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®, Third Edition

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.


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

— Graham Mandeno, Database Consultant

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

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

— Arvin Meyer, MCP, MVP

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

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

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

— Doug Steele, Microsoft Access Developer and author

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)
Contents

Foreword xv
Preface xvi
About the Authors xviii

Introduction xx
Are You a Mere Mortal? xx
About This Book xxi
What This Book Is Not xxiii
How to Use This Book xxiii
Reading the Diagrams Used in This Book xxiv
Sample Databases Used in This Book xxviii
“Follow the Yellow Brick Road” xxx

Part I Relational Databases and SQL 1

CHAPTER 1 What Is Relational? 3
Types of Databases 3
A Brief History of the Relational Model 4
   In the Beginning . . . 4
   Relational Database Systems 5
Anatomy of a Relational Database 7
   Tables 7
   Fields 9
   Records 9
   Keys 9
   Views 11
   Relationships 12
What’s in It for You? 17
   Where Do You Go from Here? 18
Summary 19
CHAPTER 5  Getting More Than Simple Columns  109

What Is an Expression?  110
What Type of Data Are You Trying to Express?  111
Changing Data Types: The CAST Function  114
Specifying Explicit Values  116
  Character String Literals  116
  Numeric Literals  118
  Datetime Literals  119
Types of Expressions  121
  Concatenation  122
  Mathematical Expressions  125
  Date and Time Arithmetic  129
Using Expressions in a SELECT Clause  133
  Working with a Concatenation Expression  134
  Naming the Expression  135
  Working with a Mathematical Expression  137
  Working with a Date Expression  138
  A Brief Digression: Value Expressions  139
That “Nothing” Value: Null  141
  Introducing Null  142
  The Problem with Nulls  143
Sample Statements  144
Summary  153
Problems for You to Solve  154
CHAPTER 8  INNER JOINs  249
What Is a JOIN?  249
The INNER JOIN  250
  What’s “Legal” to JOIN?  250
  Column References  251
  Syntax  252
  Check Those Relationships!  267
Uses for INNER JOINs  268
  Find Related Rows  268
  Find Matching Values  269
Sample Statements  269
  Two Tables  270
  More Than Two Tables  276
  Looking for Matching Values  283
Summary  294
Problems for You to Solve  295

CHAPTER 9  OUTER JOINs  299
What Is an OUTER JOIN?  299
The LEFT/RIGHT OUTER JOIN  301
  Syntax  302
The FULL OUTER JOIN  320
  Syntax  320
  FULL OUTER JOIN on Non-Key Values  323
  UNION JOIN  323
Uses for OUTER JOINs  324
  Find Missing Values  324
  Find Partially Matched Information  325
Sample Statements  325
Summary  341
Problems for You to Solve  341
### CHAPTER 13  Grouping Data  445

**Why Group Data?**  446

**The GROUP BY Clause**  448

- Syntax  449
- Mixing Columns and Expressions  454
- Using GROUP BY in a Subquery in a WHERE Clause  456
- Simulating a SELECT DISTINCT Statement  457

- “Some Restrictions Apply”  458
- Column Restrictions  459
- Grouping on Expressions  461

**Uses for GROUP BY**  462

**Sample Statements**  463

**Summary**  474

**Problems for You to Solve**  475

---

### CHAPTER 14  Filtering Grouped Data  477

**A New Meaning of “Focus Groups”**  478

**Where You Filter Makes a Difference**  482

- Should You Filter in WHERE or in HAVING?  482
- Avoiding the HAVING COUNT Trap  485

**Uses for HAVING**  490

**Sample Statements**  491

**Summary**  499

**Problems for You to Solve**  500
Part VI  Introduction to Solving Tough Problems  591

CHAPTER 18  “NOT” and “AND” Problems  593

A Short Review of Sets  593
Sets with Multiple AND Criteria  594
Sets with Multiple NOT Criteria  595
Sets Including Some Criteria but Excluding Others  596
Finding Out the “Not” Case  597
Using OUTER JOIN  598
Using NOT IN  601
Using NOT EXISTS  603
Using GROUP BY/HAVING  604
Finding Multiple Matches in the Same Table  607
Using INNER JOIN  608
Using IN  610
Using EXISTS  612
Using GROUP BY/HAVING  614
Sample Statements  618
Summary  636
Problems for You to Solve  637

CHAPTER 19  Condition Testing  641

Conditional Expressions (CASE)  641
Why Use CASE?  642
Syntax  642
Solving Problems with CASE  647
Solving Problems with Simple CASE  647
Solving Problems with Searched CASE  652
Using CASE in a WHERE Clause  655
Sample Statements  655
Summary  669
Problems for You to Solve  669

CHAPTER 20  Using Unlinked Data and “Driver” Tables  671

What Is Unlinked Data?  672
Deciding When to Use a CROSS JOIN  675
Solving Problems with Unlinked Data  676
In the 25 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, Third 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
Learning how to retrieve information from or manipulate information in a database is commonly a perplexing exercise. However, it can be a relatively easy task as long as you understand the question you’re asking or the change you’re trying to make to the database. After you understand the problem, you can translate it into the language used by any database system, which in most cases is Structured Query Language (SQL). You have to translate your request into an SQL statement so that your database system knows what information you want to retrieve or change. SQL provides the means for you and your database system to communicate.

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

With this in mind, we decided it was time to write a book that would help people learn how to query a database properly and effectively. We produced the first edition of this book in 2000. We created a second edition in 2008 that introduced basic ways to change data in your database using SQL. With this new edition, 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. The result of our efforts 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 third edition includes hundreds of new examples, and we included versions of the sample databases using Microsoft Office Access, Microsoft SQL Server, and the popular open-source MySQL database system. When you finish reading this book, you'll have the skills you need to retrieve or modify any information you require.

Acknowledgments

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

First and foremost, we want to thank our acquisitions editor, Joan Murray, for helping us get signed up to produce this third edition. Thanks also to Developmental Editor Chris Cleveland for shepherding us along the way. And we can't forget Production Editor Tonya Simpson and the production staff—they're a great team! Finally, thanks to Associate Publisher David Dusthimer, who put this team together and kept a watchful eye over the entire process.

Next, we'd like to acknowledge our technical editor, Dale Wallentine. We also had help from some of our database friends—Jeff Boyce, Ben Clothier, Henry Habermacher, Leo theDBGuy, and Doug Steele. Thanks once again to all of you for your time and input and for helping us to make this a solid treatise on SQL queries.

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

**John L. Viescas** is an independent database consultant with more than 45 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), and *Building Microsoft Access Applications* (Microsoft Press, 2005).

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

You can visit John’s Web site at www.viescas.com or contact him by e-mail at john@viescas.com.
Michael J. Hernandez has been an independent relational database consultant specializing in relational database design. He has more than 20 years of experience in the technology industry, developing database applications for a wide variety of clients. He’s been a contributing author to a wide variety of magazine columns, white papers, books, and periodicals, and is coauthor of the best-selling *SQL Queries for Mere Mortals*. Mike has been a top-rated and noted technical trainer for the government, the military, the private sector, and companies throughout the United States. He has spoken at numerous national and international conferences, and has consistently been a top-rated speaker and presenter.

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

He says he’s never going to retire, per se, but rather just change whatever it is he’s doing whenever he finally gets tired of it and move on to something else that interests him.
If you’ve used a computer more than casually, you have probably used Structured Query Language or SQL—perhaps without even knowing it. SQL is the standard language for communicating with most database systems. Any time you import data into a spreadsheet or perform a merge into a word processing program, you’re most likely using SQL in some form or another. Every time you go online to an e-commerce site on the Web and place an order for a book, a recording, a movie, or any of the dozens of other products you can order, there’s a very high probability that the code behind the web page you’re using is accessing its databases with SQL. If you need to get information from a database system that uses SQL, you can enhance your understanding of the language by reading this book.

Are You a Mere Mortal?

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

- If you use computer applications that let you access information from a database system, you’re probably a mere mortal. The first time you don’t get the information you expected using the query tools built in to your application, you’ll need to explore the underlying SQL statements to find out why.
• If you have recently discovered one of the many available desktop database applications but are struggling with defining and querying the data you need, you’re a mere mortal.

• If you’re a database programmer who needs to “think outside of the box” to solve some complex problems, you’re a mere mortal.

• If you’re a database guru in one product but are now faced with integrating the data from your existing system into another system that supports SQL, you’re a mere mortal.

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

About This Book

Everything you read in this book is based on the current International Organization for Standardization (ISO) Standard for the SQL database language – SQL/Foundation (document ISO/IEC 9075-2:2011), as currently implemented in most of the popular commercial database systems. The ISO document was also adopted by the American National Standards Institute (ANSI), so this is truly an international standard. The SQL you’ll learn here is not specific to any particular software product.

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

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

- Part I, “Relational Databases and SQL,” explains how modern database systems are based on a rigorous mathematical model and provides a brief history of the database query language that has evolved into what we know as SQL. We also discuss some simple rules that you can use to make sure your database design is sound.

- Part II, “SQL Basics,” introduces you to using the SELECT statement, creating expressions, and sorting information with an ORDER BY clause. You’ll also learn how to filter data by using a WHERE clause.

- Part III, “Working with Multiple Tables,” shows you how to formulate queries that draw data from more than one table. Here we show you how to link tables in a query using the INNER JOIN, OUTER JOIN, and UNION operators, and how to work with subqueries.

- Part IV, “Summarizing and Grouping Data,” discusses how to obtain summary information and group and filter summarized data. Here is where you’ll learn about the GROUP BY and HAVING clauses.

- Part V, “Modifying Sets of Data,” explains how to write queries that modify a set of rows in your tables. In the chapters in this section, you’ll learn how to use the UPDATE, INSERT, and DELETE statements.

- Part VI, “Introduction to Solving Tough Problems,” dips your toes into more complex problems. In the chapters in this section, you’ll expand your horizons to include solving complex “NOT” and “AND” problems (multiple conditions on one table), performing logical evaluations with CASE, and thinking “outside the box” using “unlinked” tables (Cartesian Products).

At the end of the book in the appendices, you’ll find syntax diagrams for all the SQL elements you’ve learned, layouts of the sample databases, a list of date and time manipulation functions implemented in five of the major database systems, and book recommendations to further your study of SQL. You can download the five sample databases for the three database systems (Microsoft Access, Microsoft SQL Server, and MySQL) from www.informit.com/title/9780321992475 and clicking the Downloads tab.
What This Book Is Not

Although this book is based on the 2011 SQL Standard that was current at the time of this writing, it does not cover every aspect of the standard. In truth, many features in the 2011 SQL Standard won’t be implemented for many years—if at all—in the major database system implementations. The fundamental purpose of this book is to give you a solid grounding in writing queries in SQL. Throughout the book, you’ll find us recommending that you “consult your database documentation” for how a specific feature might or might not work. That’s not to say we covered only the lowest common denominator for any feature among the major database systems. We do try to caution you when some systems implement a feature differently or not at all.

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

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

How to Use This Book

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

At the end of many of the chapters you’ll find an extensive set of “Sample Statements,” their solutions, and sample result sets. We recommend that you study several of the samples to gain a better understanding of the techniques
involved and then try working out some of the later “Problems for You to Solve” yourself without looking at the solutions we propose.

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

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

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

**Reading the Diagrams Used in This Book**

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

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

Figure 1  Syntax line end points for statements and defined terms

Figure 2  Sample statement and defined term diagrams
1. **Statement start point**—denotes the beginning of the main syntax line for a statement. Any element that appears directly on the main syntax line is a required element, and any element that appears below it is an optional element.

2. **Main syntax line**—determines the order of all required and optional elements for the statement or defined term. Follow this line from left to right (or in the direction of the arrows) to build the syntax for the statement or defined term.

3. **Keyword(s)**—indicates a major word in SQL grammar that is a required part of the syntax for a statement or defined term. In a diagram, keywords are formatted in capital letters and bold font. (You don’t have to worry about typing a keyword in capital letters when you actually write the statement in your database program, but it does make the statement easier to read.)

4. **Literal entry**—specifies the name of a value you explicitly supply to the statement. A literal entry is represented by a word or phrase that indicates the type of value you need to supply. Literal entries in a diagram are formatted in all lowercase letters.

5. **Defined term**—denotes a word or phrase that represents some operation that returns a final value to be used in this statement. We’ll explain and diagram every defined term you need to know as you work through the book. Defined terms are always formatted in italic letters.

6. **Optional element**—indicates any element or group of elements that appears below the main syntax line. An optional element can be a statement, keyword, defined term, or literal value and, for purposes of clarity, appears on its own line. In some cases, you can specify a set of values for a given option, with each value separated by a comma (see number 8). Also, several optional elements have a set of sub-optional elements (see number 7). In general, you read the syntax line for an optional element from left to right, in the same manner that you read the main syntax line. Always follow the directional arrows and you’ll be in good shape. Note that some options allow you to specify multiple values or choices, so the arrow will flow from right to left. After you’ve entered all the items you need, however, the flow will return to normal from left to right. Fortunately, all optional elements work the same way. After we show you how to use an optional element later in the book, you’ll know how to use any other optional element you encounter in a syntax diagram.

7. **Sub-optional element**—denotes any element or group of elements that appears below an optional element. Sub-optional elements allow you to fine-tune your statements so that you can work with more complex problems.

8. **Option list separator**—indicates that you can specify more than one value for this option and that each value must be separated with a comma.

9. **Alternate option**—denotes a keyword or defined term that can be used as an alternative to one or more optional elements. The syntax line for an alternate option bypasses the syntax lines of the optional elements it is meant to replace.
Now that you're familiar with these elements, you'll be able to read all the syntax diagrams in the book. And on those occasions when a diagram requires further explanation, we provide you with the information you need to read the diagram clearly and easily. To help you better understand how the diagrams work, here's a sample SELECT statement that we built using Figure 2.

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

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

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

On the book website at www.informit.com/title/9780321992475, you'll find a downloadable file on the Downloads tab containing five sample databases that we use for the example queries throughout the book. We've also included diagrams of the database structures in Appendix B.

1. **Sales Orders.** This is a typical order entry database for a store that sells bicycles and accessories. (Every database book needs at least one order entry example, right?)

2. **Entertainment Agency.** We structured this database to manage entertainers, agents, customers, and bookings. You would use a similar design to handle event bookings or hotel reservations.

3. **School Scheduling.** You might use this database design to register students at a high school or community college. This database tracks not only class registrations but also which instructors are assigned to each class and what grades the students received.

4. **Bowling League.** This database tracks bowling teams, team members, the matches they played, and the results.

5. **Recipes.** You can use this database to save and manage all your favorite recipes. We even added a few that you might want to try.

In the sample files, you can find all five sample databases in three different formats:

- Because of the great popularity of the Microsoft Office Access desktop database, we created one set of databases (.accdb file extension) using Microsoft Access 2007 (Version 12.0). We chose Version 12 of this product because it closely supports the current ISO/IEC SQL Standard, and you can open database files in this format using Access 2007, 2010, 2013, and later. You can find these files in the MSAccess subfolder.

- The second format consists of database files (.mdf file extension) created using Microsoft SQL Server 2012 Express Edition. You can find these files in the MSSQLServer folder, and you can attach these files to a Microsoft SQL Server 2012 or later server. We have also included SQL command files (.sql file extension) that you can use to create the samples on a Microsoft SQL Server from scratch. You can find these files in the SQLScripts subfolder. You can obtain a free copy of Microsoft SQL Server 2012 Express Edition at www.microsoft.com/en-us/sqlserver/editions/2012-editions/express.aspx.

- We created the third set of databases using the popular open-source MySQL version 5.6 Community Edition database system. You can use
the scripts (.sql file extension) you will find in the SQLScripts subfolder to create the database structure, load the data, and create the sample views and stored procedures in your own MySQL data folder. You can obtain a free copy of the community edition of the MySQL database system at www.mysql.com/.

To install the sample files, see the file ReadMe.txt included in the files you can download from www.informit.com/title/9780321992475.

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

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

❖ **Caution** Throughout the book, we use ISO-Standard SQL when we explain concepts and show you sample statements. In many cases, we were able to use this SQL directly to create the sample Views or Stored Procedures that you'll find in the sample databases. However, in many cases we had to modify the sample SQL so that it would work correctly with the target database system. For example, to create date expressions or calculations, we chose to use the appropriate function supported by the target database system. (For a list of all date and time functions supported by five of the major database systems, see Appendix C, “Date and Time Types, Operations, and Functions.”)
Also, although we used scripts that closely match the original samples in the book, both Microsoft SQL Server and MySQL will modify the original SQL to “optimize” it before saving the view or stored procedure. If you use Design in SQL Server Management Studio or Alter in MySQL Workbench to edit the view or procedure, what you see saved in the database might differ considerably from the script we used to define the view or procedure. When in doubt, always refer to the companion script file to see the SQL we used.

“Follow the Yellow Brick Road”

—Munchkin to Dorothy in The Wizard of Oz

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

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

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

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

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

Topics Covered in This Chapter

What Is a Set, Anyway?
Operations on Sets
Intersection
Difference
Union
SQL Set Operations
Summary

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

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

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

**What Is a Set, Anyway?**

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

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

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

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

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

**Operations on Sets**

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

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

**Intersection**

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

**Intersection in Set Theory**

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

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

\[1, 5, 8, 9, 32, 55, 78\]

The second set of numbers is as follows:

\[3, 7, 8, 22, 55, 71, 99\]

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

\[8, 55\]

The individual entries—the members—of each set don’t have to be just single values. In fact, when solving problems with SQL, you’ll probably deal with sets of rows.
According to set theory, when a member of a set is something more than a single number or value, each member (or object) of the set has multiple attributes or bits of data that describe the properties of each member. For example, your favorite stew recipe is a complex member of the set of all recipes that contains many different ingredients. Each ingredient is an attribute of your complex stew member.

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

<table>
<thead>
<tr>
<th>Potatoes</th>
<th>Water</th>
<th>Lamb</th>
<th>Peas</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rice</td>
<td>Chicken Stock</td>
<td>Chicken</td>
<td>Carrots</td>
</tr>
<tr>
<td>Pasta</td>
<td>Water</td>
<td>Tofu</td>
<td>Snap Peas</td>
</tr>
<tr>
<td>Potatoes</td>
<td>Beef Stock</td>
<td>Beef</td>
<td>Cabbage</td>
</tr>
<tr>
<td>Pasta</td>
<td>Water</td>
<td>Pork</td>
<td>Onions</td>
</tr>
</tbody>
</table>

A second set might look like the following:

<table>
<thead>
<tr>
<th>Potatoes</th>
<th>Water</th>
<th>Lamb</th>
<th>Onions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rice</td>
<td>Chicken Stock</td>
<td>Turkey</td>
<td>Carrots</td>
</tr>
<tr>
<td>Pasta</td>
<td>Vegetable Stock</td>
<td>Tofu</td>
<td>Snap Peas</td>
</tr>
<tr>
<td>Potatoes</td>
<td>Beef Stock</td>
<td>Beef</td>
<td>Cabbage</td>
</tr>
<tr>
<td>Beans</td>
<td>Water</td>
<td>Pork</td>
<td>Onions</td>
</tr>
</tbody>
</table>

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

| Potatoes | Beef Stock | Beef | Cabbage |

"Thinking in Sets" 223
**Intersection between Result Sets**

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

<table>
<thead>
<tr>
<th>Recipe</th>
<th>Starch</th>
<th>Stock</th>
<th>Meat</th>
<th>Vegetable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lamb Stew</td>
<td>Potatoes</td>
<td>Water</td>
<td>Lamb</td>
<td>Peas</td>
</tr>
<tr>
<td>Chicken Stew</td>
<td>Rice</td>
<td>Chicken Stock</td>
<td>Chicken</td>
<td>Carrots</td>
</tr>
<tr>
<td>Veggie Stew</td>
<td>Pasta</td>
<td>Water</td>
<td>Tofu</td>
<td>Snap Peas</td>
</tr>
<tr>
<td>Irish Stew</td>
<td>Potatoes</td>
<td>Beef Stock</td>
<td>Beef</td>
<td>Cabbage</td>
</tr>
<tr>
<td>Pork Stew</td>
<td>Pasta</td>
<td>Water</td>
<td>Pork</td>
<td>Onions</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Recipe</th>
<th>Starch</th>
<th>Stock</th>
<th>Meat</th>
<th>Vegetable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lamb Stew</td>
<td>Potatoes</td>
<td>Water</td>
<td>Lamb</td>
<td>Peas</td>
</tr>
<tr>
<td>Turkey Stew</td>
<td>Rice</td>
<td>Chicken Stock</td>
<td>Turkey</td>
<td>Carrots</td>
</tr>
<tr>
<td>Veggie Stew</td>
<td>Pasta</td>
<td>Vegetable Stock</td>
<td>Tofu</td>
<td>Snap Peas</td>
</tr>
<tr>
<td>Irish Stew</td>
<td>Potatoes</td>
<td>Beef Stock</td>
<td>Beef</td>
<td>Cabbage</td>
</tr>
<tr>
<td>Pork Stew</td>
<td>Beans</td>
<td>Water</td>
<td>Pork</td>
<td>Onions</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Recipe</th>
<th>Starch</th>
<th>Stock</th>
<th>Meat</th>
<th>Vegetable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lamb Stew</td>
<td>Potatoes</td>
<td>Water</td>
<td>Lamb</td>
<td>Peas</td>
</tr>
<tr>
<td>Irish Stew</td>
<td>Potatoes</td>
<td>Beef Stock</td>
<td>Beef</td>
<td>Cabbage</td>
</tr>
</tbody>
</table>
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.

![Set Diagram](image)

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

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

Yes, we know what you're thinking. If your recipe table looks like the samples above, you could simply say the following:
“Show me the recipes that have beef as the meat ingredient and onions as the vegetable ingredient.”

Translation
Select the recipe name from the recipes table where meat ingredient is beef and vegetable ingredient is onions

Clean Up
Select the recipe name from the recipes table where meat ingredient \( \text{=} \) beef and vegetable ingredient \( \text{=} \) onions

SQL
```
SELECT RecipeName
FROM Recipes
WHERE MeatIngredient = 'Beef'
    AND VegetableIngredient = 'Onions'
```
Got the hang of it? The bottom line is that when you’re faced with solving a problem involving complex criteria, a set diagram can be an invaluable way to see the solution expressed as the intersection of SQL result sets.

**Problems You Can Solve with an Intersection**

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

“Show me customers and employees who have the same name.”
“Find all the customers who ordered a bicycle and also ordered a helmet.”
“List the entertainers who played engagements for customers Bonnicksen and Rosales.”
“Show me the students who have an average score of 85 or better in Art and who also have an average score of 85 or better in Computer Science.”
“Find the bowlers who had a raw score of 155 or better at both Thunderbird Lanes and Bolero Lanes.”
“Show me the recipes that have beef and garlic.”

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

“Show me customers and employees who live in the same city.” (JOIN on the city name.)
“List customers and the entertainers they booked.” (JOIN on the engagement number.)
“Find the agents and entertainers who live in the same ZIP Code.” (JOIN on the ZIP Code.)
“Show me the students and their teachers who have the same first name.” (JOIN on the first name.)
“Find the bowlers who are on the same team.” (JOIN on the team ID.)
“Display all the ingredients for recipes that contain carrots.” (JOIN on the ingredient ID.)

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

Difference

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

Difference in Set Theory

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

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

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

The second set of numbers is as follows:

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

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

1, 5, 9, 32, 78
Note that you can turn the previous difference operation around. Thus, the difference of the second set minus the first set is

3, 7, 22, 71, 99

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

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

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

<table>
<thead>
<tr>
<th>Potatoes</th>
<th>Water</th>
<th>Lamb</th>
<th>Peas</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rice</td>
<td>Chicken Stock</td>
<td>Chicken</td>
<td>Carrots</td>
</tr>
<tr>
<td>Pasta</td>
<td>Water</td>
<td>Tofu</td>
<td>Snap Peas</td>
</tr>
<tr>
<td>Potatoes</td>
<td>Beef Stock</td>
<td>Beef</td>
<td>Cabbage</td>
</tr>
<tr>
<td>Pasta</td>
<td>Water</td>
<td>Pork</td>
<td>Onions</td>
</tr>
</tbody>
</table>

A second set might look like this:

<table>
<thead>
<tr>
<th>Potatoes</th>
<th>Water</th>
<th>Lamb</th>
<th>Onions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rice</td>
<td>Chicken Stock</td>
<td>Turkey</td>
<td>Carrots</td>
</tr>
<tr>
<td>Pasta</td>
<td>Vegetable Stock</td>
<td>Tofu</td>
<td>Snap Peas</td>
</tr>
<tr>
<td>Potatoes</td>
<td>Beef Stock</td>
<td>Beef</td>
<td>Cabbage</td>
</tr>
<tr>
<td>Beans</td>
<td>Water</td>
<td>Pork</td>
<td>Onions</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Potatoes</th>
<th>Water</th>
<th>Lamb</th>
<th>Peas</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rice</td>
<td>Chicken Stock</td>
<td>Chicken</td>
<td>Carrots</td>
</tr>
<tr>
<td>Pasta</td>
<td>Water</td>
<td>Tofu</td>
<td>Snap Peas</td>
</tr>
<tr>
<td>Pasta</td>
<td>Water</td>
<td>Pork</td>
<td>Onions</td>
</tr>
</tbody>
</table>

**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.)

<table>
<thead>
<tr>
<th>Recipe</th>
<th>Starch</th>
<th>Stock</th>
<th>Meat</th>
<th>Vegetable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lamb Stew</td>
<td>Potatoes</td>
<td>Water</td>
<td>Lamb</td>
<td>Peas</td>
</tr>
<tr>
<td>Chicken Stew</td>
<td>Rice</td>
<td>Chicken Stock</td>
<td>Chicken</td>
<td>Carrots</td>
</tr>
<tr>
<td>Veggie Stew</td>
<td>Pasta</td>
<td>Water</td>
<td>Tofu</td>
<td>Snap Peas</td>
</tr>
<tr>
<td>Irish Stew</td>
<td>Potatoes</td>
<td>Beef Stock</td>
<td>Beef</td>
<td>Cabbage</td>
</tr>
<tr>
<td>Pork Stew</td>
<td>Pasta</td>
<td>Water</td>
<td>Pork</td>
<td>Onions</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Recipe</th>
<th>Starch</th>
<th>Stock</th>
<th>Meat</th>
<th>Vegetable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lamb Stew</td>
<td>Potatoes</td>
<td>Water</td>
<td>Lamb</td>
<td>Peas</td>
</tr>
<tr>
<td>Turkey Stew</td>
<td>Rice</td>
<td>Chicken Stock</td>
<td>Turkey</td>
<td>Carrots</td>
</tr>
<tr>
<td>Veggie Stew</td>
<td>Pasta</td>
<td>Vegetable Stock</td>
<td>Tofu</td>
<td>Snap Peas</td>
</tr>
<tr>
<td>Irish Stew</td>
<td>Potatoes</td>
<td>Beef Stock</td>
<td>Beef</td>
<td>Cabbage</td>
</tr>
<tr>
<td>Pork Stew</td>
<td>Beans</td>
<td>Water</td>
<td>Pork</td>
<td>Onions</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Recipe</th>
<th>Starch</th>
<th>Stock</th>
<th>Meat</th>
<th>Vegetable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chicken Stew</td>
<td>Rice</td>
<td>Chicken Stock</td>
<td>Chicken</td>
<td>Carrots</td>
</tr>
<tr>
<td>Veggie Stew</td>
<td>Pasta</td>
<td>Water</td>
<td>Tofu</td>
<td>Snap Peas</td>
</tr>
<tr>
<td>Pork Stew</td>
<td>Pasta</td>
<td>Water</td>
<td>Pork</td>
<td>Onions</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Recipe</th>
<th>Starch</th>
<th>Stock</th>
<th>Meat</th>
<th>Vegetable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Turkey Stew</td>
<td>Rice</td>
<td>Chicken Stock</td>
<td>Turkey</td>
<td>Carrots</td>
</tr>
<tr>
<td>Veggie Stew</td>
<td>Pasta</td>
<td>Vegetable Stock</td>
<td>Tofu</td>
<td>Snap Peas</td>
</tr>
<tr>
<td>Pork Stew</td>
<td>Beans</td>
<td>Water</td>
<td>Pork</td>
<td>Onions</td>
</tr>
</tbody>
</table>

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

**Figure 7–3** Finding out which recipes have beef but not onions
The upper full circle represents the set of recipes that contain beef. The lower full circle represents the set of recipes that contain onions. As you remember from the discussion about INTERSECT, where the two circles overlap is where you’ll find the recipes that contain both. The dark-shaded part of the upper circle that’s not part of the overlapping area represents the set of recipes that contain beef but do not contain onions. Likewise, the part of the lower circle that’s not part of the overlapping area represents the set of recipes that contain onions but do not contain beef.

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

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

“Show me the recipes that have beef as the meat ingredient and that do not have onions as the vegetable ingredient.”

**Translation**
Select the recipe name from the recipes table where meat ingredient is beef and vegetable ingredient is not onions

**Clean Up**
Select the recipe name from the recipes table where meat ingredient is beef and vegetable ingredient is not onions

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

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

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

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

![Venn Diagram](image)

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

### Problems You Can Solve with Difference

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

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

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

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

“Show me customers who do not live in the same city as any employees.” (OUTER JOIN on the city name.)

“List customers and the entertainers they did not book.” (OUTER JOIN on the engagement number.)

“Find the agents who are not in the same ZIP Code as any entertainer.” (OUTER JOIN on the ZIP Code.)

“Show me the students who do not have the same first name as any teachers.” (OUTER JOIN on the first name.)

“Find the bowlers who have an average of 150 or higher who have never bowled a game below 125.” (OUTER JOIN on the bowler ID from two different tables.)

“Display all the ingredients for recipes that do not have carrots.” (OUTER JOIN on the recipe ID.)

Don’t worry! We’ll show you all about solving these problems (and more) using OUTER JOINs in Chapter 9, “OUTER JOINs.” Also, because few commercial implementations of SQL support EXCEPT (the keyword for difference), we’ll show how to use an OUTER JOIN to solve many problems that might otherwise require an EXCEPT. In Chapter 18, “‘NOT’ and ‘AND’ Problems,” we’ll show you additional ways to solve EXCEPT problems.

**Union**

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

**Union in Set Theory**

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

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

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

The second set of numbers is as follows:

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

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

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

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

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

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

<table>
<thead>
<tr>
<th>Potatoes</th>
<th>Water</th>
<th>Lamb</th>
<th>Peas</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rice</td>
<td>Chicken Stock</td>
<td>Chicken</td>
<td>Carrots</td>
</tr>
<tr>
<td>Pasta</td>
<td>Water</td>
<td>Tofu</td>
<td>Snap Peas</td>
</tr>
<tr>
<td>Potatoes</td>
<td>Beef Stock</td>
<td>Beef</td>
<td>Cabbage</td>
</tr>
<tr>
<td>Pasta</td>
<td>Water</td>
<td>Pork</td>
<td>Onions</td>
</tr>
</tbody>
</table>

A second set might look like the following:
The union of these two sets is the set of objects from both sets. Duplicates are eliminated.

<table>
<thead>
<tr>
<th>Potatoes</th>
<th>Water</th>
<th>Lamb</th>
<th>Onions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rice</td>
<td>Chicken Stock</td>
<td>Turkey</td>
<td>Carrots</td>
</tr>
<tr>
<td>Pasta</td>
<td>Vegetable Stock</td>
<td>Tofu</td>
<td>Snap Peas</td>
</tr>
<tr>
<td>Potatoes</td>
<td>Beef Stock</td>
<td>Beef</td>
<td>Cabbage</td>
</tr>
<tr>
<td>Beans</td>
<td>Water</td>
<td>Pork</td>
<td>Onions</td>
</tr>
</tbody>
</table>

**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.)

<table>
<thead>
<tr>
<th>Recipe</th>
<th>Starch</th>
<th>Stock</th>
<th>Meat</th>
<th>Vegetable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lamb Stew</td>
<td>Potatoes</td>
<td>Water</td>
<td>Lamb</td>
<td>Peas</td>
</tr>
<tr>
<td>Chicken Stew</td>
<td>Rice</td>
<td>Chicken Stock</td>
<td>Chicken</td>
<td>Carrots</td>
</tr>
<tr>
<td>Veggie Stew</td>
<td>Pasta</td>
<td>Water</td>
<td>Tofu</td>
<td>Snap Peas</td>
</tr>
<tr>
<td>Irish Stew</td>
<td>Potatoes</td>
<td>Beef Stock</td>
<td>Beef</td>
<td>Cabbage</td>
</tr>
<tr>
<td>Pork Stew</td>
<td>Pasta</td>
<td>Water</td>
<td>Pork</td>
<td>Onions</td>
</tr>
</tbody>
</table>
A second query result set might look like this one. (These are recipes from Mike’s cookbook).

<table>
<thead>
<tr>
<th>Recipe</th>
<th>Starch</th>
<th>Stock</th>
<th>Meat</th>
<th>Vegetable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lamb Stew</td>
<td>Potatoes</td>
<td>Water</td>
<td>Lamb</td>
<td>Peas</td>
</tr>
<tr>
<td>Turkey Stew</td>
<td>Rice</td>
<td>Chicken Stock</td>
<td>Turkey</td>
<td>Carrots</td>
</tr>
<tr>
<td>Veggie Stew</td>
<td>Pasta</td>
<td>Vegetable Stock</td>
<td>Tofu</td>
<td>Snap Peas</td>
</tr>
<tr>
<td>Irish Stew</td>
<td>Potatoes</td>
<td>Beef Stock</td>
<td>Beef</td>
<td>Cabbage</td>
</tr>
<tr>
<td>Pork Stew</td>
<td>Beans</td>
<td>Water</td>
<td>Pork</td>
<td>Onions</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Recipe</th>
<th>Starch</th>
<th>Stock</th>
<th>Meat</th>
<th>Vegetable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lamb Stew</td>
<td>Potatoes</td>
<td>Water</td>
<td>Lamb</td>
<td>Peas</td>
</tr>
<tr>
<td>Chicken Stew</td>
<td>Rice</td>
<td>Chicken Stock</td>
<td>Chicken</td>
<td>Carrots</td>
</tr>
<tr>
<td>Veggie Stew</td>
<td>Pasta</td>
<td>Water</td>
<td>Tofu</td>
<td>Snap Peas</td>
</tr>
<tr>
<td>Irish Stew</td>
<td>Potatoes</td>
<td>Beef Stock</td>
<td>Beef</td>
<td>Cabbage</td>
</tr>
<tr>
<td>Pork Stew</td>
<td>Pasta</td>
<td>Water</td>
<td>Pork</td>
<td>Onions</td>
</tr>
<tr>
<td>Turkey Stew</td>
<td>Rice</td>
<td>Chicken Stock</td>
<td>Turkey</td>
<td>Carrots</td>
</tr>
<tr>
<td>Veggie Stew</td>
<td>Pasta</td>
<td>Vegetable Stock</td>
<td>Tofu</td>
<td>Snap Peas</td>
</tr>
<tr>
<td>Pork Stew</td>
<td>Beans</td>
<td>Water</td>
<td>Pork</td>
<td>Onions</td>
</tr>
</tbody>
</table>

Let’s assume you have a nice database containing all your favorite recipes. You really like recipes with either beef or onions, so you want a list of recipes that contain either ingredient. Figure 7–5 (on page 238) shows you the set diagram that helps you visualize how to solve this problem.

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

![Figure 7–5 Finding out which recipes have either beef or onions](image)

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

**Problems You Can Solve with Union**

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

“Show me all the customer and employee names and addresses.”

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

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

“Show me the students who have an average score of 85 or better in Art together with the students who have an average score of 85 or better in Computer Science.”
“Find the bowlers who had a raw score of 155 or better at Thunderbird Lanes combined with bowlers who had a raw score of 140 or better at Bolero Lanes.”

“Show me the recipes that have beef together with the recipes that have garlic.”

As with other “pure” set operations, one of the limitations is that the values must match in all the columns in each result set. This works well if you're unioning two or more sets from the same table—for example, customers who ordered bicycles and customers who ordered helmets. It also works well when you're performing a union on sets from tables that have like columns—for example, customer names and addresses and employee names and addresses. We'll explore the uses of the SQL UNION operator in detail in Chapter 10, “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. We'll show you all about solving problems this way using JOINs in Chapter 8, “INNER JOINs.”

**SQL Set Operations**

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

**Classic Set Operations versus SQL**

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

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

Let’s say you’re trying to solve the following seemingly simple problem:

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

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

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

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

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

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

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

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](image)

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

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

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

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](image)

**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](image)

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

SQL: 
```
SELECT DISTINCT OrderNumber
FROM Order_Details
WHERE ProductNumber IN (1, 2, 6, 11)
EXCEPT
SELECT DISTINCT OrderNumber
FROM Order_Details
WHERE ProductNumber IN (10, 25, 26)
```
Remember from our earlier discussion about the difference operation that the sequence of the sets matters. In this case, you’re asking for bikes “except” helmets. If you want to find out the opposite case—orders for helmets that do not include bikes—you can turn it around as follows:

SQL

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

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

**Combining Sets: UNION**

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

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

**Translation**

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

**Clean Up**

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

**SQL**

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

Actually, that works just fine! So why use a UNION to solve this problem? The truth is, you probably would not. However, if we make the problem more complicated, a UNION would be useful:
“List the customers who ordered a bicycle together with the vendors who provide bicycles.”

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

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

![Figure 7–11 Orders for bicycles or helmets](image)

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  

\[
\text{SELECT DISTINCT OrderNumber} \\
\text{FROM Order_Details} \\
\text{WHERE ProductNumber IN (10, 25, 26)}
\]

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.

![SELECT Expression Diagram](image)

**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  

\[
\text{SELECT DISTINCT OrderNumber} \\
\text{FROM Order_Details} \\
\text{WHERE ProductNumber IN (1, 2, 6, 11)} \\
\text{UNION} \\
\text{SELECT DISTINCT OrderNumber} \\
\text{FROM Order_Details} \\
\text{WHERE ProductNumber IN (10, 25, 26)}
\]

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. We’ll explore UNION in much more detail in Chapter 10.
We began this chapter by discussing the concept of a set. Next, we discussed each of the major set operations implemented in SQL in detail—intersection, difference, and union. We showed how to use set diagrams to visualize the problem you’re trying to solve. Finally, we introduced you to the basic SQL syntax and keywords (INTERSECT, EXCEPT, and UNION) for all three operations just to whet your appetite.

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

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

Likewise, if your problem involves “it must be this but it must not be that,” you might need to solve the “this” and then the “that” and then subtract the “that” from the “this” to get your answer. We showed you the SQL Standard EXCEPT operation, but an OUTER JOIN might also do the trick. Get the details in Chapters 9 and 18.

Finally, we showed you how to add sets of information using a UNION. As promised, we’ll really get into UNION in Chapter 10.
This page intentionally left blank
Index

A
acronyms, naming fields, 24
Actian, 60
agents, 53
aggregate functions, 420-422
  AVG, 427-428
  COUNT, 422-425
filters, 432, 434
MAX, 428-430
MIN, 430-431
Null values, 451
sample statements, 435
  bowling league database, 439-440
  entertainment agency database, 437
  recipes database, 441-442
  sales orders database, 435-436
  school scheduling database, 438
subqueries as columns, 381-383
SUM, 425-426
  using multiple functions, 431-432
aliases, assigning to tables, 258-260
ALL, predicate keywords for subqueries, 392-395
American National Standards Institute (ANSI), 61
analytical databases, 4
AND, 185, 190
  sets, 594-595
ANSI (American National Standards Institute), 61
ANSI/ISO standard, 62-65
ANY, predicate keywords for subqueries, 392-395
approximate numeric types, 112
ASCII collating sequence, 163
assigning correlation names to tables, 258-260
attributes, 221
AVG, calculating mean values, 427-428

B
BETWEEN, 173
Between predicate, 160
binary data types, 112
Boolean data types, 113
bowling league database
  aggregate functions, 439-440
  CASE sample statements, 666-668
  DELETE sample statements, 586-588
driver tables sample statements, 703-704
expressions sample statements, 151-152
GROUP BY clause, 470-472
HAVING clause sample statements, 497-498
OUTER JOINS, 335-337
search conditions, 210
SELECT statements, 103-104
set sample statements, 629-633
subqueries
  expressions, 403
  filters, 410-411
UNION statements, 368-370
unlinked data sample statements, 703-704
unlinked tables sample statements, 695-696
UPDATE sample statements, 533-537

C
calculated columns, 135
calculated fields, 27
calculating mean values with AVG, 427-428
Call-Level Interface (CLI), 65
Cartesian Product, 303, 673
cascade deletion rule, 48
CASE, 641
  reasons for using, 642
  sample statements, 655-656
    bowling league database, 666-668
entertainment database, 659-662
sales order database, 656-659
school scheduling database, 662-665
Searched, 644
Simple, 644
solving problems, 647
Searched CASE, 652-654
Simple CASE, 647-651
syntax, 642-646
WHERE clause, 655
CAST function, changing data types, 114-116
Chamberlain, Dr., 58
changing data types, CAST function, 114-116
character data types, 111
column restrictions, 459-460
combining
result sets, unions, 236-238
sets, UNION, 245-247
comparing string values, comparison predicates, 163-166
data
grouping. See grouping data
versus information, 79-81
inserting with SELECT, 548-555
unlinked data. See unlinked data
data types, 111-113
approximate numeric data types, 112
binary data types, 112
Boolean data types, 113
changing, CAST function, 114-116
date and time arithmetic, 121
date and time arithmetic expressions, 129
date expressions, 129-131
time expressions, 131-133
date expressions, 129-131
SELECT clause, 138-139
DATE functions, 653
date literals, 119-120
datetime data types, 113
datetime literals, explicit values, 119-121
DB2 (Database 2), 60
defined pattern strings, samples, 176
DELETE, 571-572
rows
deleting, 573-574
deleting the correct rows, 575-576
samples, 580-581
bowling league database, 586-588
entertainment agency database, 582-584
sales orders database, 581-582
school scheduling database, 584-585
uses for, 579-580
deleting rows
with DELETE, 573-574
ensuring you delete the correct rows, 575-576
with subqueries, 577-579
with WHERE clause, 575
distinct, 89
DISTINCT keyword, 364
subqueries, 390
DISTINCT option, 425
don’t put a square peg in a round hole rule, 115
don’t put a ten-pound sack in a five-pound box rule, 115
driver tables
sample statements, 697
bowling league database, 703-704
entertainment database, 698-700
sales order database, 697-698
school scheduling database, 700-703
setting up, 679-682
solving problems, 679
using, 682-686
duplicate fields, resolving, 36-37, 39-41
duplicate rows, eliminating, 88-90
E
eliminating duplicate rows, 88-90
embedding
JOINs with JOINs, OUTER JOINs, 310-320
JOINs within JOINs in tables, 262-267
SELECT statements
OUTER JOINs, 307-310
in tables, 260-262
entertainers, 53
entertainment database
aggregate functions, 437
CASE sample statements, 659-662
DELETE sample statements, 582-584
driver tables sample statements, 698-700
expressions, 147-148
GROUP BY clause, 466-467
HAVING clause, 493-494
INSERT, 561-563
OUTER JOINs, 329-331
search conditions, 206-208
SELECT statements, 100-101
set sample statements, 622-624
subqueries
  expressions, 400
  filters, 410-411
UNIONs, 365
unlinked data sample statements, 698-700
unlinked tables sample statements, 689-691
UPDATE, 526-530
Entry SQL, 64
equality, comparison predicates, 166-167
ESCAPE option, 178
Euler diagram, 225
Euler, Leonard, 225
exact numeric data types, 112
EXCEPT, SQL set operations, 243-245
excluding rows, 191-193
  with NOT, search conditions, 181-184
executing queries, 97
EXISTS
  finding multiple matches in the same table, 612-614
  predicate keywords for subqueries, 395-396
expressing conditions, 203-204
expressions, 110
  concatenation, 121-125
data types. See data types
date and time arithmetic, 121, 129
date expressions, 129-131
time expressions, 131-133
grouping, GROUP BY clause, 461-462
mathematical, 121, 125-129
mixing with columns, GROUP BY clause, 454-456
sample statements, 144-145
  bowling league database, 151-152
  emergency agency database, 147-148
  sales orders database, 145-146
  school scheduling database, 149-150
SELECT clause, 133
  concatenation, 134-135
date expressions, 138-139
mathematical expressions, 137-138
naming expressions, 135-136
subqueries, sample statements
  bowling league database, 403
  entertainment agency database, 400
  recipes database, 403
  sales orders database, 399-400
  school scheduling database, 401
UPDATE, 507-508
  subqueries, 518-520
  updating selected rows, 508
value expressions, 139-141
F
field of vision, expanding, 85-87
fields, 23
calculated fields, 27
checklists for, 25-27
multipart fields, resolving, 27-29
multivalued fields, resolving, 30-32
naming, 23-25
relational databases, 9
filtering. See also search conditions
  focus groups, 478-480
  HAVING clause, 480-484
    samples. See sample statements, HAVING clause
      uses for, 490-491
  HAVING COUNT, 485-490
  multiple conditions, 184
    AND, 185, 189-190
    checking for overlapping ranges, 197-199
    excluding rows, 191-193
    OR, 185-190
    order of preference, 193-197
  rows, subqueries, 512-515
WHERE clause, 482-484
filters
  aggregate functions, 432-434
  subqueries, 384, 398
    predicate keywords, 386-396
    syntax, 384-386
subqueries, samples
  bowling league database, 410-411
  entertainment agency database, 406-407
  recipes database, 412-413
  sales orders database, 405-406
  school scheduling database, 408-409
finding
  largest values with MAX, 428-430
  matching values, INNER JOINs, 269
missing values
  EXCEPT, 243-245
  OUTER JOINs, 324
multiple matches in the same table, 607
  EXISTS, 612-614
  GROUP BY, 614-617
  HAVING, 614-617
  IN, 610-612
  INNER JOINs, 608-610
  partially matched information, OUTER JOINs, 325
related rows, INNER JOINs, 268-269
smallest values with MIN, 430-431
FIPS (Federal Information Processing Standard), 65
focus groups, filtering, 478-480
foreign keys, 10
FROM clause, 256
  SELECT statement, 78
unlinked tables, 672
FROM keyword, 573
FULL OUTER JOINs, 320
  non-key values, 323
  syntax, 320-322
Full SQL, 64
functions, aggregate, 420-422
  AVG, 427-428
  COUNT, 422-425
  filters, 432, 434
  MAX, 428-430
  MIN, 430-431
Null values, 451
samples. See sample statements, aggregate functions
  SUM, 425-426
  using multiple functions, 431-432
G
generating primary key values with INSERT, 547-548
greater than, comparison predicates, 168-170
GROUP BY clause, 448
  column restrictions, 459-460
  finding multiple matches in the same table, 614-617
  grouping on expressions, 461-462
  mixing columns and expressions, 454-456
  NOT, 604-607
  sample statements, 463
    bowling league database, 470-472
    entertainment agency database, 466-467
    recipes database, 473-474
    sales orders database, 464-465
    school scheduling database, 468-469
  SELECT statement, 79
  simulating SELECT DISTINCT statements, 457-458
  subqueries in WHERE clauses, 456-457
  syntax, 449-454
  uses for, 462-463
grouping data, 446-448
GROUP BY clause, 448
  column restrictions, 459-460
  grouping on expressions, 461-462
  mixing columns and expressions, 454-456
  samples. See sample statements, GROUP BY clause
  simulating SELECT DISTINCT statements, 457-458
  subqueries in WHERE clauses, 456-457
  syntax, 449-454
  uses for, 462-463
grouping expressions, GROUP BY clause, 461-462
H
HAVING clause, 480-482
filtering, 482-484
finding multiple matches in the same table, 614-617
NOT, 604-605, 607
sample statements, 491
  bowling league database, 497-498
  entertainment agency database, 493-494
recipes database, 498-499
sales orders database, 492-493
school scheduling database, 494-497
SELECT statement, 79
uses for, 490-491
HAVING COUNT, 485-490
history of relational databases, 4-5

I
IBM, origins of SQL, 58
IBM DB2, 733-735
IBM proprietary EBCDIC sequence, 164
IN
finding multiple matches in the same table, 610-612
predicate keywords for subqueries, 387-392
IN predicate, 160
inequality, comparison predicates, 166-167
information versus data, 79-81
INGRES (Interactive Graphics Retrieval System), 6, 60
INNER JOINs, 250
column references, 251-252
finding multiple matches in the same table, 608-610
samples, 269-270
matching values, 283-294
multiple tables, 276-283
two tables, 270-275
syntax, 252
tables, 253-256
tables, assigning correlation names, 258-260
tables, embedding JOINs within JOINs, 262-267
tables, embedding SELECT statements, 260-262
tables, relationships, 267-268
uses for
finding matching values, 269
finding related rows, 268-269
INSERT, 541-543
generating primary key values, 547-548
sample statements, 556-557
entertainment agency database, 561-563
sales orders database, 557-560
school scheduling database, 564-567
uses for, 555-556
values, 543-547
inserting
data with SELECT, 548-555
values, 543-545, 547
Interactive Graphics Retrieval System (INGRES), 6, 60
Intermediate SQL, 64
International Organization of Standardization (ISO), 62
INTERSECT, SQL set operations, 240-242
intersection, 221
intersections, set operations, 222-223
between result sets, 224-227
problems you can solve, 227-228
interval data types, 113
date expressions, 130
ISNULL predicate, 160
ISO (International Organization for Standardization), 62

J
JOIN eligible, 251
joining tables, INNER JOINs, 270-275
multiple tables, 276-283
JOINs, 227, 249-250
embedding with JOINs, OUTER JOINs, 310-320
embedding within JOINs, 262-267
INNER JOINs, 250
keywords, 256
NATURAL JOINs, 257
OUTER JOINs. See OUTER JOINs
UPDATE clause, 515-518
what can you join, 251

K
keys
foreign keys, 10
primary keys, 10
relational databases, 9-11
tables, 42-45
keywords
DISTINCT, 89, 364
subqueries, 390
FROM, 573
JOIN, 256
Index

predicate keywords for subqueries, 386
  ALL, 392-395
  ANY, 392-395
  EXISTS, 395-396
  IN, 387-392
  SOME, 392-395
  SELECT, 76

L
largest values, finding with MAX, 428-430
LEFT OUTER JOINs, 301, 309
less than, comparison predicates, 168-170
LIKE predicate, 160
linking columns, JOINs, 251
linking tables, 16
literal values, 116

M
many-to-many relationships, 14-16, 47
matches, finding in the same table, 607
  EXISTS, 612-614
  GROUP BY, 614-617
  HAVING, 614-617
  IN, 610-612
  INNER JOINs, 608-610
matching values, INNER JOINs
  finding with, 269
  samples, 283-294
mathematical expressions, 121, 125-129
  SELECT clause, 137-138
MAX
  finding largest values, 428-430
  finding smallest values, 430-431
  subqueries, 381-383
mean values, calculating with AVG, 427-428
Microsoft Office Access, 736-737
Microsoft SQL Server, 738
missing values, finding
  with EXCEPT, 243-245
  with OUTER JOINs, 324
mixing columns and expressions, GROUP BY
  clause, 454-456
multipart fields, resolving, 27-29
multiple conditions, 184
  AND, 185, 189-190
  checking for overlapping ranges, 197-199
  excluding rows, 191-193
  OR, 185-190
  order of precedence, 193-194
    less is more, 196-197
    prioritizing conditions, 194-196
multivalued fields, resolving, 30-32
MySQL, 740-741

N
naming
  expressions, SELECT clause, 135-136
  fields, 23-25
  tables, 33-35
national character data types, 111
NATURAL JOIN, 257
NIST (National Institute of Standards and Technology), 65
non-key values, FULL OUTER JOINs, 323
NOT, 597
  excluding rows with NOT, search conditions, 181-184
  GROUP BY, 604-607
  HAVING, 604-607
  NOT EXISTS, 603-604
  NOT IN, 601-603
  OUTER JOINs, 598-600
  sets, 595-596
  NOT EXISTS, 603-604
  NOT IN, 601-603
Null, 141-143, 199-203
  aggregate functions, 451
  problems with, 143-144
  search conditions, 179-181
numeric literals, explicit values, 118

O
ODBC, 65
one-to-many relationships, 14, 46
one-to-one relationships, 13, 46
operations, sets. See set operations
OR, 185-190
  result sets, 202
Oracle, 743-744
ORDER, SELECT statement, 92
ORDER BY clause, SELECT query, 93-96
ORDER BY clause, SELECT statement, 92
order of precedence, 126
order of preference, multiple conditions, 193-194
   less is more, 196-197
   prioritizing conditions, 194-196
organizational databases, 4
origins of SQL, 58-59
   early implementations, 59-60
   standardization, 60-62
OUTER JOINs, 234, 299-301
   FULL OUTER JOINs. See FULL OUTER JOINs
   LEFT OUTER JOINs, 301
   NOT, 598-600
   RIGHT OUTER JOINs, 301
samples, 325
   bowling league database, 335-337
   entertainment agency database, 329-331
   recipes database, 338-340
   sales orders database, 326-328
   school scheduling database, 331-334
syntax, 302
   tables, 302-307
   tables, embedding JOINs with JOINs, 310-320
   tables, embedding SELECT statements, 307-310
UNION JOINs, 323
uses for, 324
   finding missing values, 324
   finding partially matched information, 325
overlapping ranges, checking for, 197-199

P
parentheses
   mathematical expressions, 127
   prioritizing conditions, 195
partially matched information, finding with
   OUTER JOINs, 325
participation, relationships, 49-54
pattern match condition, 175-179
PC-based RDBMS programs, 6
performing UNIONs, 347
predicate keywords for subqueries, 386
   ALL, 392-395
   ANY, 392-395
   EXISTS, 395-396
IN, 387-392
   SOME, 392-395
predicates
   comparison predicates, 163
   comparing string values, 163-166
   equality and inequality, 166-167
   less than and greater than, 168-170
   range predicates, 170-173
   WHERE clause, 159-160
primary key values, generating with INSERT, 547-548
primary keys, 10
   tables, 42-44
   prioritizing conditions, 194-196
problems
   solving
      with driver tables, 679
      with unlinked data, 676-678
   solving with CASE, 647
      Searched CASE, 652-654
      Simple CASE, 647-651
Q
   quantified predicates, 392
   QUEL (Query Language), 60
   queries, executing, 97
   quotes, single quotes, 117
R
   range condition, 170-173
   ranges, checking for overlapping ranges, 197-199
   RDBMS (relational database management system), 5-7
   reasons for learning SQL, 69
   recipes database
      aggregate functions, 441-442
      GROUP BY clause, 473-474
      HAVING clause samples, 498-499
      OUTER JOINs, 325
      search conditions, 211-212
      SELECT statements, 105-106
      set sample statements, 633-635
      subqueries
         expressions, 403
         filters, 412-413
         UNIONs, 370-371
restrict deletion rule, 48
restrictions
column restrictions, GROUP BY clause, 459-460
data types, 116
grouping on expressions, GROUP BY clause, 461-462
result sets
combining with unions, 236-238
difference, 230-233
intersections between, 224-227
OR, 202
RIGHT OUTER JOINS, 301
row subqueries, 376-377
rows
counting with COUNT, 422-423
deleting
with DELETE, 573-574
with subqueries, 577-579
with WHERE clause, 575
duplicate rows, eliminating, 88-90
ensuring you delete the correct rows, 575-576
ensuring you’re updating the correct rows, UPDATE, 509
excluding, 191-193
filtering with subqueries, 512-515
finding related rows, INNER JOINs, 268-269
updating with UPDATE, 508
S
SAA, 65
sales order database
aggregate functions, 435-436
CASE sample statements, 656, 658-659
DELETE sample statements, 581-582
driver tables sample statements, 697-698
GROUP BY clause, 464-465
HAVING clause samples, 492-493
INSERT samples, 557-560
OUTER JOINs, 326-328
search conditions, 205-206
SELECT statements, 98-99
set sample statements, 618, 620-622
subqueries
expressions, 399-400
filters, 405-406
UNIONS, 360-364
unlinked data sample statements, 697-698
unlinked tables sample statements, 687-688
UPDATE samples, 522-526
sample statements
aggregate functions, 435
bowling league database, 439-440
entertainment agency database, 437
recipes database, 441-442
sales orders database, 435-436
school scheduling database, 438
CASE, 655-656
bowling league database, 666-668
entertainment database, 659-662
sales order database, 656, 658-659
school scheduling database, 662-665
defined pattern strings, 176
DELETE, 580-581
bowling league database, 586-588
entertainment agency database, 582-584
sales orders database, 581-582
school scheduling database, 584-585
driver tables, 697
bowling league database, 703-704
entertainment database, 698-700
sales order database, 697-698
school scheduling database, 700-703
expressions, 144-145
bowling league database, 151-152
emergency agency database, 147-148
sales orders database, 145-146
school scheduling database, 149-150
GROUP BY clause, 463
bowling league database, 470-472
entertainment agency database, 466-467
recipes database, 473-474
sales orders database, 464-465
school scheduling database, 468-469
HAVING clause, 491
bowling league database, 497-498
entertainment agency database, 493-494
recipes database, 498-499
sales orders database, 492-493
school scheduling database, 494-497
INNER JOINS, 269-270
matching values, 283-294
multiple tables, 276-283
two tables, 270-275
INSERT, 556-557
entertainment agency database, 561-563
sales orders database, 557-560
school scheduling database, 564-567
OUTER JOINS, 325
bowling league database, 335-337
entertainment agency database, 329-331
recipes database, 338-340
sales orders database, 326-328
school scheduling database, 331-334
search conditions, 204
bowling league database, 210
entertainment agency database, 206-208
recipes database, 211-212
sales orders database, 205-206
school scheduling database, 208-209
SELECT statements, 97
bowling league database, 103-104
entertainment agency database, 100-101
recipes database, 105-106
sales orders database, 98-99
school scheduling database, 102
sets, 618
bowling league database, 629-633
entertainment database, 622-624
recipes database, 633-635
sales orders database, 618-622
school scheduling database, 625-628
subqueries, expressions
bowling league database, 403
entertainment agency database, 400
recipes database, 403
sales orders database, 399-400
school scheduling database, 401
subqueries, filters
bowling league database, 410-411
entertainment agency database, 406-407
recipes database, 412-413
sales orders database, 405-406
school scheduling database, 408-409
UNIONS, 359
bowling league database, 368-370
entertainment agency database, 365
recipes database, 370-371
sales orders database, 360-364
school scheduling database, 366-367
unlinked data, 686
unlinked tables. See unlinked tables, sample statements
unlinked tables
   bowling league database, 695-696
   entertainment database, 689-691
   sales order database, 687-688
   school scheduling database, 691-695
UPDATE, 521-522
   bowling league database, 533-537
   entertainment agency database, 526-530
   sales orders database, 522-526
   school scheduling database, 530-533
saving SELECT statements, 96-97
scalar subqueries, 376-378
school scheduling database
   aggregate functions, 438
   CASE sample statements, 662-665
   DELETE sample statements, 584-585
   driver tables, 700-703
   expressions, 149-150
   GROUP BY clause, 468-469
   HAVING clause samples, 494-497
   INSERT sample statements, 564-567
   OUTER JOINs, 331-334
   search conditions, 208-209
   SELECT statements, 102
   subqueries
      expressions, 401
      filters, 408-409
set sample statements, 625-628
   UNIONs, 366-367
unlinked data sample statements, 700-703
unlinked tables sample statements, 691-695
UPDATE samples, 530-533
search conditions, 158, 254
   comparison predicates, 163
      comparing string values, 163-166
      equality and inequality, 166-167
      less than and greater than, 168-170
   excluding rows with NOT, 181-184
   Null condition, 179-181
   pattern match condition, 175-179
   range condition, 170-173
   samples, 204
      bowling league database, 210
      entertainment agency database, 206-208
      recipes database, 211-212
      sales orders database, 205-206
      school scheduling database, 208-209
   set membership condition, 173-175
   Searched CASE, 644
      solving problems, 652-654
   SELECT clause, 76
      expressions, 133
         concatenation, 134-135
         date expressions, 138-139
         mathematical expressions, 137-138
         naming, 135-136
         inserting data, 548-555
   SELECT statement, 78
   SELECT DISTINCT statements, simulating
      with GROUP BY clause, 457-458
   SELECT expression, 76
   SELECT operation, 76
   SELECT query, 76
      sorting, 91-92
         ORDER BY clause, 93-96
   SELECT statements, 76-77
      clauses, 77-79
   eliminating duplicate rows, 88-90
   embedding OUTER JOINs, 307-310
   embedding in tables, 260-262
   ORDER, 92
   ORDER BY clause, 92
   sample statements, 97
      bowling league database, 103-104
      entertainment agency database, 100-101
      recipes database, 105-106
      sales orders database, 98-99
      school scheduling database, 102
   saving, 96-97
   sorting, 91-92
   WHERE clause. See WHERE clause
   writing requests with UNIONs, 348-349
   complex statements, 351-354
SEQUEL-XRM, 58
sequences, collating, 92-93
set diagrams, 225
set membership condition, 173-175
set operations, 221
   difference, 228-230
      between result sets, 230-233
      problems you can solve, 233-234
   intersections, 222-223
      between result sets, 224-227
      problems you can solve, 227-228
SQL
   EXCEPT, 243-245
   INTERSECT, 240-242
   UNION, 245-247
   versus SQL set operations, 239
   unions, 234-235, 238
   combining result sets, 236-238
   problems you can solve, 238-239
   sets, 220-221
   combining with UNION, 245-247
   including some criteria but excluding others, 596-597
   with multiple AND criteria, 594-595
   with multiple NOT criteria, 595-596
   NOT. See NOT
   overview, 593-594
   sample statements, 618
   bowling league database, 629-633
   entertainment database, 622-624
   recipes database, 633-635
   sales orders database, 618-622
   school scheduling database, 625-628
   shortcuts, requesting all columns, 87-88
   Simple CASE, 644
      solving problems, 647-651
   single quotes, 117
   smallest values, finding with MIN, 430-431
   solving
      problems
         with driver tables, 679
         with unlinked data, 676-678
      problems with CASE, 647
         Searched CASE, 652-654
         Simple CASE, 647-651
   SOME, predicate keywords for subqueries, 392-395
   sorting
      SELECT query, 91-92
         ORDER BY clause, 93-96
      SELECT statement, 91-92
      UNIONS, 357-358
   SQL, origins of, 58-59
      early implementations, 59-60
      standardization, 60-62
   SQL set operations
      versus classic set operations, 239
      EXCEPT, 243-245
      INTERCEPT, 240-242
      UNION, 245-247
   SQL Standard, data types, 111
      restrictions, 116
   SQL standard diagrams. See Appendix A
   SQL standards, structure of, 66-68
   SQL/86, 61-63
   SQL/89, 62-63
   SQL/92, 63-64
   SQL/DS, 60
   SQUARE (Specifying Queries As Relational Expressions), 59
   standardization, 66
      ANSI/ISO standard, 62-65
      FIPS, 65
      ODBC, 65
      origins of SQL, 60-62
      SAA, 65
      X/Open, 65
   START TRANSACTION, 510
   statements
      DELETE. See DELETE
      INSERT. See INSERT
      SELECT DISTINCT statements. See SELECT
      DISTINCT statements
      UPDATE. See UPDATE
   Stonebraker, Michael, 60
   string values, comparing, 163-166
   structure of SQL standards, 66-68
   structures, 22
      fields, 23
         checklists for, 25-27
         naming, 23-25
         resolving multipart fields, 27-29
         resolving multivalued fields, 30-32
      relationships, 45, 48
         deletion rules, 48-49
         participation, 49-54
      tables, 32
         checklists for, 35-36
         keys, 42-45
         naming, 33-35
         resolving duplicate fields, 36-41
   subqueries, 376
      as column expressions, 378
         aggregate functions, 381-383
         syntax, 378-381
deleting rows, 577-579
filtering rows, UPDATE, 512-515
as filters, 384
  predicate keywords, 386-396
  syntax, 384-386
row subqueries, 376-377
sample statements, 399
  in expressions, bowling league database, 403
  in expressions, entertainment agency database, 400
  in expressions, recipes database, 403
  in expressions, sales orders database, 399-400
  in expressions, school scheduling database, 401
as filters, bowling league database, 410-411
as filters, entertainment agency database, 406-407
as filters, recipes database, 412-413
as filters, sales orders database, 405-406
as filters, school scheduling database, 408-409
scalar subqueries, 376-378
table subqueries, 376-377
UPDATE expressions, 518-520
uses for, 397-398
  WHERE clause, GROUP BY clause, 456-457
SUM, counting rows and values, 425-426
syntax
  FULL OUTER JOINs, 320-322
  INNER JOIN, tables, 252-256
    assigning correlation names, 258-260
    embedding JOINs within JOINs, 262-267
    embedding SELECT statements, 260-262
    relationships, 267-268
  OUTER JOINs, tables, 302-307
    embedding JOINs with JOINs, 310-320
    embedding SELECT statements, 307-310
System R, 6, 60

T
table subqueries, 376-377
tables, 32
  checklists for, 35-36
  driver tables. See driver tables
duplicate fields, resolving, 36-41
  INNER JOIN, 253-256
    assigning correlation names, 258-260
    embedding JOINs within JOINs, 262-267
    embedding SELECT statements, 260-262
    relationships, 267-268
    joining, samples, 270-275
    matching values, 283-294
    multiple tables, 276-283
  keys, 42-45
  linking tables, 16
  naming, 33-35
  OUTER JOINs, 302-307
    embedding JOINs with JOINs, 310-320
    embedding SELECT statements, 307-310
  relational databases, 7-8
  unlinked, 673-675
time expressions, 131-133
time literals, 119-120
timestamp literals, 119-120
totals, computing with SUM, 425-426
trailing blanks, 166
transactions, UPDATE, 510-511
  translating requests into SQL, 81-85
  expanding field of vision, 85-87
  using shortcuts to request all columns, 87-88

U
UNION JOINs, 323
UNIONs, 221, 234-235, 238, 345-348
  combining result sets, 236-238
  performing, 347
  problems you can solve, 238-239
  sample statements, 359
    bowling league database, 368-370
    entertainment agency database, 365
    recipes database, 370-371
    sales orders database, 360, 362-364
    school scheduling database, 366-367
SQL set operations, 245-247
  uses for, 358-359
  writing requests, 348
    complex SELECT statements, 351-354
    SELECT statements, 348-349
    sorting, 357-358
    using UNION more than once, 355-356
unlinked data, 672-674
  CROSS JOINs, deciding when to use, 675
driver tables. See driver tables
  sample statements, 686
  unlinked tables. See unlinked tables,
sample statements
  solving problems with, 676-678
driver tables, 679
unlinked tables, 673-675
  sample statements
    bowling league database, 695-696
    entertainment database, 689-691
    sales order database, 687-688
    school scheduling database, 691-695
UPDATE, 505-507
  ensure you’re updating the correct
  rows, 509
  expressions, 507-508
    subqueries, 518-520
    updating selected rows, 508
JOINs, 515-516, 518
  samples, 521-522
    bowling league database, 533-537
    entertainment agency database, 526-530
    sales order database, 522, 524-526
    school scheduling database, 530-533
  subqueries, filtering rows, 512-515
  transactions, 510-511
  updating multiple columns, 511-512
  uses for, 520-521
  updating
    multiple columns, 511-512
    selected rows, UPDATE, 508
USING clause, 256

W
WHERE clause, 157-159, 256
CASE, 655
deleting rows, 575
filtering, 482-484
predicates, 159-160
SELECT statement, 79
using, 160-162
  using GROUP BY in subqueries, 456-457
Wong, Eugene, 60
writing requests with UNIONs, 348
  complex SELECT statements, 351, 353-354
  SELECT statements, 348-349
  sorting, 357-358
  using UNION more than once, 355-356

X
X/OPEN, 65
X3H2, 61-62

V
value expressions, 139-141
values
  counting with COUNT, 424-425
  explicit values. See explicit values
  inserting, 543-547
  non-key values, FULL OUTER JOINs, 323
  Null, 141-143, 199-203
    problems with, 143-144
Venn diagram, 225
Venn, John, 225
views, relational databases, 11-12
Michael J. Hernandez’s best-selling Database Design for Mere Mortals®, has earned worldwide respect as the clearest, simplest way to learn relational database design. Now, he has made this hands-on, software-independent tutorial even easier, while ensuring that his design methodology is still relevant to the latest databases, applications, and best practices. Step by step, Database Design for Mere Mortals, Third Edition, shows you how to design databases that are soundly structured, reliable, and flexible, even in modern web applications. Hernandez guides you through everything from database planning to defining tables, fields, keys, table relationships, business rules, and views. You’ll learn practical ways to improve data integrity, how to avoid common mistakes, and when to break the rules.


For more information and sample content, visit informit.com. eBook and print formats available.