John L. Viescas

Foreword by **Keith W. Hare**
Vice Chair, USA SQL Standards Committee

SQL Queries for Mere Mortals

Fourth 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®

The good books show you how to do something. The great books enable you to think clearly about how you can do it. This book is the latter. To really maximize the potential of your database, thinking about data as a set is required and the authors' accessible writing really brings out the practical applications of SQL and the set-based thinking behind it.

— Ben Clothier, Lead Developer at IT Impact, Inc., co-author of *Professional Access 2013 Programming*, and Microsoft Access MVP

Unless you are working at a very advanced level, this is the only SQL book you will ever need. The author has 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

It's beyond brilliant! I have been working with SQL for a really long time, and the techniques presented in this book exposed some of the bad habits I picked up over the years in my learning process. I wish I had learned these techniques a long time ago and saved myself all the headaches of learning SQL the hard way. Who said you can't teach old dogs new tricks?

— Leo (theDBguy), Utter Access Moderator and Microsoft Access MVP

I learned SQL primarily from the first and second editions of this book…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 anymore, 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 must have remembered because he takes everyday English queries and literally translates 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, 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 does 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

I highly recommend *SQL Queries for Mere Mortals* to anyone working with data. John makes it easy to learn one of the most critical aspects of working with data: creating queries. Queries are the primary tool for selecting, sorting, and reporting data. They can compensate for table structure, new reporting requirements, and incorporate new data sources. *SQL Queries for Mere Mortals* uses clear, easy to understand discussions and examples to take readers through the basics and into complex problems. From novice to expert, you will find this book to be an invaluable reference as you can apply the concepts to a myriad of scenarios, regardless of the program.

— Teresa Hennig, Microsoft MVP-Access, and lead author of several Access books, including *Professional Access 2013 Programming* (Wrox)

SQL Queries *for* Mere Mortals®

Fourth Edition

A Hands-On Guide to Data Manipulation in SQL

John L. Viescas

Addison-Wesley

Boston ▪ Columbus ▪ Indianapolis ▪ New York ▪ San Francisco ▪ Amsterdam
Cape Town ▪ Dubai ▪ London ▪ Madrid ▪ Milan ▪ Munich ▪ Paris ▪ Montreal ▪ Toronto
Delhi ▪ Mexico City ▪ São Paulo ▪ Sydney ▪ Hong Kong ▪ Seoul ▪ Singapore ▪ Taipei ▪ Tokyo

Library of Congress Control Number: 2017964124

ISBN-13: 978-0-134-85833-3
ISBN-10: 0-134-85833-6

1 18

Editor-in-Chief: Mark Taub

Acquisitions Editor: Trina Macdonald

Development Editor: Rick Kughen

Managing Editor: Sandra Schroeder

Senior Project Editor: Lori Lyons

Production Manager: Dhayanidhi Karunanidhi

Copy Editor: Rick Kughen

Indexer: Lisa Stumpf

Proofreader: Abigail Manheim

Technical Reviewer: Douglas J. Steele

Cover Designer: Chuti Prasertsith

Compositor: codemantra

Contents at a Glance

Foreword xxi
Preface xxii
Acknowledgments xxiv
About the Author xxv
Introduction 1

PART I: RELATIONAL DATABASES AND SQL 13
Chapter 1: What Is Relational? 15
Chapter 2: Ensuring Your Database Structure Is Sound 33
Chapter 3: A Concise History of SQL 71

PART II: SQL BASICS 87
Chapter 4: Creating a Simple Query 89
Chapter 5: Getting More Than Simple Columns 125
Chapter 6: Filtering Your Data 175

PART III: WORKING WITH MULTIPLE TABLES 239
Chapter 7: Thinking in Sets 241
Chapter 8: INNER JOINs 273
Chapter 9: OUTER JOINs 321
Chapter 10: UNIONs 369
Chapter 11: Subqueries 399

PART IV: SUMMARIZING AND GROUPING DATA 441
Chapter 12: Simple Totals 443
Chapter 13: Grouping Data 471
Chapter 14: Filtering Grouped Data 505

PART V: MODIFYING SETS OF DATA 533
Chapter 15: Updating Sets of Data 535
Chapter 16: Inserting Sets of Data 573
Chapter 17: Deleting Sets of Data 603

PART VI: INTRODUCTION TO SOLVING TOUGH PROBLEMS 625
Chapter 18: "NOT" and "AND" Problems 627
Chapter 19: Condition Testing 677
Chapter 20: Using Unlinked Data and "Driver" Tables 709
Chapter 21: Performing Complex Calculations on Groups 749
Chapter 22: Partitioning Data into Windows 793
In Closing 857

PART VII: APPENDICES 859
Appendix A: SQL Standard Diagrams 861
Appendix B: Schema for the Sample Databases 877
Appendix C: Date and Time Types, Operations, and Functions 889
Appendix D: Suggested Reading 907

Index 909

Contents

Foreword xxi
Preface xxii
Acknowledgments xxiv
About the Author xxv

***Introduction* 1**
Are You a Mere Mortal? 1
About This Book 2
What This Book Is Not 4
How to Use This Book 4
Reading the Diagrams Used in This Book 5
Sample Databases Used in This Book 9
“Follow the Yellow Brick Road” 12

***PART I: RELATIONAL DATABASES AND SQL* 13**
***Chapter 1: What Is Relational?* 15**
Topics Covered in This Chapter 15
Types of Databases 15
A Brief History of the Relational Model 16
In the Beginning . . . 16
Relational Database Systems 17
Anatomy of a Relational Database 19
Tables 20
Columns 21
Rows 21
Keys 22
Views 23
Relationships 25
What’s in It for You? 29
Where Do You Go from Here? 30
Summary 31

Chapter 2: Ensuring Your Database Structure Is Sound 33

Topics Covered in This Chapter 33
Why Is this Chapter Here? 34
Why Worry about Sound Structures? 34
Fine-Tuning Columns 35
  What's in a Name? (Part One) 35
  Smoothing Out the Rough Edges 38
  Resolving Multipart Columns 40
  Resolving Multivalued Columns 43
Fine-Tuning Tables 46
  What's in a Name? (Part Two) 46
  Ensuring a Sound Structure 48
  Resolving Unnecessary Duplicate Columns 50
  Identification Is the Key 56
Establishing Solid Relationships 60
  Establishing a Deletion Rule 63
  Setting the Type of Participation 64
  Setting the Degree of Participation 66
Is That All? 69
Summary 69

Chapter 3: A Concise History of SQL 71

Topics Covered in This Chapter 71
The Origins of SQL 72
Early Vendor Implementations 73
". . . And Then There Was a Standard" 75
Evolution of the ANSI/ISO Standard 76
  Other SQL Standards 79
Commercial Implementations 83
What the Future Holds 83
Why Should You Learn SQL? 84
Which Version of SQL Does this Book Cover? 84
Summary 85

PART II: SQL BASICS 87

Chapter 4: Creating a Simple Query 89

Topics Covered in This Chapter 89
Introducing SELECT 90
The SELECT Statement 91
A Quick Aside: Data versus Information 93
Translating Your Request into SQL 95
  Expanding the Field of Vision 100
  Using a Shortcut to Request All Columns 101
Eliminating Duplicate Rows 103
Sorting Information 105
  First Things First: Collating Sequences 107
  Let's Now Come to Order 108
Saving Your Work 111
Sample Statements 113
Summary 122
Problems for You to Solve 123

Chapter 5: Getting More Than Simple Columns 125

Topics Covered in This Chapter 125
What Is an Expression? 126
What Type of Data Are You Trying to Express? 127
Changing Data Types: The CAST Function 130
Specifying Explicit Values 132
  Character String Literals 133
  Numeric Literals 135
  Datetime Literals 135
Types of Expressions 138
  Concatenation 138
  Mathematical Expressions 142
  Date and Time Arithmetic 146
Using Expressions in a SELECT Clause 150
  Working with a Concatenation Expression 151
  Naming the Expression 152
  Working with a Mathematical Expression 154

Working with a Date Expression 156
A Brief Digression: Value Expressions 157
That “Nothing” Value: Null 159
Introducing Null 160
The Problem with Nulls 162
Sample Statements 163
Summary 172
Problems for You to Solve 173

Chapter 6: Filtering Your Data 175
Topics Covered in This Chapter 175
Refining What You See Using WHERE 176
The WHERE Clause 176
Using a WHERE Clause 179
Defining Search Conditions 181
Comparison 181
Range 189
Set Membership 192
Pattern Match 194
Null 199
Excluding Rows with NOT 201
Using Multiple Conditions 204
Introducing AND and OR 205
Excluding Rows: Take Two 211
Order of Precedence 214
Checking for Overlapping Ranges 219
Nulls Revisited: A Cautionary Note 221
Expressing Conditions in Different Ways 225
Sample Statements 226
Summary 234
Problems for You to Solve 235

PART III: WORKING WITH MULTIPLE TABLES 239
Chapter 7: Thinking in Sets 241
Topics Covered in This Chapter 241
What Is a Set, Anyway? 242
Operations on Sets 243

Intersection 244
Intersection in Set Theory 244
Intersection between Result Sets 246
Problems You Can Solve with an Intersection 249
Difference 250
Difference in Set Theory 250
Difference between Result Sets 252
Problems You Can Solve with Difference 256
Union 257
Union in Set Theory 257
Combining Result Sets Using a Union 259
Problems You Can Solve with Union 261
SQL Set Operations 262
Classic Set Operations versus SQL 262
Finding Common Values: INTERSECT 262
Finding Missing Values: EXCEPT (DIFFERENCE) 265
Combining Sets: UNION 268
Summary 271

Chapter 8: INNER JOINs 273
Topics Covered in This Chapter 273
What Is a JOIN? 273
The INNER JOIN 274
What's "Legal" to JOIN? 275
Column References 275
Syntax 276
Check Those Relationships! 291
Uses for INNER JOINs 293
Find Related Rows 293
Find Matching Values 293
Sample Statements 294
Two Tables 295
More Than Two Tables 300
Looking for Matching Values 306
Summary 316
Problems for You to Solve 316

Chapter 9: OUTER JOINs 321
Topics Covered in This Chapter 321
What Is an OUTER JOIN? 321
The LEFT/RIGHT OUTER JOIN 323
Syntax 324
The FULL OUTER JOIN 344
Syntax 344
FULL OUTER JOIN on Non-Key Values 347
UNION JOIN 348
Uses for OUTER JOINs 349
Find Missing Values 349
Find Partially Matched Information 349
Sample Statements 350
Summary 365
Problems for You to Solve 366

Chapter 10: UNIONs 369
Topics Covered in This Chapter 369
What Is a UNION? 369
Writing Requests with UNION 372
Using Simple SELECT Statements 372
Combining Complex SELECT Statements 375
Using UNION More Than Once 379
Sorting a UNION 381
Uses for UNION 383
Sample Statements 385
Summary 395
Problems for You to Solve 396

Chapter 11: Subqueries 399
Topics Covered in This Chapter 399
What Is a Subquery? 400
Row Subqueries 400
Table Subqueries 402
Scalar Subqueries 402

Subqueries as Column Expressions 402
Syntax 402
An Introduction to Aggregate Functions: COUNT and MAX 406
Subqueries as Filters 408
Syntax 408
Special Predicate Keywords for Subqueries 411
Uses for Subqueries 422
Build Subqueries as Column Expressions 422
Use Subqueries as Filters 423
Sample Statements 424
Subqueries in Expressions 425
Subqueries in Filters 430
Summary 437
Problems for You to Solve 438

PART IV: SUMMARIZING AND GROUPING DATA 441
Chapter 12: Simple Totals 443
Topics Covered in This Chapter 443
Aggregate Functions 444
Counting Rows and Values with COUNT 446
Computing a Total with SUM 450
Calculating a Mean Value with AVG 451
Finding the Largest Value with MAX 452
Finding the Smallest Value with MIN 454
Using More Than One Function 455
Using Aggregate Functions in Filters 457
Sample Statements 459
Summary 466
Problems for You to Solve 467

Chapter 13: Grouping Data 471
Topics Covered in This Chapter 471
Why Group Data? 472
The GROUP BY Clause 475
Syntax 475
Mixing Columns and Expressions 481

Using GROUP BY in a Subquery in a WHERE Clause 483
Simulating a SELECT DISTINCT Statement 484
"Some Restrictions Apply" 485
Column Restrictions 486
Grouping on Expressions 488
Uses for GROUP BY 490
Sample Statements 491
Summary 501
Problems for You to Solve 501

Chapter 14: Filtering Grouped Data 505
Topics Covered in This Chapter 505
A New Meaning for "Focus Groups" 506
Where You Filter Makes a Difference 510
Should You Filter in WHERE or in HAVING? 510
Avoiding the HAVING COUNT Trap 513
Uses for HAVING 518
Sample Statements 519
Summary 527
Problems for You to Solve 528

PART V: MODIFYING SETS OF DATA 533
Chapter 15: Updating Sets of Data 535
Topics Covered in This Chapter 535
What Is an UPDATE? 536
The UPDATE Statement 536
Using a Simple UPDATE Expression 537
A Brief Aside: Transactions 540
Updating Multiple Columns 541
Using a Subquery to Filter Rows 543
Some Database Systems Allow a JOIN in the UPDATE Clause 546
Using a Subquery UPDATE Expression 548
Uses for UPDATE 551
Sample Statements 552
Summary 569
Problems for You to Solve 569

Chapter 16: Inserting Sets of Data 573
Topics Covered in This Chapter 573
What Is an INSERT? 573
The INSERT Statement 575
Inserting Values 575
Generating the Next Primary Key Value 578
Inserting Data by Using SELECT 581
Uses for INSERT 587
Sample Statements 588
Summary 598
Problems for You to Solve 598

Chapter 17: Deleting Sets of Data 603
Topics Covered in This Chapter 603
What Is a DELETE? 603
The DELETE Statement 604
Deleting All Rows 605
Deleting Some Rows 607
Uses for DELETE 611
Sample Statements 612
Summary 620
Problems for You to Solve 621

PART VI: INTRODUCTION TO SOLVING TOUGH PROBLEMS 625
Chapter 18: "NOT" and "AND" Problems 627
Topics Covered in This Chapter 627
A Short Review of Sets 628
Sets with Multiple AND Criteria 628
Sets with Multiple NOT Criteria 629
Sets Including Some Criteria but Excluding Others 630
Finding Out the "Not" Case 632
Using OUTER JOIN 632
Using NOT IN 635
Using NOT EXISTS 637
Using GROUP BY/HAVING 638

Finding Multiple Matches in the Same Table 641

Using INNER JOIN 642

Using IN 644

Using EXISTS 646

Using GROUP BY/HAVING 648

Sample Statements 652

Summary 671

Problems for You to Solve 672

Chapter 19: Condition Testing 677

Topics Covered in This Chapter 677

Conditional Expressions (CASE) 678

Why Use CASE? 678

Syntax 678

Solving Problems with CASE 683

Solving Problems with Simple CASE 683

Solving Problems with Searched CASE 688

Using CASE in a WHERE Clause 691

Sample Statements 692

Summary 705

Problems for You to Solve 706

Chapter 20: Using Unlinked Data and "Driver" Tables 709

Topics Covered in This Chapter 709

What Is Unlinked Data? 710

Deciding When to Use a CROSS JOIN 713

Solving Problems with Unlinked Data 714

Solving Problems Using "Driver" Tables 717

Setting Up a Driver Table 717

Using a Driver Table 720

Sample Statements 725

Examples Using Unlinked Tables 726

Examples Using Driver Tables 736

Summary 743

Problems for You to Solve 744

Chapter 21: Performing Complex Calculations on Groups 749
Topics in this Chapter 749
Grouping in Sub-Groups 750
Extending the GROUP BY Clause 753
Syntax 753
Getting Totals in a Hierarchy Using Rollup 754
Calculating Totals on Combinations Using CUBE 765
Creating a Union of Totals with GROUPING SETS 771
Variations on Grouping Techniques 775
Sample Statements 780
Examples using ROLLUP 781
Examples using CUBE 783
Examples using GROUPING SETS 786
Summary 788
Problems for You to Solve 789

Chapter 22: Partitioning Data into Windows 793
Topics in this Chapter 793
What You Can Do With a "Window" into Your Data 794
Syntax 798
Calculating a Row Number 814
Ranking Data 818
Splitting Data into Quintiles 824
Using Windows with Aggregate Functions 827
Sample Statements 834
Examples Using ROW_NUMBER 835
Examples Using RANK, DENSE_RANK, and PERCENT_RANK 838
Examples Using NTILE 842
Examples Using Aggregate Functions 844
Summary 852
Problems for You to Solve 853

In Closing 857

PART VII: APPENDICES 859

Appendix A: SQL Standard Diagrams 861

Appendix B: Schema for the Sample Databases 877

Sales Orders Example Database 878

Sales Orders Modify Database 879

Entertainment Agency Example Database 880

Entertainment Agency Modify Database 881

School Scheduling Example Database 882

School Scheduling Modify Database 883

Bowling League Example Database 884

Bowling League Modify Database 885

Recipes Database 886

"Driver" Tables 887

Appendix C: Date and Time Types, Operations, and Functions 889

IBM DB2 889

Microsoft Access 893

Microsoft SQL Server 895

MySQL 897

Oracle 901

PostgreSQL 904

Appendix D: Suggested Reading 907

Database Books 907

Books on SQL 908

Index 909

Foreword

In the 30 years since the database language SQL was adopted as an international standard, and the 30 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, Fourth 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 DM32.2
—the USA SQL Standards Committee; Convenor, ISO/IEC JTC1 SC32 WG3
—the International SQL Standards Committee

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 ERNEST 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 my many years as a database consultant, I'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, I decided it was time to write a book that would help people learn how to query a database properly and effectively. I, along with my good friend, Michael J. Hernandez, produced the first edition of this book in 2000. We created a second edition in 2008 that introduced basic ways to change data in your database using SQL. With the third edition in 2014, we stepped lightly into the realm of tougher problems—the sorts of problems that make the heads of even experienced users spin around three times. In this fourth edition, I expand your knowledge of tougher problems by covering Window functions and

Grouping Sets. The result of my effort 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 fourth edition includes dozens of new examples, and I included versions of the sample databases using Microsoft Office Access, Microsoft SQL Server, and the popular open-source MySQL and PostgreSQL database systems. 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 I need it the most. These people continually provide me with encouragement, help me to remain focused, and motivate me to see this project through to the end.

First and foremost, I want to thank my acquisitions editor, Trina MacDonald, for helping me get signed up to produce this fourth edition. Thanks also to Developmental Editor, Rick Kughen, for shepherding me along the way. And I can't forget the production staff—they're a great team! Next, I'd like to acknowledge my technical editor, Doug Steele. I also had help from one of my database friends, Ben Clothier. Thanks once again to all of you for your time and input and for helping me 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. I have a lot of respect for Keith's knowledge and expertise on the subject, and I'm pleased to have his thoughts and comments at the beginning of my book.

About the Author

John L. Viescas is an independent database consultant with more than 50 years of experience. He began his career as a systems analyst, designing large database applications for IBM mainframe systems. He spent 6 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 Tandem's U.S. Western Sales region. He developed and delivered technical seminars on Tandem's 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, IBM's DB2, Microsoft's SQL Server, Oracle Corporation's Oracle, and Tandem's 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*, three editions of *Microsoft Office Access Inside Out* (Microsoft Press, 2003, 2007, and 2010—the successor to the *Running* series), *Building Microsoft Access Applications* (Microsoft Press, 2005), and *Effective SQL* (Addison-Wesley, 2017).

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 was recognized as a "Most Valuable Professional" (MVP) from 1993 to 2015 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 John's Web site at www.viescas.com or contact him by e-mail at john@viescas.com.

Reader Services

Register your copy of *SQL Queries for Mere Mortals* on the InformIT site for convenient access to updates and corrections as they become available. To start the registration process, go to informit.com/register and log in or create an account. Enter the product ISBN **9780134858333** and click Submit. Look on the Registered Products tab for an Access Bonus Content link next to this product, and follow that link to access any available bonus materials. If you would like to be notified of exclusive offers on new editions and updates, please check the box to receive email from us.

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, I'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 I'm going to show you how to solve will involve getting data from *multiple* tables. I'm not only going to show you how to put together a good stew—I'm 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. I'm 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, I'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 I'm 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? Maybe you're not old enough!) 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 rows that are made up of one or more 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, I'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, I 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.

To 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: "List all students and the classes for which they are currently enrolled." "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."* "List the names and addresses *for both staff and students."*

In the following three sections, I'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 my 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 my friend 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.

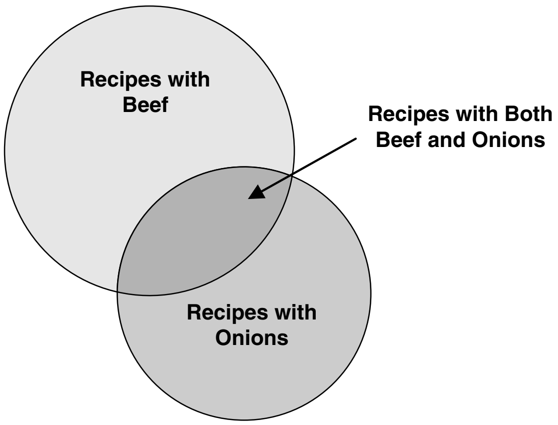

Figure 7-1 *Finding out which recipes have both beef and onions*

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, I 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	`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 first to find all the beef rows, then find all the onions rows, and then intersect them on RecipeID. (If you're confused about why I'm 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.

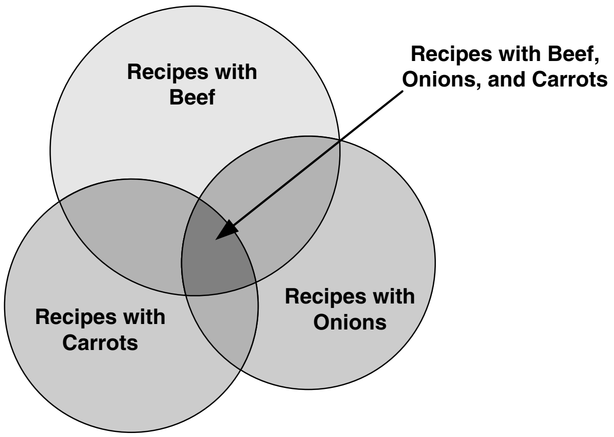

Figure 7-2 *Determining which recipes have beef, onions, and carrots*

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 I'll show you all about solving these problems (and more) by using JOINs. And because so few commercial implementations of SQL support INTERSECT, I'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 I 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, I 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.

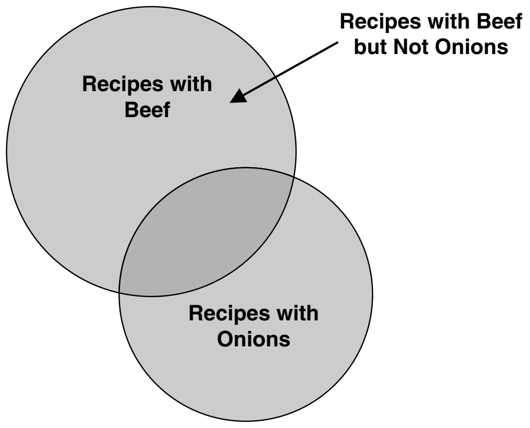

Figure 7-3 *Finding out which recipes have beef but not onions*

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 first to 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.

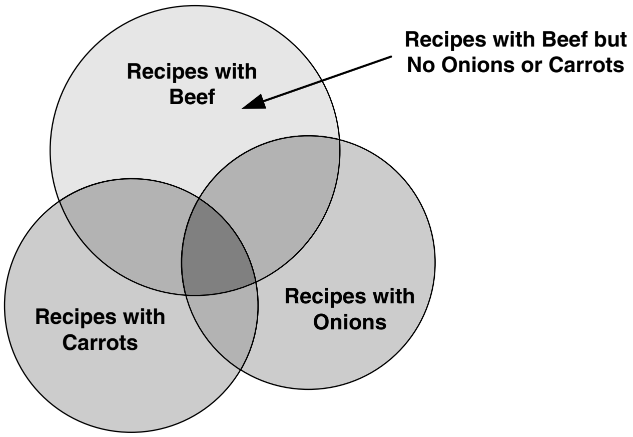

Figure 7-4 *Finding out which recipes have beef but no onions or carrots*

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 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! I'll show you all about solving these problems (and more) using OUTER JOINs in Chapter 9, "OUTER JOINs." Also, because only a few commercial implementations of SQL support EXCEPT (the keyword for difference), I'll show how to use an OUTER JOIN to solve many problems that might otherwise require an EXCEPT. In Chapter 18, "'NOT' and 'AND' Problems," I'll show you additional ways to solve EXCEPT problems.

Union

So far I'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 shows you the set diagram that helps you visualize how to solve this problem.

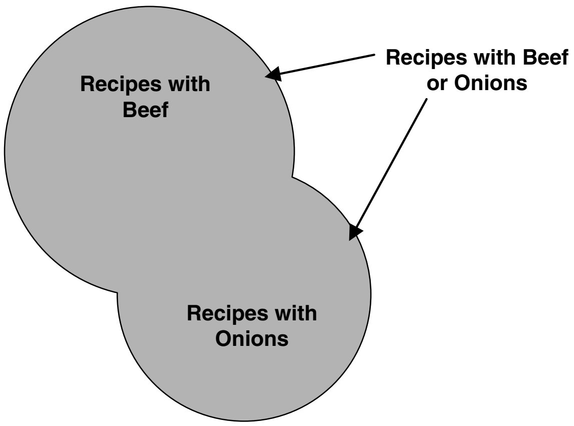

Figure 7-5 *Finding out which recipes have either beef or onions*

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 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. I'll explore the uses of the SQL UNION operator in detail in Chapter 10, "UNIONs."

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. I'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. I think that these set operations are important enough to at least warrant an overview of the syntax.

As promised, I'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 I didn't JOIN Order_Details to Products and look for bike or helmet product names. The simple answer is that I haven't introduced the concept of a JOIN yet, so I 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.

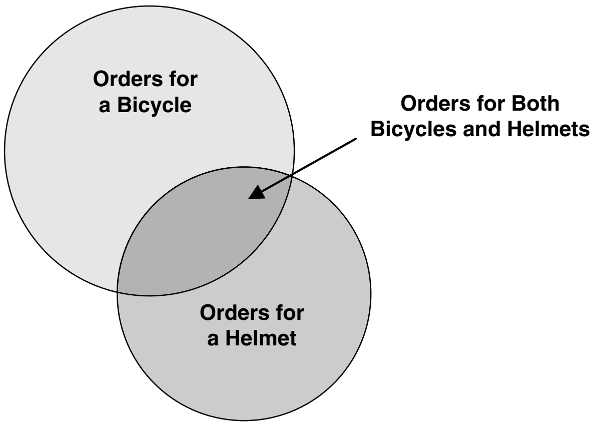

Figure 7-6 *One possible relationship between two sets of orders*

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.)

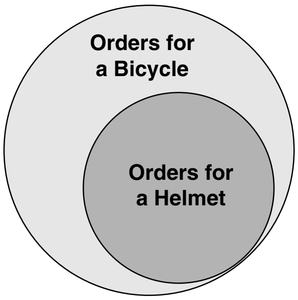

Figure 7-7 *All orders for a helmet also contain an order for a bicycle*

"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—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.)

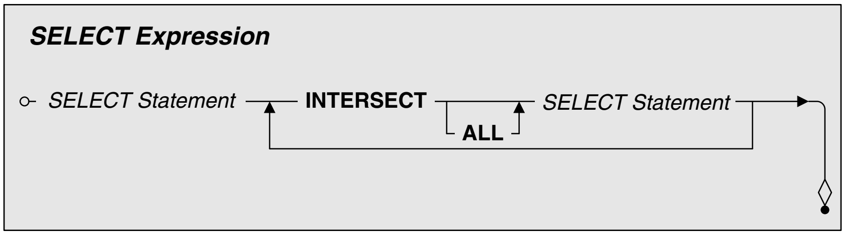

Figure 7-8 *Linking two SELECT statements with INTERSECT*

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.) I'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 virtually all 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)`

```
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.

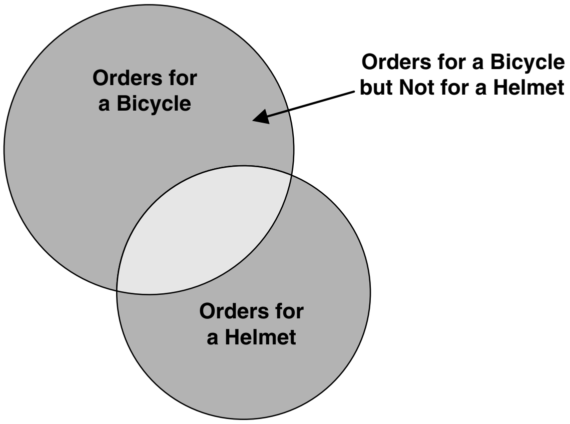

Figure 7-9 *Orders for a bicycle that do not also contain a helmet*

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.

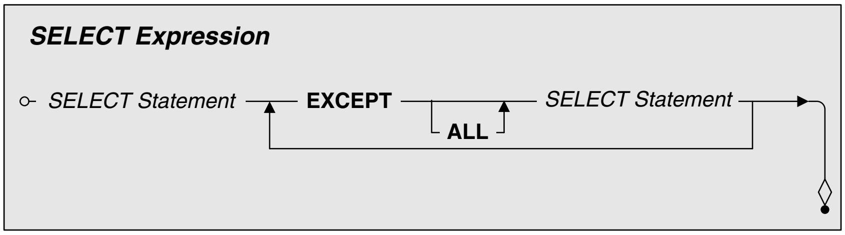

Figure 7-10 *Linking two SELECT statements with EXCEPT*

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 my 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.) I'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 nearly all commercial implementations of SQL *do* support OUTER JOIN.

Combining Sets: UNION

One more problem about bicycles and helmets, then I'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 I 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 I haven't shown you how to do a JOIN yet, I'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.

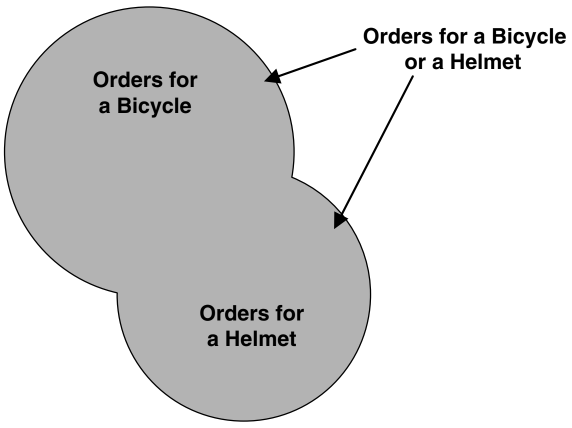

Figure 7-11 *Orders for bicycles or helmets*

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.

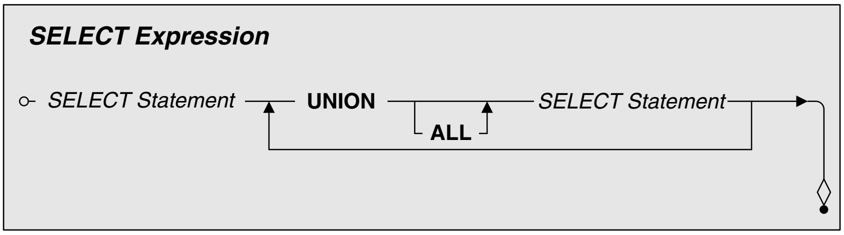

Figure 7-12 *Linking two SELECT statements with UNION*

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 nearly all 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. I'll explore UNION in much more detail in Chapter 10.

Summary

I began this chapter by discussing the concept of a set. Next, I discussed each of the major set operations implemented in SQL in detail—intersection, difference, and union. I showed how to use set diagrams to visualize the problem you're trying to solve. Finally, I 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. I showed you the SQL Standard EXCEPT operation, but an OUTER JOIN might also do the trick. Get the details in Chapters 9 and 18.

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

Index

Symbols

* (asterisk), 195
|| (concatenation operator), 138
> (greater than), 186–188
>= (greater than or equal to), 186
< (less than), 186–188
<= (less than or equal to), 186
() (parentheses), 144, 210–211, 216
  CASE (conditional expressions), 684
% (percent sign), 195
? (question mark), 195
_ (underscore), 195

Numbers

2016 SQL Standard, 4

A

ABS, 142
Access, CASE (conditional expressions), 684
Actian, 74
adding sorting specifications to UNION, 382
aggregate expressions, 445
aggregate functions, 444–446, 798–799
  AVG, calculating mean values, 451–452
  Bowling League Database, 844–846
  COUNT, 406–408, 446
  COUNT (*value expression*), 448–449
  COUNT(*), 446
    counting all the rows, 446–448
  in filters, 457–459
  grouping data, 477–478
  MAX, 406–408
    finding largest values, 452–454
  MIN, finding smallest values, 454–455
  Null values, 445
  OVER(), 803–805
  Sales Orders Database, 847–850
  sample statements, 459–466
    Bowling League Database, 463–465
    Entertainment Agency Database, 461–462
    Recipes Database, 465–466
    Sales Orders Database, 460–461
    School Scheduling Database, 462–463
  School Scheduling Database, 851–852
  subqueries, 457–458
  SUM, computing totals, 450–451
  syntax diagrams, 444
  using more than one function, 455–457
  windows functions, 827–834
alias names, assigning to tables, INNER JOIN, 282–284
ALL, 371, 375, 417–420
all rows, deleting with DELETE statement, 605–607
alphabetical order, sorting by, 108
American National Standards Institute (ANSI), 2, 75
  evolution of SQL/86, 76–79

analytical databases, 16
AND, 205–206
  finding multiple matches in the same table
    EXISTS, 646–648
    GROUP BY, 648–652
    HAVING, 648–652
    IN, 644–646
    INNER JOIN, 642–644
  sample statements, 652–657
    Bowling League Database, 663–668
    Entertainment Agency Database, 657–659
    Recipes Database, 668–671
    Sales Orders Database, 653–657
    School Scheduling Database, 659–663
  sets with multiple AND criteria, 628–629
  using with OR, 209–211
ANSI (American National Standards Institute), 2, 75
  evolution of SQL/86, 76–79
ANSI NCITS-H2, 80
ANSI/ISO standards, 76–79
ANY, 417–420
APPROXIMATE NUMERIC, 129
arithmetic operations
  IBM DB2, 890–908
  Microsoft Access, 893
  Microsoft SQL Server, 895
  MySQL, 898
  Oracle, 902
  PostgreSQL, 904
artificial primary keys, 60
assigning correlation names to tables, INNER JOIN, 282–284
asterisks (*), 195
  shortcuts, 102–103
ASYMMETRIC, 189–190
attributes. *See* columns
AVG, calculating mean values, 451–452
avoiding HAVING COUNT trap, 513–518

B

base tables, views (relational databases), 23
BETWEEN comparisons, 189–190
BETWEEN predicate, 178, 189–192
  checking for overlapping ranges, 219–220
BETWEEN...AND, 189
BINARY, 128
blank spaces, 160
BOOLEAN, 129
Bowling League Database
  aggregate functions, 463–465, 844–846
  AND, 663–668
  CASE (conditional expressions), 702–705
  CUBE, 783–784
  DELETE statement, 618–620
  driver tables, 742–743
  expressions, 169–171
  GROUP BY, 496–499
  GROUPING SETS, 786–787
  HAVING clause, 525–526
  INNER JOIN
    looking for matching values, 311–314
    more than two tables, 302–305
    two tables, 298–299
  INSERT statement, 596–597
  NOT, 663–668
  NTILE(), 842–843
  OUTER JOIN, 359–361
  search conditions, 231–232
  SELECT statement, 119–120
  subqueries
    in expressions, 428–429
    in filters, 434–435
  UNION, 392–394
  unlinked tables, 734–735
  UPDATE statement, 565–568

C

calculated columns, 40
calculating
  mean values with AVG, 451–452
  row number with ROW_NUMBER, 814–818

totals on combinations using CUBE, 765–770
Call-Level Interface (CLI), 80
Cartesian product, 277–278, 325, 711–712
cascade deletion rule, 63
CASE (conditional expressions), 678
  parentheses, 684
  reasons for using, 678
  sample statements, 692–693
    Bowling League Database, 702–705
    Entertainment Agency Database, 696–698
    Sales Orders Database, 693–696
    School Scheduling Database, 698–702
  searched CASE, solving problems, 688–691
  simple CASE, solving problems, 683–687
  syntax, 678–682
  WHERE clause, 691
CASE expression, syntax diagrams, 679
case sensitivity, 183
  string comparison, 197
CAST function
  changing data types, 130–132
  concatenation expressions, 141
CEIL, 142
Chamberlin, Dr. Donald, 72
changing, data types, CAST function, 130–132
CHARACTER, 127
character string literals, 133–134
checking, relationships, INNER JOIN, 291–292
classic set operations versus SQL, 259–262
  EXCEPT, 265–268
  INTERSECT, 262–265
  UNION, 268–271
clauses
  FROM clause, 279–280, 343
    embedding SELECT statements, 286
    OUTER JOIN, 324–325, 327
  CORRESPONDING clause, 373
  HAVING. *See* HAVING clause
  ORDER BY clause. *See* ORDER BY clause
  SELECT statement, 91, 92–93
  USING clause, 280–281
  VALUES clause, INSERT statement, 575–578
  WHERE clause. *See* WHERE clause
CLI (Call-Level Interface), 80
cloud servers, 83
Codd, Dr. Edgar F.16–17, 72
collating sequences, 107
column expressions, subqueries, 402–405, 422–423
column references, 275–276
  syntax diagrams, 276
column restrictions, GROUP BY, 486–488
columns
  calculated columns, 40
  counting values in, 448–449
  fine-tuning, 35
    naming conventions, 35–38
    structure of columns, 38–40
  mixing with expressions, 481–483
  multipart columns, 39
    resolving, 40–43
  multiple columns, updating, 541–543
  multivalued columns, 39
    resolving, 43–45
  relational databases, 21
  requesting all columns with SELECT statement, 101–103
  resolving duplicate columns, 50–56
  retrieving multiple columns with SELECT statement, 100–101
combining
  sets, UNION, 257–259, 268–271
  UNION. *See* UNION
commercial implementations of SQL, 83
COMMIT, 540
common values, finding with INTERSECT, 262–265
comparing string values, 181–184
comparison predicate, 178, 181
  aggregate functions as filters, 457–458
  comparing string values, 181–184

equality and inequality, 185–186
less than and greater than, 186–188
composite primary keys, 57
Computer Associates International, Inc., 74
computing totals with SUM, 450–451
concatenation expressions, 138–141
CAST function, 141
syntax diagrams, 138
conditional expressions. *See* CASE (conditional expressions)
conditions
expressing, 225–226
multiple conditions, 204–205
converting values into data types, CAST function, 130–132
correlation names, assigning to tables (INNER JOIN), 282–284
CORRESPONDING clause, 373
COUNT, 406–408, 445–446
counting all the rows, 446–448
counting values in columns or expressions, 448–449
HAVING clause, 513–518
COUNT (*value expression*), 446, 448–449
COUNT(*), 827–828
counting all the rows, 446–448
counting
all the rows, COUNT(*), 446–448
values in columns or expressions, 448–449
CROSS JOIN, 713
deciding when to use, 713–714
sample statements, 725–726
unlinked tables, 711–712, 715–716
CUBE, 778–779
Bowling League Database, 783–784
calculating totals on combinations, 765–770
Sales Orders Database, 784–786

D

data
fetching from two tables
with JOIN, 370
with UNION, 371
grouping, 472–474
with aggregate functions, 477–478
with GROUP BY. See GROUP BY
mixing columns and expressions, 481–483
versus information, 93–95
inserting
with INSERT statements. See INSERT statement
with SELECT expressions, 581–587
ranking, with RANK(), 818–824
splitting into quintiles, 824–827
unlinked data. *See* unlinked data
data types
APPROXIMATE NUMERIC, 129
BINARY, 128
BOOLEAN, 129
changing with CAST function, 130–132
CHARACTER, 127
DATETIME, 129
EXACT NUMERIC, 128
extended data types, 130
for IBM DB2, 857, 889
INTERVAL, 129, 147–148
JOIN eligible data types, 275
for Microsoft Access, 893
for Microsoft SQL Server, 895
MySQL, 897
NATIONAL CHARACTER, 128
Oracle, 901
PostgreSQL, 904
data warehouses, 19
database design, 30–31, 34–35
database structures, 34–35
database theory, 30–31
databases
analytical databases, 16
operational databases, 16
relational database model, 16–17
relational databases. *See* relational databases
sample databases, 9–11
types of, 15–16
DATE, 136, 138
Date, C. J., 76
date and time arithmetic expressions, 138, 146

date expressions, 147–148
time expressions, 149–150
date expressions, 147–148, 156–157
SELECT statement, time expressions, 156
syntax diagrams, 147
DATETIME, 129
datetime literals, 135–138
DB2, 74
DEFAULT keyword, 576
default pad character, 184
degree of participation, relationships, 66–68
DELETE statement, 604–605
deleting
all rows, 605–607
some rows, 607–611
sample statements, 612–613
Bowling League Database, 618–620
Entertainment Agency Database, 614–617
Sales Orders Database, 613–614
School Scheduling Database, 617–618
syntax diagrams, 604
uses for, 611–612
VALUES clause, 603–604
WHERE clause, 603–604
deleting. *See also* resolving
all rows with DELETE statement, 605–607
duplicate rows with SELECT statement, 50–56
some rows with DELETE statement, 607–611
deletion rule, relationships, 63–64
delimited identifiers, 37, 48
DENSE_RANK(), 820, 823–824
derived tables, 284
descending order, sorting by, 109
design, database design, 30–31, 34–35
diagrams. *See also* Appendix A; syntax diagrams
predicates, 681–682
Search Condition, 680
SELECT queries, 106
SELECT statement, 92
utilizing, 5–9
difference, 243, 250
problems you can solve with, 256–257
result sets, 252–256
set theory, 250–252
DISTINCT, 103–105, 415, 576
GROUP BY, 485
MAX, 454
MIN, 455
UNION, 389
driver tables, 717
Bowling League Database, 742–743
Entertainment Agency Database, 737–739
Sales Orders Database, 736–737
School Scheduling Database, 739–742
setting up, 717–720
using, 720–725
dynamic data, 16

E

eliminating. *See* deleting
embedding
INNER JOIN in SELECT statements, 284–286
JOINs within JOINs, 286–291
OUTER JOIN, 333–344
SELECT statements in OUTER JOINs, 330–333
Entertainment Agency Database
aggregate functions, 461–462
AND, 657–659
CASE (conditional expressions), 696–698
DELETE statement, 614–617
driver tables, 737–739
expressions, 165–167
GROUP BY, 493–494
GROUPING SETS, 787–788
HAVING clause, 521–522
INNER JOIN
looking for matching values, 309–311
more than two tables, 301–302
two tables, 296–297

INSERT statement, 593–595
NOT, 657–659
NTILE(), 843–844
OUTER JOIN, 354–355
ROW_NUMBER(), 835–837
search conditions, 228–230
SELECT statement, 116–117
subqueries
in expressions, 426–427
in filters, 432–433
UNION, 389–390
unlinked tables, 728–731
UPDATE statement, 558–562
Entry SQL, 78
equality, comparison predicate, 185–186
ESCAPE option, LIKE predicate, 198–199
Euler, Leonard, 246–247
Euler diagram, 246–247
events, relational databases, 20
evolution of SQL/86, 76–79
EXACT NUMERIC, 128
EXCEPT, 265–268
excluding rows with NOT, 201–204, 211–214
executing queries, 112
EXISTS, 420–422
AND, 654–655, 657–658
finding multiple matches in the same table, 646–648
EXP, 142
explicit values, specifying, 132–133
expressing conditions, 225–226
expressions, 126
aggregate expressions, 445
counting values in, 448–449
grouping with GROUP BY, 488–490
mixing with columns, 481–483
sample statements, 163
Bowling League Database, 169–171
Entertainment Agency Database, 165–167
Sales Orders Database, 164–165
School Scheduling Database, 167–169
SELECT clause, 150–151
concatenation expressions, 151–152
naming, 152–154
SELECT expression, 372
SELECT statement
date expressions, 156–157
mathematical expressions, 154–156
subqueries, sample statements, 425–430
types of
data, 127–130
concatenation expressions, 138–141
date and time arithmetic expressions, 138, 146–150
mathematical expressions, 138, 142–146
UPDATE expressions, subqueries, 548–551
value expressions, 157–159
syntax diagrams, 549
extended data types, 130
eXtensible Markup Language (XML), 19
extensions, SQL/92, 79

F

Federal Information Processing Standard (FIPS), 80
fetching data
from two tables with JOIN, 370
from two tables with UNION, 371
fields. *See* columns
filter rows, subqueries, 543–546
filtering
deciding between WHERE and HAVING, 510–513
grouped data, 506–510
filters
aggregate functions, 457–459
subqueries, 408–411
subqueries as, 423–424
sample statements, 430–437
finding
common values, INTERSECT, 262–265
largest values with MAX, 452–454

matching values with INNER JOINs, 293–294
sample statements, 306–316
missing values
EXCEPT, 265–268
OUTER JOIN, 349
multiple matches in the same table
EXISTS, 646–648
GROUP BY, 648–652
HAVING, 648–652
IN, 644–646
INNER JOIN, 642–644
partially matched information, OUTER JOIN, 349–350
related rows, INNER JOIN, 293
smallest values with MIN, 454–455
fine-tuning
columns, 35
naming conventions, 35–38
resolving multipart columns, 40–43
resolving multivalued columns, 43–45
structure of columns, 38–40
tables, 46
naming conventions, 46–48
relationships, 60–63
relationships, deletion rule, 63–64
relationships, participation, 64–68
resolving duplicate columns, 50–56
structure of tables, 48–50
FIPS (Federal Information Processing Standard), 80
FLOOR, 142
foreign keys, relational databases, 22–23
FROM clause, 279–280, 343
embedding, SELECT statements, 286
INNER JOIN, 277
OUTER JOIN, 324–325, 327
SELECT statement, 92, 331
FULL OUTER JOIN, 344–347
non-key values, 347–348
syntax, 344–347
syntax diagrams, 344
Full SQL, 79
functions
aggregate functions. *See* aggregate functions
CAST function, 130–132
IBM DB2, 890–892
Microsoft Access, 893–894
Microsoft SQL Server, 896–897
MySQL, 898–901
Oracle, 902–903
future of SQL (Structured Query Language), 83–84

G

generating primary key values with INSERT statements, 578–580
greater than (>), 186–188
greater than or equal to (>=), 186
GROUP BY, 475, 692, 725, 757, 761
column restrictions, 486–488
finding multiple matches in the same table, 648–652
grouping expressions, 488–490
mixing columns and expressions, 481–483
versus OVER(), 800
sample statements, 491–501
Bowling League Database, 496–499
Entertainment Agency Database, 493–494
Recipes Database, 499–500
Sales Orders Database, 491–492
School Scheduling Database, 495–496
SELECT statement, 93
sets with multiple NOT criteria, 638–641
simulating SELECT DISTINCT statements, 484–485
in subqueries in WHERE clauses, 483–484
syntax, 475–481, 753
syntax diagrams, 475
uses for, 490–491
grouped data, filtering, 506–510
grouping
calculating totals on combinations with CUBE, 765–770
creating a union of totals with GROUPING SETS, 771–775
CUBE, 778–779

data, 472–474
aggregate functions, 477–478
GROUP BY. See GROUP BY
mixing columns and expressions, 481–483
expressions, with GROUP BY, 488–490
GROUPING SETS, 775–776
ROLLUP, 777–778
sample statements, 780–781
CUBE, 783–786
GROUPING SETS, 786–788
ROLLUP, 781–783
in sub-groups, 750–753
totals in hierarchies, ROLLUP, 754–765
variations on techniques, 775–780
GROUPING function, 758
GROUPING SETS, 775–776
Bowling League Database, 786–787
creating union of totals, 771–775
Entertainment Agency Database, 787–788

H

HAVING
filtering, 510–513
grouped data, 508–510
finding multiple matches in the same table, 648–652
sample statements, 519–527
Bowling League Database, 525–526
Entertainment Agency Database, 521–522
Recipes Database, 526–527
Sales Orders Database, 520–521
School Scheduling Database, 522–525
SELECT statement, 93
sets with multiple NOT criteria, 638–641
uses for, 518–519
HAVING COUNT trap, avoiding, 513–518
Hernandez, Mike, 17
hierarchies, totals, ROLLUP, 754–765
history of SQL (Structured Query Language)
commercial implementations, 83
early vendor implementations, 73–74
origins of, 72–73
standards, 75–76

I

IBM, 18, 19, 74
early vendor implementations, 73–74
System R, 72, 73
IBM DB2
arithmetic operations, 890–908
data types, 857, 889
functions, 890–892
Identity data type, 579
identifiers
delimited identifiers, 37, 48
regular identifiers, 37, 48
Identity data type, 579
IN, finding multiple matches in the same table, 644–646
IN predicate, 178
membership condition, 192–194
for subqueries, 411–417
INCITS DM32.2, 80
inequality, comparison predicate, 185–186
information
versus data, 93–95
sorting with SELECT statement, 105–107
Ingres, 74
INGRES (Interactive Graphics Retrieval System), 18
Inmon, William H., 19
INNER JOIN, 274, 713
assigning correlation names to tables, 282–284
FROM clause, 277
embedding
JOINs within JOINs, 286–291
SELECT statements, 284–286
finding
matching values, 293–294
multiple matches in the same table, 642–644
related rows, 293

relationships, checking, 291–292
sample statements, 295
looking for matching values, 306–316
more than two tables, 300–306
two tables, 295–300
syntax, 276
syntax diagrams, 277
tables, 277–282
INSERT INTO, 576
INSERT statement, 573–575
generating primary key values, 578–580
inserting values, 575–578
sample statements, 588–589
Bowling League Database, 596–597
Entertainment Agency Database, 593–595
Sales Orders Database, 589–592
School Scheduling Database, 595–596
SELECT expression, 581
syntax diagrams with VALUES clause, 575
uses for, 587–588
inserting
data
with INSERT statement. See INSERT statement
with SELECT expressions, 581–587
values, with INNER JOINs, 575–578
Interactive Graphics Retrieval System (INGRES), 18
Intermediate SQL, 78
International Organization for Standardization. *See* ISO (International Organization for Standardization)
INTERSECT, 262–265
intersection, 243, 244
problems you can solve with, 249–250
result sets, 246–249
set theory, 244–245
INTERVAL, 129, 147–148
IS NULL, 634–635
IS NULL predicate, 178
ISO (International Organization for Standardization), 2, 76
evolution of SQL/86, 76–79

J

JOIN, 273–274
determining what is legal to join, 275
embedding within JOINs, 286–291
OUTER JOIN, 333–344
fetching data from two tables, 370
INNER JOIN. *See* INNER JOIN
NATURAL JOIN, 329–330
OUTER JOIN. *See* OUTER JOIN
UNION JOIN, 348
UPDATE clause, 546–548
JOIN eligible data types, 275

K

keys
primary keys, tables, 56–60
relational databases, 22–23
keywords
DATE, 138
DEFAULT keyword, 576
DISTINCT, 103–105, 576
GROUP BY, 93
HAVING, 93
INSERT INTO, 576
special predicate keywords
ALL, 417–420
ANY, 417–420
EXISTS, 420–422
IN, 411–417
SOME, 417–420
TIME, 138
TOP, 111
VALUES, 576
WHERE, 93

L

largest values, finding with MAX, 452–454
LEFT OUTER JOIN, 323–324, 332
less than (<), 186–188

less than or equal to (<=), 186
LIKE predicate, 178
  ESCAPE option, 198–199
  pattern match condition, 194–199
linking
  multiple tables, embedding JOINs within JOINs, 286–291
  SELECT statements with INTERSECT, 265
linking tables, 44–45
literal values, 132–133
  character string literals, 133–134
  datetime literals, 135–138
  numeric literals, 135
LN, 142

M

mandatory participation, 64
many-to-many relationships, 61–62
  relational databases, 27–29
matches, finding multiple matches in the same table
  EXISTS, 646–648
  GROUP BY, 648–652
  HAVING, 648–652
  IN, 644–646
  INNER JOIN, 642–644
matching values, finding with INNER JOINs, 293–294
  sample statements, 306–316
mathematical expressions, 138, 142–146
  Nulls, 163
  parentheses, 144
  SELECT statement, 154–156
  syntax diagrams, 143
MAX, 406–408
  finding largest values, 452–454
mean values, calculating with AVG, 451–452
membership condition, IN predicate, 192–194
mere mortals, 1–2
Microsoft Access, 19, 195
  arithmetic operations, 893
  data types, 893
  functions, 893–894
Microsoft Office Access, CASE (conditional expressions), 684
Microsoft SQL Server
  arithmetic operations, 895
  data types, 895
  functions, 896–897
  Identity data type, 579
  ORDER BY clause, 653, 757
MIN, finding smallest values, 454–455
missing values, 161
  finding with OUTER JOIN, 349
MOD, 142
multicolumn sorts, 109–110
multipart columns, 39
  resolving, 40–43
multiple columns, updating, 541–543
multiple conditions, 204–205
multiple tables, linking (embedding JOINs within JOINs), 286–291
multivalued columns, 39
  resolving, 43–45
MySQL
  arithmetic operations, 898
  data types, 897
  functions, 898–901
  ROLLUP, 765

N

names, qualifying in ON clause, 279–280
naming conventions
  for columns, 35–38
  for tables, 46–48
naming expressions (SELECT clause), 152–154
NATIONAL CHARACTER, 128
National Committee for Information Technology Standards (NCITS), 80
National Institute of Standards and Technology (NIST), 80
NATURAL JOIN, 281, 329–330
NCITS (National Committee for Information Technology Standards), 80

nested parenthetical operations, 144
NEXTVAL property, Oracle, 579
NIST (National Institute of Standards and Technology), 80
non-key values, FULL OUTER JOIN, 347–348
NOT
  excluding rows, 201–204, 211–214
  sample statements, 652–657
    Bowling League Database, 663–668
    Entertainment Agency Database, 657–659
    Recipes Database, 668–671
    Sales Orders Database, 653–657
    School Scheduling Database, 659–663
  sets with multiple NOT criteria, 629–630
    GROUP BY, 638–641
    HAVING, 638–641
    NOT EXISTS, 637–638
    NOT IN, 635–637
    OUTER JOIN, 632–635
  syntax diagrams, 202
NOT EXISTS, sets with multiple NOT criteria, 637–638
NOT IN, sets with multiple NOT criteria, 635–637
NTILE(), 824–827
  Bowling League Database, 842–843
  Entertainment Agency Database, 843–844
Null, 159–162, 199–201, 221–225
  aggregate functions, 445
  problems with, 162–163
Null condition, syntax diagrams, 199
numeric literals, 135

O

objects, relational databases, 20
ODBC (Open Database Connectivity), 80
ON clause
  qualifying names, 279–280
  search conditions, 278
one-to-many relationships, 61
  relational databases, 26
one-to-one relationships, 60–61
  relational databases, 25–26
Open Database Connectivity (ODBC), 80
operational databases, 16
operations, set operations, 243–244
operators
  AND, 205–206
    using with OR, 209–211
  NOT operator, excluding rows, 211–214
  OR, 206–209
    determining result sets, 224
    using with AND, 209–211
optional participation, 64
OR, 205, 206–209
  determining result sets, 224
  using with AND, 209–211
Oracle
  arithmetic operations, 902
  data types, 901
  functions, 902–903
  NEXTVAL property, 579
Oracle Corporation, 74
ORDER BY, 106–107, 474, 652–653, 692, 725, 757
  predicates, 805–810
  sorting SELECT queries, 108–111
  UNION, 382
order of precedence, 143
  search conditions, 214–215
    less is more, 217–218
    prioritizing, 215–217
origins of SQL (Structured Query Language), 72–73
orphaned rows, 63
OUTER JOIN, 321–323, 349, 713
  embedding
    JOINs within JOINs, 333–344
    SELECT statements, 330–333
  finding
    missing values, 349
    partially matched information, 349–350
  FULL OUTER JOIN. *See* FULL OUTER JOIN
  grouping data types, 478
  LEFT OUTER JOIN, 323–324
  RIGHT OUTER JOIN, 323–324

sample statements, 350–365
Bowling League Database, 359–361
Entertainment Agency Database, 354–355
Recipes Database, 362–364
Sales Orders Database, 351–353
School Scheduling Database, 356–359
sets with multiple NOT criteria, 632–635
syntax, 324–330
tables, 324–330
OVER(), 799, 811
aggregate functions, 803–805
versus GROUP BY, 800
predicates
ORDER BY, 805–810
PARTITION BY, 800–801
ROWS (or RANGE), 811–814
overlapping ranges, checking for, 219–221

P

parentheses, 210–211, 216
CASE expression, 684
mathematical expressions, 144
partially matched information, finding with OUTER JOIN, 349–350
participation, relationships, 64–68
PARTITION BY, 800–801
pattern match condition, 194–199
syntax diagrams, 195
pattern strings, samples, 195–196
percent sign (%), 195
PERCENT_RANK(), 820–821, 824
PostgreSQL, 904–905
arithmetic operations, 904
data types, 904
functions, 904–905
POWER, 142
predicates, 93, 177
BETWEEN . . . AND, 189
BETWEEN, 178
range condition, 189–192
combining with AND, 205–206
comparison, 178
comparison predicate, aggregate functions as filters, 457–458
diagrams, 681–682
IN, 178
membership condition, 192–194
IS NULL, 178
unknown values, 199–201
LIKE, 178
ESCAPE option, 198–199
pattern match condition, 194–199
OR, 206–209
ORDER BY, 805–810
PARTITION BY, 800–801
ROWS (or RANGE), 811–814
primary key values, generating with INSERT statements, 578–580
primary keys, 20
relational databases, 22–23
tables, 56–60
primary tables, one-to-one relationships, 25
prioritizing search conditions, order of precedence, 215–217
problems, solving
with searched CASE, 688–691
with simple CASE, 683–687
with unlinked data, 714–716

Q

qualifying names, ON clause, 279–280
quantified predicates, (SOME, ANY, ALL), 417–420
QUEL (Query Language), 74
queries
executing, 112
RDBMS (relational database management system), 24
SELECT queries. *See* SELECT query
query expression, 106–107
Query Language (QUEL), 74
query optimizers, 218
query specification, 106–107
question mark (?), 195
quintiles, splitting, data, 824–827
quotes, single quote, 133

R

RANGE, 811–814, 827–828
range condition, 189–192
ranges, overlapping ranges (checking for), 219–221
RANK(), 818–824
ranking data, with RANK(), 818–824
RDBMS (relational database management system), 17–19, 74
REAL, 146
reasons for learning SQL (Structured Query Language), 84
Recipes Database
  aggregate functions, 465–466
  AND, 668–671
  GROUP BY, 499–501
  HAVING clause, 526–527
  INNER JOIN
    looking for matching values, 314–315
    more than two tables, 305–306
    two tables, 299–300
  NOT, 668–671
  OUTER JOIN, 362–364
  ROW_NUMBER(), 837–838
  search conditions, 232–233
  SELECT statement, 120–121
  subqueries
    in expressions, 429–430
    in filters, 435–437
  UNION, 394–395
records. *See* rows
referential integrity, 62–63
refining searches, 176
regular identifiers, 37, 48
related rows, finding with INNER JOINs, 293
relational database management system (RDBMS), 17–19, 74
relational database model, 16–17
relational databases
  columns, 21
  keys, 22–23
  reasons for learning, 29–30
  relations, 19
  relationships, 25
    many-to-many, 27–29
    one-to-many, 26
    one-to-one, 25–26
  rows, 21–22
  tables, 20–21
  views, 23–24
Relational Software, Inc., 74
Relational Technology, Inc., 74
relations. *See* tables
relationships
  checking with INNER JOIN, 291–292
  deletion rule, 63–64
  many-to-many relationships, 61–62
  one-to-many relationships, 61
  one-to-one relationships, 60–61
  participation, 64–68
  relational databases, 25
    many-to-many, 27–29
    one-to-many, 26
    one-to-one, 25–26
  tables, 60–63
requests
  translating into SQL, SELECT statement, 95–99
  writing with UNION, 372
    combining complex SELECT statements, 375–379
    using simple SELECT statements, 372–375
resolving. *See also* deleting
  duplicate columns, 50–56
  multipart columns, 40–43
  multivalued columns, 43–45
restrict deletion rule, 63
restrictions, column restrictions (GROUP BY), 486–488
result sets
  determining with OR, 224
  difference, 252–256
  intersection, 246–249
  union, combining, 259–261
retrieving multiple columns, with SELECT statement, 100–101
RIGHT OUTER JOIN, 323–324
ROLLBACK, 540

ROLLUP, 777–778
MySQL, 765
Sales Orders Database, 781–782
sample statements, 781–782
School Scheduling Database, 782–783
totals in hierarchies, 754–765
row numbers, calculating with ROW_NUMBER, 814–818
row subqueries, 400–402
row value constructors, 401
ROW_NUMBER(), 814–818, 827–828, 829
Entertainment Agency Database, 835–837
Recipes Database, 837–838
rows
counting with COUNT(*), 446–448
deleting all rows with DELETE statement, 605–607
deleting some rows with DELETE statement, 607–611
eliminating duplicates with SELECT statement, 103–105
excluding, with NOT, 201–204, 211–214
filter rows, subqueries, 543–546
related rows, finding with INNER JOIN, 293
relational databases, 21–22
selected rows, updating, 631–633
ROWS (or RANGE), 811–814, 827–828

S

SAA (Systems Application Architecture), 80
Sales Orders Database
aggregate functions, 460–461, 847–850
AND, 653–657
CASE (conditional expressions), 693–696
CUBE, 784–786
DELETE statement, 613–614
driver tables, 736–737
expressions, 164–165
GROUP BY, 491–492
HAVING clause, 520–521
INNER JOIN
looking for matching values, 306–309
more than two tables, 300–301
two tables, 295–296
INSERT statement, 589–592
NOT, 653–657
OUTER JOIN, 351–353
ROLLUP, 781–782
search conditions, 227–228
SELECT statement, 114–115
subqueries
in expressions, 425–426
in filters, 430–431
UNION, 385–389
unlinked tables, 726–728
UPDATE statement, 554–557
windows functions, 838–839
sample databases, 9–11
sample statements
aggregate functions, 459–466
Bowling League Database, 463–465
Entertainment Agency Database, 461–462
Recipes Database, 465–466
Sales Orders Database, 460–461
School Scheduling Database, 462–463
AND, 652–657
Bowling League Database, 663–668
Entertainment Agency Database, 657–659
Recipes Database, 668–671
Sales Orders Database, 653–657
School Scheduling Database, 659–663
CASE (conditional expressions), 692–693
Bowling League Database, 702–705
Entertainment Agency Database, 696–698
Sales Orders Database, 693–696
School Scheduling Database, 698–702
CROSS JOIN, 725–726
DELETE statement, 612–613
Bowling League Database, 618–620
Entertainment Agency Database, 614–617

Sales Orders Database, 613–614
School Scheduling Database, 617–618
expressions, 163–171
Bowling League Database, 169–171
Entertainment Agency Database, 165–167
Sales Orders Database, 164–165
School Scheduling Database, 167–169
GROUP BY, 491–501
Bowling League Database, 496–499
Entertainment Agency Database, 493–494
Recipes Database, 499–500
Sales Orders Database, 491–492
School Scheduling Database, 495–496
grouping, 780–781
CUBE, 783–786
GROUPING SETS, 786–788
ROLLUP, 781–783
HAVING clause, 519–527
Bowling League Database, 525–526
Entertainment Agency Database, 521–522
Recipes Database, 526–527
Sales Orders Database, 520–521
School Scheduling Database, 522–525
INNER JOIN, 295
looking for matching values, 306–316
more than two tables, 300–306
two tables, 295–300
INSERT statement, 588–589
Bowling League Database, 596–597
Entertainment Agency Database, 593–595
Sales Orders Database, 589–592
School Scheduling Database, 595–596
NOT, 652–657
Bowling League Database, 663–668
Entertainment Agency Database, 657–659
Recipes Database, 668–671
Sales Orders Database, 653–657
School Scheduling Database, 659–663
OUTER JOIN, 350–365
Bowling League Database, 359–361
Entertainment Agency Database, 354–355
Recipes Database, 362–364
Sales Orders Database, 351–353
School Scheduling Database, 356–359
search conditions, 226–227
Bowling League Database, 231–232
Entertainment Agency Database, 228–229
Recipes Database, 232–233
Sales Orders Database, 227–228
School Scheduling Database, 230–231
SELECT queries, 113–121
SELECT statement, 113–121
Bowling League Database, 119–120
Entertainment Agency Database, 116–117
Recipes Database, 120–121
Sales Orders Database, 114–115
School Scheduling Database, 117–118
subqueries, 424–425
Bowling League Database, 428–429, 434–435
Entertainment Agency Database, 426–427, 432–433
in expressions, 425–430
in filters, 430–437
Recipes Database, 429–430, 435–437
Sales Orders Database, 425–426, 430–431
School Scheduling Database, 427–428, 433–434
UNION, 385–395
Bowling League Database, 392–394
Entertainment Agency Database, 389–390
Recipes Database, 394–395
Sales Orders Database, 385–389
School Scheduling Database, 390–392
unlinked data
driver tables, 736–743
unlinked tables, 726–736
UPDATE statement, 552–553

Bowling League Database, 565–568
Entertainment Agency Database, 558–562
Sales Orders Database, 554–557
School Scheduling Database, 562–565
windows functions, 834–835
aggregate functions, 844–852
NTILE(), 842–844
RANK(), DENSE_RANK, and PERCENT_RANK, 838–841
ROW_NUMBER(), 835–838
samples, defined pattern strings, 195–196
saved queries, RDBMS (relational database management system), 24
saving SELECT statement, 111–112
scalar subqueries, 400, 402
schema for sample databases. *See* Appendix B
School Scheduling Database
aggregate functions, 462–463, 851–852
AND, 659–663
CASE (conditional expressions), 698–702
DELETE statement, 617–618
driver tables, 739–742
expressions, 167–169
GROUP BY, 495–496
HAVING clause, 522–525
INNER JOIN
looking for matching values, 311
two tables, 297–298
INSERT statement, 595–596
NOT, 659–663
OUTER JOIN, 356–359
ROLLUP, 782–783
search conditions, 230–231
SELECT statement, 117–118
subqueries
in expressions, 427–428
in filters, 433–434
UNION, 390–392
unlinked tables, 731–734
UPDATE statement, 562–565
windows functions, 840–841
search conditions, 177
AND, 205–206
OR, 206–209
order of precedence, 214–215
less is more, 217–218
prioritizing, 215–217
sample statements, 226–233
Bowling League Database, 231–232
Entertainment Agency Database, 228–229
Recipes Database, 232–233
Sales Orders Database, 227–228
School Scheduling Database, 230–231
syntax diagrams, 205, 680
searched CASE, solving problems, 688–691
searching, for unknown values, 199–201
secondary tables, one-to-one relationships, 25
SELECT clause, expressions, 150–151
concatenation expressions, 151–152
naming, 152–154
SELECT DISTINCT statements, simulating, 484–485
SELECT expression, 372
inserting data, 581–587
SELECT queries, 106
diagrams, 106
sample statements, 113–121
sorting information, 108–111
SELECT statement, 90, 91–93
FROM clause, 331
clauses, 91, 92–93
complex SELECT statements, combining, 375–379
data versus information, 93–95
date expressions, time expressions, 156
diagrams, 92
embedding
INNER JOIN, 284–286
OUTER JOINs, 330–333
expressions
date expressions, 156–157
mathematical expressions, 154–156

GROUP BY, syntax diagrams, 475
requesting all columns, 101–103
retrieving multiple columns, 100–101
rows, eliminating duplicates, 103–105
sample statements, 113–121
Bowling League Database, 119–120
Entertainment Agency Database, 116–117
Recipes Database, 120–121
Sales Orders Database, 114–115
School Scheduling Database, 117–118
saving, 111–112
simple SELECT statements
syntax diagrams, 403
writing requests with UNION, 372–375
sorting information, 105–107
translating requests into SQL, 95–99
WHERE clause, 176–178
using, 179–181
selected rows, updating, 631–633
SEQUEL (Structured English Query Language), 72
SEQUEL-XRM, 72
set diagrams, 246–247
set membership, IN predicate, 411–417
set operations, 243–244
classic set operations versus SQL
EXCEPT, 265–268
INTERSECT, 262–265
UNION, 268–271
classic versus SQL, 259–262
difference, 243, 250
problems you can solve with, 256–257
result sets, 252–256
set theory, 250–252
intersection, 243, 244
problems you can solve with, 249–250
result sets, 246–249
set theory, 244–245
union, 243
combining result sets, 259–261
problems you can solve with, 261–262
set theory, 257–259
set theory
difference, 250–252
intersection, 244–245
union, 257–259
sets, 242–243, 628
AND, multiple AND criteria, 628–629
combining
with UNION, 268–271
with union, 257–259
finding multiple matches in the same table
EXISTS, 646–648
GROUP BY, 648–652
HAVING, 648–652
IN, 644–646
INNER JOIN, 642–644
including some criteria but excluding others, 630–631
NOT
GROUP BY, 638–641
HAVING, 638–641
multiple NOT criteria, 629–630
NOT EXISTS, 637–638
NOT IN, 635–637
OUTER JOIN, 632–635
shortcuts, requesting all columns with SELECT statement, 101–103
simple CASE, solving problems, 683–687
simple primary keys, 57
simple SELECT statements, syntax diagrams, 403
simple UPDATE expression, 537–538
simple WHERE clause, deleting some rows, 607
simulating SELECT DISTINCT statements, GROUP BY, 484–485
single quote, 133
smallest values, finding, with MIN, 454–455
solving problems
with searched CASE, 688–691
with simple CASE, 683–687
with unlinked data, 714–716
SOME, 417–420
some rows, deleting with DELETE statement, 607–611

sort order, 108–109
sorting
  by alphabetical order, 108
  information
    ORDER BY clause, SELECT queries, 108–111
    SELECT statement, 105–107
    UNION, 381–383
special predicate keywords, subqueries
  ALL, 417–420
  ANY, 417–420
  EXISTS, 420–422
  IN predicate, 411–417
  SOME, 417–420
specifying, explicit values, 132–133
Specifying Queries As Relational Expressions (SQUARE), 73
splitting data into quintiles, 824–827
SQL (Structured Query Language)
  commercial implementations, 83
  future of, 83–84
  history of SQL (Structured Query Language)
    early vendor implementations, 73–74
    evolution of SQL/86, 76–79
    standards, 75–76
  origins of, 72–73
  reasons for learning, 84
  set operations
    classic versus SQL, 259–262
    EXCEPT, 265–268
    INTERSECT, 262–265
    UNION, 268–271
  standards. *See* standards
SQL Standard, structure of, 81–82
SQL/86, 76
  evolution of, 76–79
SQL/89, 76–77
SQL/92, 78–79
SQL:2016, 83–84
SQL/Data System (SQL/DS), 74
SQL/DS (SQL/Data System), 74
SQRT, 142
SQUARE (Specifying Queries As Relational Expressions), 73
standards
  evolution of SQL/86, 76–79
  FIPS (Federal Information Processing Standard), 80
  history of SQL (Structured Query Language), 75–76
  ODBC (Open Database Connectivity), 80
  SAA (Systems Application Architecture), 80
  SQL Standard, structure of, 81–82
  SQL/89, 76–77
  SQL/92, 78–79
  X/OPEN, 79
START TRANSACTION, 540
static data, 16
Stonebraker, Michael, 74
string comparison, case sensitivity, 197
string values, comparing, 181–184
structure of
  columns, fine-tuning, 38–40
  databases, 34–35
  SQL Standard, 81–82
  tables, 48–50
Structured English Query Language (SEQUEL), 72
sub-groups, grouping, 750–753
subqueries, 400, 422
  aggregate functions, 457–458
    COUNT, 406–408
    MAX, 406–408
  as column expressions, 402–405, 422–423
  deleting some rows, DELETE statement, 609–611
  in filter rows, 543–546
  as filters, 408–411, 423–424
  row subqueries, 400–402
  sample statements, 424–425
    Bowling League Database, 428–429, 434–435
    Entertainment Agency Database, 426–427, 432–433
    in expressions, 425–430
    in filters, 430–437
    Recipes Database, 429–430, 435–437

Sales Orders Database, 425–426, 430–431
School Scheduling Database, 427–428, 433–434
scalar subqueries, 400, 402
special predicate keywords
ALL, 417–420
ANY, 417–420
EXISTS, 420–422
IN predicate, 411–417
SOME, 417–420
table subqueries, 400, 402
UPDATE expressions, 548–551
in WHERE clauses, GROUP BY, 483–484
subtotal combinations, calculating with CUBE, 765–770
SUM, computing totals, 450–451
SYMMETRIC, 189–190
syntax
CASE (conditional expressions), 678–682
FULL OUTER JOIN, 344–347
GROUP BY, 475–481, 753
INNER JOIN, 276
OUTER JOIN, 324–330
subqueries as column expressions, 402–405
subqueries as filters, 408–411
windows functions, 798–800
syntax diagrams
aggregate functions, 444
CASE expression, 679
column references, 276
comparison condition, 181
concatenation expressions, 138
date and time literals, 136
date expressions, 147
DELETE statement, 604
FULL OUTER JOIN, 344
INSERT statement with SELECT expressions, 581
INSERT statement with VALUES clause, 575
mathematical expressions, 143
naming expressions, 153
NOT operator, 202
Null condition, 199
numeric literals, 135
pattern match condition, 195
predicates, 681–682
query using INNER JOIN on two tables, 277
range condition, 189
search conditions, 205
SELECT statement that includes value expression, 159
SELECT statement with GROUP BY clause, 475
SELECT statement with WHERE clause, 177
simple SELECT statements, 403
time expressions, 149
UNION JOIN, 348
UNION statement, 370
using UNION to combine two simple SELECT statements, 372
utilizing, 5–9
Value Expression, 679
value expressions, 157, 549
System R, 18, 72, 73

T

table references, 345
table subqueries, 400, 402
tables
assigning correlation names to, INNER JOIN, 282–284
derived tables, 284
driver tables, 717
Bowling League Database, 742–743
Entertainment Agency Database, 737–739
Sales Orders Database, 736–737
School Scheduling Database, 739–742
setting up, 717–720
using, 720–725
fine-tuning, 46
naming conventions, 46–48
relationships, 60–63
relationships, deletion rule, 63–64
relationships, participation, 64–68
resolving duplicate columns, 50–56
structure of tables, 48–50

INNER JOIN, 277–282
linking, embedding JOINs within JOINs, 286–291
linking tables, 44–45
naming conventions, 46–48
OUTER JOIN, 324–330
primary keys, 56–60
relational databases, 20–21
unlinked tables, 713
creating, 710–712
techniques for grouping, variations on, 775–780
TIME, 136
time expressions, 149–150
SELECT statement, 156
syntax diagrams, 149
TIMESTAMP, 137, 138
TOP, 111
totals
calculating totals on combinations, using CUBE, 765–770
computing with SUM, 450–451
creating a union of totals, with GROUPING SETS, 771–775
in hierarchies, ROLLUP, 754–765
transactions, 540–541
translating requests into SQL, SELECT statement, 95–99
translation statements, 108
triggers, 549, 551
tuple. *See* rows
type of participation, relationships, 64–68
types of data, expressions, 127–130
types of expressions
concatenation expressions, 138–141
date and time arithmetic expressions, 138, 146
date expressions, 147–148
time expressions, 149–150
mathematical expressions, 138, 142–146

U

underscore (_), 195
UNION, 268–271, 369–372
combining three tables, 380
DISTINCT, 389
fetching data from two tables, 371
sample statements, 385–395
Bowling League Database, 392–394
Entertainment Agency Database, 389–390
Recipes Database, 394–395
Sales Orders Database, 385–389
School Scheduling Database, 390–392
sorting, 381–383
syntax diagrams, 370
combining two simple SELECT statements, 372
uses for, 383–384
using more than once, 379–381
writing requests, 372
combining complex SELECT statements, 375–379
using simple SELECT statements, 372–375
union, 243
problems you can solve with, 261–262
result sets, combining, 259–261
set theory, 257–259
UNION ALL, 371
UNION JOIN, 348
union of totals, creating with GROUPING SETS, 771–775
unknown values. *See* Null, 161
unlinked data, 710–713
driver tables, 717
setting up, 717–720
using, 720–725
sample statements
driver tables, 736–743
unlinked tables, 726–736
solving problems, 714–716
unlinked tables, 713
Bowling League Database, 734–735
creating, 710–712
Entertainment Agency Database, 728–731
Sales Orders Database, 726–728
School Scheduling Database, 731–734

UPDATE clause, JOIN, 546–548
UPDATE expressions
  simple UPDATE expression, 537–538
  subqueries, 548–551
UPDATE statement, 536–537
  sample statements, 552–553
    Bowling League Database, 565–568
    Entertainment Agency Database, 558–562
    Sales Orders Database, 554–557
    School Scheduling Database, 562–565
  subqueries in filter rows, 543–546
  updating
    multiple columns, 541–543
    selected rows, 631–633
  uses for, 551–552
updating
  multiple columns, 541–543
  selected rows, 631–633
USING clause, 280–281
USING syntax, OUTER JOIN, 329

V

Value Expression, 679–680
  syntax diagram, 679
value expressions, 157–159
  syntax diagrams, 157, 549
values
  converting into data types, CAST function, 130–132
  finding largest values, with MAX, 452–454
  inserting with INSERT statements, 575–578
  literal values, 132–133
  matching values, finding with INNER JOIN, 293–294
  mean values, calculating with AVG, 451–452
  missing values, 161
    finding with OUTER JOIN, 349
  non-key values, FULL OUTER JOIN, 347–348
  Null, 159–162
  smallest values, finding with MIN, 454–455
  unknown values, 161
VALUES clause
  DELETE statement, 603–604
  INSERT statement, 575–578
  syntax diagrams, INSERT statement, 575
VALUES keyword, 576
Venn, John, 246–247
Venn diagram, 246–247
verifying you're updating the correct rows, 632–633
  DELETE statement, 607–608
views, relational databases, 23–24
VM/CMS operating system, 74

W

WHERE clause, 176–178
  CASE (conditional expressions), 691
  comparison predicate, 181
    comparing string values, 181–184
    equality and inequality, 185–186
    less than and greater than, 186–188
  DELETE statement, 603–604
  deleting, some rows, 607
  filtering, 510–513
  IS NULL, unknown values, 199–201
  LIKE predicate, pattern match condition, 194–199
  IN predicate, membership condition, 192–194
  range condition, 189–192
  row subqueries, 401–402
  SELECT statement, 93
  subqueries
    as filters, 423–424
    GROUP BY, 483–484
  using, 179–181
WIDTH_BUCKET, 142
windows functions, 794–798
  aggregate functions, 827–834
  calculating row number, 814–818
  OVER()
    ORDER BY clause, 805–810
    PARTITION BY, 800–801
    ROWS (or RANGE), 811–814

ranking data with RANK(), 818–824
Sales Orders Database, 838–839
sample statements, 834–835
aggregate functions, 844–852
NTILE(), 842–844
RANK(), DENSE_RANK, and PERCENT_RANK, 838–841
ROW_NUMBER(), 835–838
School Scheduling Database, 840–841
splitting data into quintiles with NTILE(), 824–827
syntax, 798–800
WITH RECURSIVE, 717
Wong, Eugene, 74
writing requests with UNION, 372
combining complex SELECT statements, 375–379
using simple SELECT statements, 372–375

X–Y

X3, 75
X3H2, 75–76
evolution of SQL/86, 76–79
XML (eXtensible Markup Language), 19
X/OPEN, 79

Z

zero, 160
zero-length string, 160