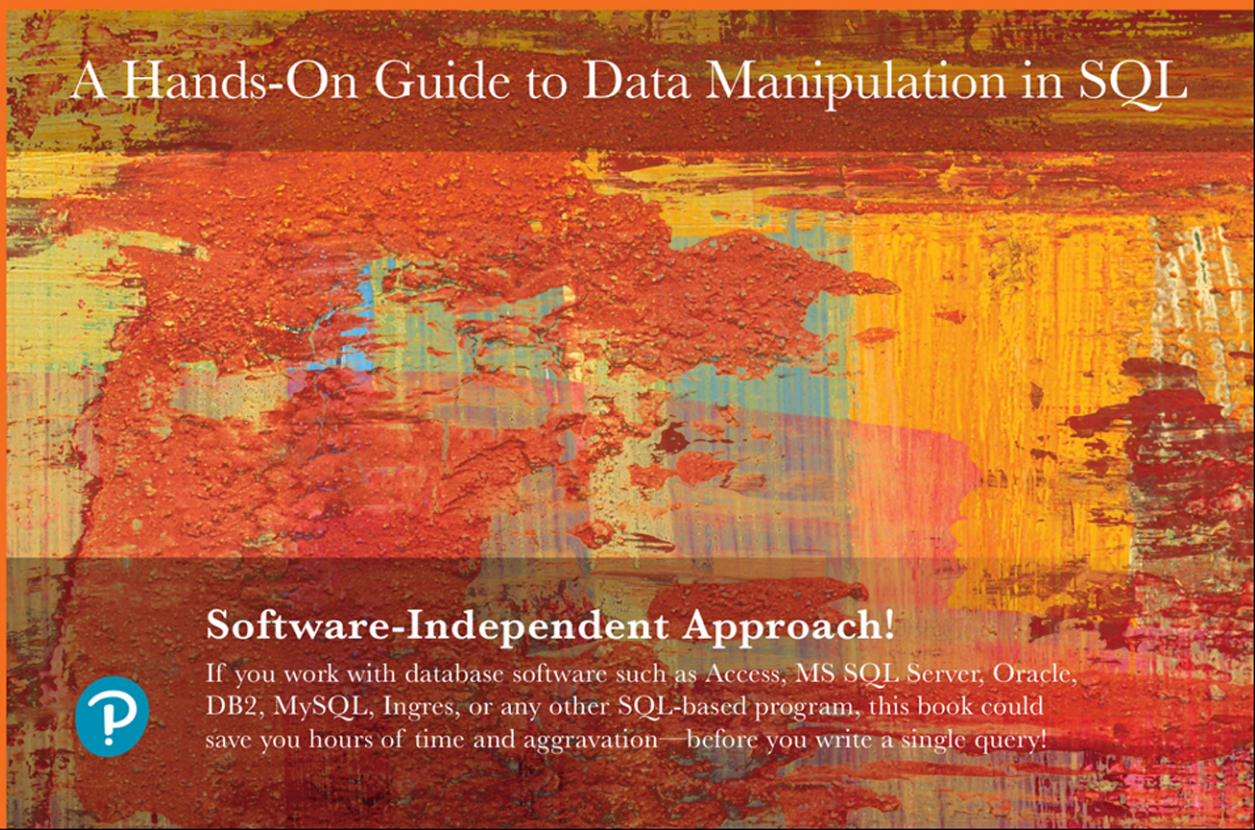
John L. Viescas

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

SQL Queries for Mere Mortals

Fourth Edition

A Hands-On Guide to Data Manipulation in SQL



Software-Independent Approach!

If you work with database software such as Access, MS SQL Server, Oracle, DB2, MySQL, Ingres, or any other SQL-based program, this book could save you hours of time and aggravation—before you write a single query!

Praise for SQL Queries for Mere Mortals®

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

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

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

— Graham Mandeno, Database Consultant

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

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

I learned SQL primarily from the first and second editions of this book…Starting from how to design your tables so that SQL can be effective (a common problem for database beginners), and then continuing through the various aspects of SQL construction and capabilities, the reader can become a moderate expert upon completing the book and its samples. Learning how to convert a question in English into a meaningful SQL statement will greatly facilitate your mastery of the language. Numerous examples from real life will help you visualize how to use SQL to answer the questions about the data in your

you'll see later, the SQL keyword UNION links the two queries to get the final answer.

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

Problems You Can Solve with Union

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

> *"Show me all the customer and employee names and addresses."*
>
> *"List all the customers who ordered a bicycle combined with all the customers who ordered a helmet."*
>
> *"List the entertainers who played engagements for customer Bonnicksen combined with all the entertainers who played engagements for customer Rosales."*
>
> *"Show me the students who have an average score of 85 or better in Art together with the students who have an average score of 85 or better in Computer Science."*
>
> *"Find the bowlers who had a raw score of 155 or better at Thunderbird Lanes combined with bowlers who had a raw score of 140 or better at Bolero Lanes."*
>
> *"Show me the recipes that have beef together with the recipes that have garlic."*

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

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

SQL Set Operations

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

Classic Set Operations versus SQL

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

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

Finding Common Values: INTERSECT

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

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

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

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

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

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

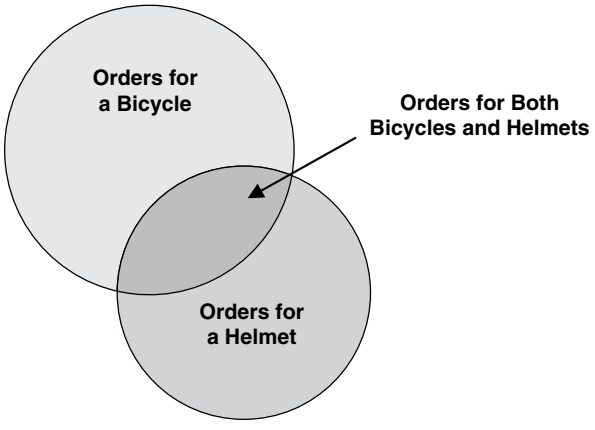


Figure 7-6 *One possible relationship between two sets of orders*

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

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

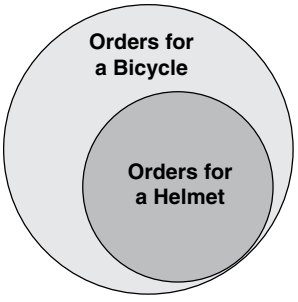


Figure 7-7 *All orders for a helmet also contain an order for a bicycle*

"Show me the orders that contain a bike."

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

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

"Show me the orders that contain a helmet."

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

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

Now you're ready to get the final solution by using—you guessed it—an *intersection* of the two sets. Figure 7-8 shows the SQL syntax diagram

that handles this problem. (Note that you can use INTERSECT more than once to combine multiple SELECT statements.)

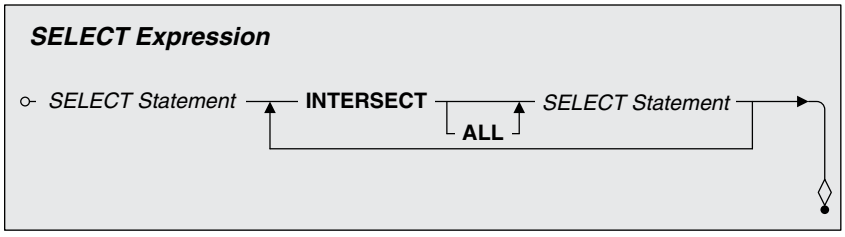


Figure 7-8 *Linking two SELECT statements with INTERSECT*

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

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

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

Finding Missing Values: EXCEPT (DIFFERENCE)

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

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

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

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

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

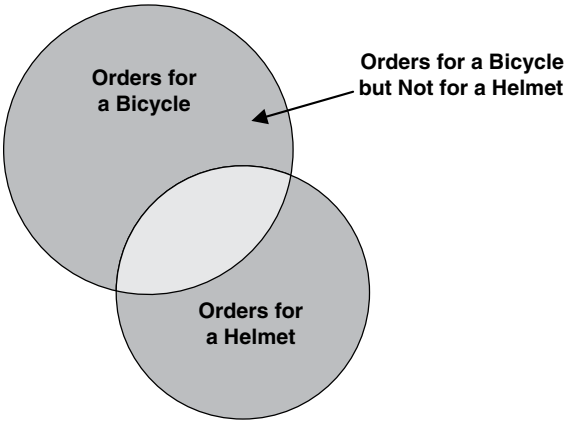


Figure 7-9 *Orders for a bicycle that do not also contain a helmet*

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