of Standards and Technology), 61
not equal to operator, 161
NOT operator
  in comparison conditions, 185–186
  double, 186–187
  as first keyword, 184–187, 185*f*
  within predicate, 175–178, 176*f*
  sample statements, 186–187
NTEXT data type, 108
Nulls, 135–139. *See also* IS NULL predicate
  COUNT and, 420
  COUNT(*) and, 417
  defined, 136–137
  in mathematical expressions, 138–139, 139*f*
  and multiple condition searches, 193–197, 194*f,* 195*f,* 196*f*
  uses of, 137–138
NUMERIC data type, 108

numeric functions, syntax, 595*f*
numeric literals
BETWEEN and, 164–165
specifying, 114*f*
syntax, 114*f,* 592*f*

O

objects, defined, 7, 32
ODBC (Open Database Connectivity) specification, 62
Office Access 2007, 29
ON clause, 247–250, 247*f,* 296*f,* 297–301
one-to-many relationship, 12, 12*f,* 43, 43*f*
one-to-one relationship, 11–12, 12*f,* 42–43, 43*f*
Open Database Connectivity (ODBC) specification, 62
operational databases, defined, 4
optional participation, 46–47, 48*f*
Oracle
concatenation in, 118
data entry in, 538
date and time functions in, 613–614
GROUP BY in, 456
history of, 5, 55–56
NEXTVAL in, 543
OUTER JOIN in, 299
ORDER BY
ASC keyword, 91, 96, 98–99, 100–101
column name, specifying, 90–91, 351–352, 351*f*
column number, specifying, 351–352, 351*f*
DESC keyword, 90–91, 100
order of precedence in, 88–89
in SELECT statements, 87–92, 87*f*
sample statements, 96, 98–99, 100–101, 146, 200–202
in UNIONs, 351–352, 351*f*
ordering
of columns, 82–83
of rows, 87–92, 87*f*
order of precedence
in mathematical expressions, 121–123
in multiple search conditions, 182
default order, 187–188
and efficiency of search, 190–191
prioritizing of conditions, 188–190
in ORDER BY clause, 88–89
OR operator, 180–182, 181*f*
with AND, 183–184
null values and, 196, 196*t*
sample statements, 203
orphaned records
avoiding, 9, 44–45
defined, 44
OUTER JOIN
ON clause, 296*f,* 297–301
commercial systems' support, 299
embedded JOINS in, 304–314, 305*f,* 306*f,* 308*f*
sample statements, 320–324, 326–331, 333–334
embedded SELECT statements in, 301–304, 302*f*
sample statements, 320–322, 326–327, 329–331
FULL, 314–317, 314*f*
alternatives to, 316, 334
on non-key values, 317
LEFT
defined, 295–296
sample statements, 319–334
syntax, 296–314, 296*f*
overview, 293–295
RIGHT
defined, 295–296
syntax, 296–314, 296*f*
sample statements, 319–334
syntax, 296–314, 296*f*
of three of more tables, 304–314, 306*f,* 308*f*
of two tables, 296–301, 296*f,* 305*f*
uses of, 228, 318–319
USING clause, 296*f,* 297, 300–301
overlapping ranges, checking for, 191–193, 192*f*

P

Paradox, 5
parent-child relationship, 311
parentheses
for combining search conditions, 184
in embedded JOINS within JOINS, 257–258, 257*f,* 304–305, 306*f,* 308*f*
in embedded SELECT statements within JOINS, 254–255, 255*f*
in mathematical expressions, 122–123

parentheses (*cont.*)
in prioritizing of conditions, 189–190
for subqueries, 374
participation degree
defined, 48
enforcement of, 49–50
setting, 48–50, 49*f*
participation type, setting, 46–47, 47*f*, 48*f*
pattern-matching condition. *See* LIKE
PK. *See* primary keys
POSITION, 595*f*
predicates. *See also* search conditions
quantified, 386–389, 387*f*
in WHERE clause (*See also* BETWEEN; comparison predicates; IN predicate; IS NULL predicate; LIKE)
efficiency of, 190–191
expressing in different ways, 197–198
INTERSECT, 234–236, 236*f*
multiple conditions, 178–197
NOT operator, 175–178, 176*f*, 184–187, 185*f*
nulls, evaluation of, 193–197, 194*f*, 195*f*, 196*f*
AND operator, 179, 180*f*, 183–184, 195–196, 204–205
order of precedence. *See* order of precedence
OR operator, 180–182, 181*f*, 184–185, 196, 196*t*, 203
AND and OR together, 183–184
overview, 152–154
sample statements, 198–206
primary keys (PK)
artificial, 41–42, 41*f*
automatic generation of, 542–544
composite, 8, 39
criteria for, 39–41, 40*f*, 41*f*
defined, 6, 8, 9*f*, 39
functions of, 8, 33
simple, 39
primary tables, defined, 11

Q
quantified predicates, 386–389, 387*f*
QUEL (Query Language), 56, 57
query. *See also* SELECT query
defined, 10, 73, 92–93
execution methods, 93
saved, defined, 10
Query Language (QUEL), 56, 57
query optimizers, 190–191, 256, 308–309, 320, 354
quotes
for character string literals, 112–113, 113*f*
single, embedded, 186

R
range, finding data within. *See* BETWEEN
R:BASE, 5
RDBMSs. *See* relational database management systems
REAL data type, 108–109
records
orphaned
avoiding, 9, 44–45
defined, 44
overview of, 8
as term, 72
recovering lost data, 570. *See also* transactions
regular identifiers, 22, 32
relation, 6. *See also* tables
relational database management systems (RDBMSs)
collating sequences of, 88–89
history of, 55–56, 64–65
products, 5–6
relational databases
anatomy of, 6–15
history of model, 4–6
importance of understanding, 15–16
relational database software. *See* relational database management systems
"A Relational Model of Data for Large Shared Databanks" (Codd), 5
Relational Software, Inc., 55–56
Relational Technology, Inc., 56
relationships between tables. *See also* linking tables
degree of participation
defined, 48
enforcement of, 49–50
setting, 48–50, 49*f*
deletion rule
defined, 44

establishing, 44–45, 46*f*
types of, 45, 46*f*
relationship characteristics, establishing, 44–50
type of participation, setting, 46–47, 47*f*, 48*f*
unresolved, defined, 13, 13*f*
Request/Translation/Clean UP/SQL technique, 77–81
restrict deletion rule, 45, 46*f*
result set, defined, 77
RIGHT OUTER JOIN
defined, 295–296
syntax, 296–314, 296*f*
ROLLBACK, 506
ROUND, 521, 522
rows
counting of. *See* COUNT(*)
ordering of, 87–92, 87*f*
as term, 72
row subqueries, 370–371
row value constructors, 370–371

S

SAA (Systems Application Architecture), 61
sample database schema, 601–605
saved query, defined, 10
saving, of SELECT statements, 92–93
scalar subqueries, 370, 372, 373–375
search conditions. *See also* HAVING clause; ON clause; predicates; WHERE clause
combining with parentheses, 184
expressing in different ways, 197–198
order of precedence, 182
default order, 187–188
and efficiency of search, 190–191
prioritizing of conditions, 188–190
syntax diagram, 178*f*, 185*f*, 597*f*
in WHERE clause, 152–153
secondary tables, defined, 11
SELECT clause
expressions in, 128–135
concatenation expressions, 128–129
date expressions, 132–133
mathematical expressions, 131–132
sample statements, 139–147
value expressions, 135, 135*f*
overview of, 74, 74*f*
SELECT expression
defined, 342
INSERT using, 544–550, 544*f*, 545*f*
as part of SELECT operation, 72
syntax, 591*f*
in value expression, 373, 373*f*
SELECT operation
overview, 72
parts of, 72 (*See also* SELECT expression; SELECT query; SELECT statement)
SELECT query, 87–92, 87*f*
defined, 87
as part of SELECT operation, 72
sample statements, 96, 98–99, 100–101, 146, 200–202
sorting rows with, 87–92, 87*f*
syntax, 87–88, 87*f*, 591*f*
SELECT statement
added or deleted columns and, 84
all columns, retrieval of, 83–84, 83*f*
clauses of, 73–75, 74*f* (*See also* FROM clause; GROUP BY; HAVING; SELECT clause; WHERE clause)
defining, 73
duplicate rows, eliminating, 84–86, 85*f*
embedded
in INNER JOIN, 254–256, 255*f*
sample statements, 278–282, 284–288
in OUTER JOIN, 301–304, 302*f*
sample statements, 320–322, 326–327, 329–331
keywords, 73
multiple columns, retrieval of, 81–83, 81*f*
ordering of columns, 82–83
ordering of rows, 87–92, 87*f*
overview of, 73
as part of SELECT operation, 72
sample statements, 93–102
saving, 92–93
subqueries in, 372–377
aggregate functions for, 375–377, 376*f*
GROUP BY in, 452–453, 461–463
sample statements, 395–499
syntax, 372–375
uses, 392
syntax, 74*f*, 77–78, 78*f*, 81*f*, 135*f*, 372*f*
transforming into DELETE statement, 572, 572*f*

SELECT statement (*cont.*)
transforming into UPDATE statement, 505, 506*f*
translating requests into SQL, 77–81
SEQUEL (Structured English Query Language), 54
SEQUEL-XRM, 54
SERIAL/ROWID data type, 109
set(s)
defined, 214–215
members of (*See also* IN predicate)
characteristics of, 216–217
defined, 215
operations on, 215 (*See also* difference; intersection; union)
set diagrams
difference, 225–227, 225*f*, 227*f*
intersection, 218–221, 219*f*, 220*f*, 235*f*
union, 231–232, 232*f*
set identifier columns, 348, 357, 391
set theory
difference in, 222–224
intersection in, 216–217
union in, 228–230
simple primary keys, 39
smallest value. *See* MIN
SMALLINT data type, 108
SOME, 386–389, 387*f*, 393–394, 598*f*
spaces, in expression names, 130
Specifying Queries as Relational Expressions (SQUARE), 55
SQL
early vendor implementations, 55–56
future of, 65
history of, 53–65
importance of learning, 65–66
pronunciation of, 54
SQL3, 62, 62*t*–64*t*
SQL/86, 57–58, 59
SQL/89, 58, 59
SQL/92, 59–60
SQL:1999, 65
SQL:2003, 65
SQL:2007, 65
SQL Access group, 61
SQL/Data System (SQL/DS), 56
SQL/DS (SQL/Data System), 56
SQL Server. See under Microsoft
SQL Standard
alternative standards, 61–62
database models and, 4
data types in, 107–109
datetime expressions and, 127
development of, 56–58
evolution of, 58–64
extensions to, 60–61, 73
identifiers, types of, 22, 32
ORDER BY clause in, 88
parts of, 62, 62*t*–64*t*
SQUARE (Specifying Queries as Relational Expressions), 55
START TRANSACTION, 506
static data, defined, 4
Stonebraker, Michael, 56
stored procedure, defined, 73, 92
Structured English Query Language (SEQUEL), 54
subqueries
for automatic primary keys generation, 543–544
as column expressions, 372–377
aggregate functions for, 375–377, 376*f*
GROUP BY in, 452–453, 461–463
sample statements, 395–499
syntax, 372–375
uses, 392
defined, 370
in DELETE statement, 573–575
as filters, 377–392
predicate keywords, 380–392
sample statements, 400–409
syntax, 378–380, 378*f*
uses, 393–394
row, 370–371
scalar, 370, 372, 373–375
within subqueries, 382–383
subquery UPDATE expressions, 514–516
table, 370, 371–372
types of, 370–372
in UPDATE statement, 508–514
uses for, 392–394
as value expression of aggregate function, 428
WHERE clause of, 373, 380

SUBSTRING, 594*f*
SUM, 416*f,* 421–423, 427–428, 433, 434, 595*f*. *See also* aggregate functions
Sybase Enterprise Application Studio, 6, 161
SYMMETRIC BETWEEN predicate, 165
syntax diagrams
- IN, 168*f,* 381*f,* 598*f*
- aggregate functions, 416*f,* 595*f*
- ALL, 487*f,* 598*f*
- ANY, 487*f,* 598*f*
- asterisk shortcut, 83*f*
- AVG, 416*f,* 595*f*
- CAST, 110*f,* 594*f*
- character string functions, 594*f*
- character string literals, 113*f,* 592*f*
- column references, 246*f,* 593*f*
- comparison predicates, 157*f,* 597*f*
- concatenation expression, 118*f*
- correlation names for table, 252*f*
- COUNT/COUNT(*), 376*f,* 416*f,* 595*f*
- date expression, 125*f*
- date functions, 594*f*
- datetime functions, 594*f*
- datetime literals, 115*f,* 592*f*
- DELETE, 568*f,* 599*f*
- DISTINCT, 85*f*
- EXCEPT, 238*f*
- EXISTS, 598*f*
- filtering with comparison predicate, 378*f*
- GROUP BY, 445*f*
- HAVING, 476*f*
- INSERT
  - using SELECT expression, 544*f,* 545*f,* 599*f*
  - with VALUES, 539*f,* 599*f*
- INTERSECT, 236*f*
- interval literals, 593*f*
- IS NULL, 174*f,* 598*f*
- JOIN, 596*f*
  - FULL OUTER, 314*f*
  - INNER
    - of three or more tables, 257*f,* 260*f,* 347*f*
    - of two tables, 247*f,* 257*f,* 305*f*
  - of more than two tables in alternating sequence, 260*f,* 308*f*
  - OUTER
    - of three or more tables, 306*f,* 308*f*
    - of two tables, 296*f,* 305*f*
    - using SELECT statements, 302*f*
  - of three or more tables, 257*f,* 260*f,* 306*f,* 308*f,* 347*f*
  - of two tables, 247*f,* 257*f,* 296*f,* 305*f*
  - UNION, 318*f*
- LIKE predicate, 169*f,* 598*f*
- literals, 113*f,* 114*f*, 115*f,* 592*f*
- mathematical expressions, 121*f*
- MAX, 376*f,* 416*f,* 595*f*
- MIN, 416*f,* 595*f*
- naming of expression, 130*f*
- NOT, 176*f,* 185*f*
- numeric functions, 595*f*
- numeric literals, 114*f,* 592*f*
- ORDER BY, 351*f*
- BETWEEN predicate, 164*f,* 597*f*
- quantified predicate, 387*f*
- search conditions, 178*f,* 185*f,* 597*f*
- SELECT expression, 373*f,* 591*f*
- SELECT query, 87*f,* 591*f*
- SELECT statement, 74*f,* 78*f,* 81*f,* 135*f,* 372*f*
  - with all clauses, 476*f*
  - embedded in INNER JOIN, 255*f*
  - OUTER JOIN using, 302*f*
  - UNION of two, 342*f*
- SOME, 487*f,* 598*f*
- subquery with IN predicate, 381*f*
- SUM, 416*f,* 595*f*
- time expressions, 126*f*
- time functions, 594*f*
- TIMESTAMP functions, 594*f*
- UNION, 241*f,* 340*f*
  - ORDER BY clause, 351*f*
  - of three tables, 349*f*
  - of two SELECT statements, 342*f*
- UPDATE, 502*f,* 599*f*
- value expression, 134*f,* 373*f,* 592*f*
- WHERE clause, 153*f*

System R, 5, 54–55
Systems Application Architecture (SAA), 61

T

tables
- base, defined, 9
- correlation names (aliases), 252–254, 252*f*
- derived, in INNER JOIN, 254–256, 255*f*

tables (*cont.*)
  fine-tuning of, 30–42
  INNER JOIN of, 247–251, 247*f*
  linking
    advantages of, 14–15
    defined, 13
    defining of, 13–14
    in resolving duplicate fields, 37–38, 37*f*, 38*f*
    in resolving multivalued fields, 28, 29*f*
  logical, defined, 244
  naming of, 22, 30–32
  overview of, 6–7, 6*f*
  primary, defined, 11
  relationships between. *See* relationships between tables
  secondary, defined, 11
  structure, fine-tuning of, 32–33
table subqueries, 370, 371–372
TEXT data type, 107
TIME data type, 109
time expressions, 126–127, 126*f*
time functions
  in specific software, 607–614
  syntax, 594*f*
TIME keyword, 116, 117
time literals
  specifying, 116
  syntax, 115*f*, 592*f*
TIMESTAMP data type, 109
TIMESTAMP functions
  in specific software, 607–614
  syntax, 594*f*
TIMESTAMP keyword, 116
timestamp literals
  specifying, 116
  syntax, 115*f*, 592*f*
TINYINT data type, 108, 109
TOP, 92
totals. *See* SUM
transactions, 506–507, 554, 570
triggers, 42, 516
TRIM, 594*f*
true, value for, 507–508
tuple, 6. *See also* records
type of participation, setting, 46–47, 47*f*, 48*f*

U
UNION
  alternatives to, 352–353, 357–358
  column names in, 345
  commercial systems' support, 241
  of complex SELECT statements, 345–348, 347*f*
  CORRESPONDING clause, 343
  data type computability, 341–342
  multiple, 349–350, 349*f*
  overview, 39–342
  sample statements, 353–365
  set identifier columns and, 348, 357
  set requirements, 341
  of simple SELECT statements, 342–345, 342*f*
  sorting of, 351–352, 351*f*
    sample statements, 354–355, 362–363
  syntax, 239–241, 241*f*, 340–341, 340*f*
  uses of, 215, 232, 233, 352–353
UNION ALL, 228–230, 341, 343, 345, 348
UNION JOIN, 317, 318*f*
union (set operation), 228–233
  combining result sets with, 230–232
  concept, 228–230
  limitations of, 233
  set diagrams, 231–232, 232*f*
  uses of, 215, 232–233
University of California, Berkeley, 5, 56
UNIX, SQL standards for, 61
unresolved relationships, defined, 13, 13*f*
UPDATE
  DISTINCT keyword and, 86
  JOIN in, 508–509, 511–512
  multiple columns, 507–508
  overview, 501–502
  sample statements, 503–504, 517–533
  selected rows, 504
  subquery UPDATE expressions, 514–516
  syntax, 502–503, 502*f*, 599*f*
  transactions and, 506–507, 554
  transforming SELECT statement into, 505, 506*f*
  translation process, 503
  uses for, 516–517
  verifying accuracy of target materials, 505

WHERE clause, 502, 502*f,* 504–505, 506*f,* 507–508
subquery filters, 508–514
USE, 538
USING clause
alternatives to, 301
INNER JOIN, 247, 247*f,* 250–251
OUTER JOIN, 296*f,* 297, 300–301

V
value expressions, 133–135, 134*f*
components of, 134
defined, 134
in SELECT clause, 135, 135*f,* 372–373, 373*f*
sample statements, 139–147
syntax for, 134, 134*f,* 373, 373*f,* 592*f*
VALUES, INSERT statements with, 539–544
sample statements, 552–554, 556, 559–560
syntax, 539*f,* 599*f*
VARCHAR data type, 107
Venn, John, 219
Venn diagrams, 219. *See also* set diagrams
views
defined, 73, 92
overview of, 9–10, 11*f*
Visual Studio, 6

W
"what if?" questions, 106
WHERE clause, 151–156
in aggregate functions, 419, 430
alternatives to, 352–353, 357–358
composition of, 154–156
in DELETE statement, 568, 568*f,* 571–575
sample statements, 576–583
subqueries, 573–575
in multiple UNIONs, 350
predicates (filters) in (*See also* BETWEEN; comparison predicates; IN predicate; IS NULL predicate; LIKE)
efficiency of, 190–191
expressing in different ways, 197–198
INTERSECT, 234–236, 236*f*
multiple conditions, 178–197
NOT operator, 175–178, 176*f,* 184–187, 185*f*
nulls, evaluation of, 193–197, 194*f,* 195*f,* 196*f*
AND operator, 179, 180*f,* 183–184, 195–196, 204–205
order of precedence. *See* order of precedence
OR operator, 180–182, 181*f,* 184–185, 196, 196*t,* 203
AND and OR together, 183–184
overview, 152–154
sample statements, 198–206
in UPDATE statements, 508–514
in SELECT statement, 74*f,* 75
of subquery, 373, 380
syntax of, 152, 153*f*
in UPDATE statement, 502, 502*f,* 504–505, 506*f,* 507–508
subquery filters, 508–514
uses of
generally, 151–152
vs. HAVING, 478–485
Wong, Eugene, 56

X
X3, 57, 62
"X3.135-1992 Database Language SQL," 59–60
X3H2, 57
X/OPEN standard, 61