# THE LANGUAGE of SQL

SECOND EDITION

LARRY ROCKOFF

# The Language of SQL

## Second Edition

Larry Rockoff

Addison-Wesley

Hoboken, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Cape Town • Sydney • Tokyo • Singapore • Mexico City

**The Language of SQL, Second Edition**



ISBN-13: 978-0-13-465825-4

ISBN-10: 0-13-465825-6

Library of Congress Control Number: 2016945436

Printed in the United States of America

First Printing: August 2016

### Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. The publisher cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

### Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an “as is” basis. The author and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book.

### Special Sales

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the U.S., please contact intlcs@pearson.com.

**Editor**
Mark Taber

**Project and Copy Editor**
Dan Foster, Scribe Tribe

**Technical Editor**
Siddhartha Singh

**Designer**
Chuti Prasertsith

**Compositor**
Danielle Foster, Scribe Tribe

**Indexer**
Valerie Haynes Perry

**Proofreader**
Scout Festa

# Contents at a Glance

# Table of Contents

## About the Author

**Larry Rockoff** has been involved with SQL and business intelligence development for many years. His main area of interest is in using reporting tools to explore and analyze data in complex databases. He holds an MBA from the University of Chicago, with a specialization in management science. He currently works with data warehouse and reporting applications for a major retail pharmacy.

Besides writing about SQL, he has also published books on Microsoft Access and Excel.

He also maintains a website that features book reviews on technology topics, focusing on analytics and business intelligence as well as broader societal issues, at

larryrockoff.com

Please feel free to visit that site to contact the author with any comments or questions. You are also encouraged to follow his Facebook author page or Twitter site at

facebook.com/larryrockoff
twitter.com/larryrockoff

## Acknowledgments

A huge thanks goes out to all at Pearson who assisted with this book. I'd like to specifically thank Mark Taber, who was instrumental in bringing this book to Pearson from my previous publisher. I'd also like to thank project editor and copy editor Dan Foster, as well as Danielle Foster, who was responsible for the page layout. Siddhartha Singh did a superb job on the technical review. Chuti Prasertsith provided a wonderfully vibrant cover design. Finally, I must mention the generally thankless but essential tasks of the book's indexer, Valerie Haynes Perry, and proofreader, Scout Festa.

As this is a second edition, I'd also like to thank all readers of the first edition, and especially those individuals who have contacted me at larryrockoff.com and offered gracious comments as to the usefulness of the book in their personal lives. It's both humbling and thrilling to realize that your thoughts on a relatively mundane topic can assist someone halfway around the world.

## We Want to Hear from You!

As the reader of this book, *you* are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

We welcome your comments. You can email or write directly to let us know what you did or didn't like about this book—as well as what we can do to make our books better.

*Please note that we cannot help you with technical problems related to the topic of this book, and that due to the high volume of mail we receive, we might not be able to reply to every message.*

*When you write, please be sure to include this book's title and author, as well as your name and phone number or email address.*

Email: feedback@developers-library.info

Mail: Reader Feedback
Addison-Wesley Developer's Library
800 East 96th Street
Indianapolis, IN 46240 USA

## Reader Services

Visit our website and register this book at www.informit.com/register for convenient access to any updates, downloads, or errata that might be available for this book.

# Introduction

SQL, or Structured Query Language, is the primary language used to communicate with relational databases. The goal of this book is to serve as a useful introductory guide to this essential language.

In an alternate universe, the title of this book might have been *The Logic of SQL*. This is because, like all computer languages, the language of SQL has much more to do with cold hard logic than with English vocabulary. Nevertheless, the word *language* has been retained in the title for a number of reasons. First, a certain language-based syntax in SQL distinguishes it from other computer languages. Unlike other languages, SQL employs many ordinary words, such as WHERE and FROM, as keywords in its syntax.

In the spirit of the language embedded in SQL, we've adopted an emphasis on language in our sequence of topics. With this book, you'll learn SQL as you would learn English. SQL keywords are presented in a logical progression, from simple to more complex. In essence, this is an attempt to deal with language and logic simultaneously.

To learn any language, one must begin by hearing and remembering the actual words that form the basis of its utterance. At the same time, those words have a certain meaning that must be understood. In the case of SQL, the meaning has a great deal to do with logic.

One final reason for persisting with the title *The Language of SQL* rather than *The Logic of SQL* is that it simply sounds better. While there can be few literary pretensions in a computer-language book, the hope is that the presence of the word language will generate some additional enthusiasm for a subject that is, after all, quite interesting.

## Topics and Features

Even if you're not yet familiar with SQL, suffice it to say that it is a complex language with many components and features. In this book, we'll focus on one main topic:

- How to use SQL to retrieve data from a database

To a lesser extent, we will also cover:

- How to update data in a database
- How to build and maintain databases
- How to design relational databases
- Strategies for displaying data after it has been retrieved

A number of features make this book unique among introductory SQL books:

- **You will not be required to download software or sit with a computer as you read the text.**

  Our intent is to provide examples of SQL usage that can be understood simply by reading the book. The text includes small data samples that allow you to clearly see how SQL statements work.
- **A language-based approach is employed to enable you to learn SQL as you would learn English.**

  Topics are organized in an intuitive and logical sequence. SQL keywords are introduced one at a time, allowing you to build on your prior understanding as you encounter new words and concepts.
- **This book covers the syntax of three widely used databases: Microsoft SQL Server, MySQL, and Oracle.**

  If there are any differences between these databases, the Microsoft SQL Server syntax is shown in the main text. Special "Database Differences" sidebars show and explain any variations in the syntax for MySQL or Oracle.
- **An emphasis is given to relevant aspects of SQL for retrieving data.**

  This approach is useful for those who need only to use SQL in conjunction with a reporting tool. In our final chapter, we'll move beyond pure SQL to cover strategies for displaying data after it has been retrieved, including ideas on how to use crosstab reports and pivot tables. In the real world, these types of tools can substantially lessen the burden on the SQL developer and provide greater flexibility for the end user.

> **Note**
>
> Visit our website and register this book at informit.com/register for convenient access to downloads, updates, or errata that may be available for this book.

## What's New in the Second Edition

Here are some of the new features of this second edition:

- **Coverage of the latest database versions**

  All syntax and examples have been taken from the latest versions of the three main databases covered in this book: Microsoft SQL Server 2016, MySQL 5.7, and Oracle 12c.

- **Coverage of subtotals and crosstabs**

  We added a new chapter on subtotals and crosstabs to provide additional possibilities for summarizing data. This material allows users to add subtotals and running totals to their queries. Additionally, exposure to SQL crosstab queries allows for a greater appreciation of the value of pivot tables, which are covered toward the end of the book.
- **Coverage of ranking functions**

  We added new material on ranking functions. This important class of functions permits users to produce row numbers and calculate percentiles. A related capability is the ability to divide data into partitions prior to the application of ranking functions.
- **Expanded coverage of conditional logic**

  Our first edition included some basic material on the CASE expression and conditional logic. This topic is often excluded from introductory SQL books but nevertheless has tremendous practical value for any SQL developer. This second edition adds a number of new examples in Chapters 8 and 9 of how the CASE expression can be employed.
- **New and more consistent datasets**

  As in the first edition, each chapter has its own small set of data for examples. However, unlike before, this second edition now uses a consistent set of datasets for all chapters. A Customers table referenced in one chapter will be the same as a Customers table in any other chapter. If you wish to load sample data for testing, you can now run a single script to load all data at once. The previous edition required separate scripts for each chapter.
- **Other improvements**

  A few other noteworthy topics have been added, including common table expressions and comments. Terminology throughout the book has been modified for greater consistency and conformity with standard usage. Finally, an overview of the SELECT statement has been added early in Chapter 2, to give you a better idea of the topics to come.

## Plan of the Book

This book presents its topics in a unique sequence. The majority of SQL books run through their topics as if you were a database administrator who needs to create and design a database from scratch, then load the database with data, and then finally start to retrieve that data. In this book, we start right off with data retrieval, and then come around to database design in the final chapters. This is done as a motivational tactic, allowing you to quickly get into interesting topics related to data retrieval before having to deal with the more arcane subjects of indexes and foreign keys.

The 20 chapters in the book can be broken down into a number of broad sections:

- Chapter 1 presents introductory material about relational databases that is necessary to understand before encountering the SELECT statement.
- Chapters 2 through 5 begin an exploration of the SELECT statement, covering the basics of calculations, functions, and sorting.

- Chapters 6 through 8 deal with selection criteria, from simple Boolean logic to conditional logic.
- Chapters 9 and 10 explore ways to summarize data, from simple counts to more complex aggregations and subtotals.
- Chapters 11 through 15 discuss ways to retrieve data from multiple tables via joins, subqueries, views, and set logic.
- Chapters 16 through 18 move beyond the SELECT statement to focus on broader topics associated with relational databases, such as stored procedures, updates, and table maintenance.
- Finally, Chapters 19 and 20 bring us back to the basics of database design and then to strategies for displaying data.

Appendixes A, B, and C provide information on how to get started with each of the three databases covered in the book: Microsoft SQL Server, Oracle, and MySQL.

## Companion Website

A listing of all SQL statements in this book can be found at this site:

- **www.informit.com/store/language-of-sql-9780134658254**

These three files are provided:

- SQL Statements for Microsoft SQL Server
- SQL Statements for MySQL
- SQL Statements for Oracle

These three files list all SQL statements in the book for each of these databases. Additionally, these files contain a SQL script that allows you to create all the data used in the book. After running the setup script, you can execute statements in the book and see the same output.

Instructions on how to execute the setup script are provided within each of the files.

# 4

# Using Functions

**Keywords Introduced**

LEFT · RIGHT · SUBSTRING · LTRIM · RTRIM · UPPER · LOWER · GETDATE · DATEPART · DATEDIFF · ROUND · PI · POWER · ISNULL

Anyone familiar with Microsoft Excel is probably aware that functions provide a huge amount of functionality for the typical spreadsheet user. Without the ability to use functions, most of the data available in spreadsheets would be of limited value. The same is true in the world of SQL. Familiarity with SQL functions will greatly enhance your ability to generate dynamic results for anyone viewing data or reports generated from SQL.

This chapter covers a wide variety of some of the most commonly used functions in four different categories: character functions, date/time functions, numeric functions, and conversion functions. Additionally, we'll talk about composite functions—a way of combining multiple functions into a single expression.

## What Is a Function?

Similar to the calculations covered in the previous chapter, functions provide another way to manipulate data. As was seen, calculations can involve multiple fields, either with arithmetic operators such as multiplication, or by concatenation. Similarly, functions can involve data from multiple values, but the end result of a function is always a single value.

What is a function? A function is merely a rule for transforming any number of input values into one output value. The rule is defined within the function and can't be altered. However, the user of a function is allowed to specify any desired value for the inputs to the function. Some functions may allow some of the inputs to be optional. That means that the specification of that particular input isn't required. Functions can also be designed to have no inputs. However, regardless of the type or number of input values, functions always return precisely one output value when the function is invoked.

There are two types of functions: scalar and aggregate. The term *scalar* comes from mathematics and refers to an operation that is done on a single number. In computer usage, it means that the function is performed on data in a single row. For example, the LTRIM function removes spaces from one specified value in one row of data.

In contrast, aggregate functions are meant to be performed on a larger set of data. For example, the SUM function can be used to calculate the sum of all the values of a specified column. Because aggregate functions apply to larger sets or groups of data, we will leave discussion of this type of function to Chapter 9, “Summarizing Data.”

Every SQL database offers dozens of scalar functions. The actual functions vary widely between databases, in terms of both their names and how they work. As a result, we will cover only a few representative examples of some of the more useful functions.

The most common types of scalar functions can be classified under three categories: character, date/time, and numeric. These are functions that allow you to manipulate character, date/time, or numeric datatypes. In addition, we will talk about some useful conversion functions that can be used to convert data from one datatype to another.

## Character Functions

Character functions are those that enable you to manipulate character data. Just as character datatypes are sometimes called *string datatypes*, character functions are sometimes called *string functions*. We’ll cover these seven examples of character functions: LEFT, RIGHT, SUBSTRING, LTRIM, RTRIM, UPPER, and LOWER.

In this chapter, rather than retrieving data from specific tables, we’ll simply use SELECT statements with literal values in the *columnlist*. There will be no FROM clause to indicate a table. Let’s start with an example for the LEFT function. When this SQL command is issued:

```
SELECT
LEFT('sunlight',3) AS 'The Answer'
```

this data is returned:

| The Answer |
|---|
| sun |

The inclusion of a column alias in this SQL statement allows the output to display “The Answer” as a column header. Note that there is no FROM clause in the SELECT statement. Instead of retrieving data from a table, we’re selecting data from a single literal value, namely ‘sunlight’. In many SQL implementations, including SQL Server and MySQL, a FROM clause isn’t strictly necessary in a SELECT statement, although in practice one would seldom write a SELECT statement like this. We’re using this format, without a FROM clause, only to more easily illustrate how functions work.

Let's now look at the format of this function in greater detail. The general format of the LEFT function is:

```
LEFT(CharacterValue, NumberOfCharacters)
```

All functions have any number of arguments within the parentheses. For example, the LEFT function has two arguments: *CharacterValue* and *NumberOfCharacters*. The term *argument* is a commonly used mathematical term that describes a component of functions, and has nothing to do with anything being disagreeable or unpleasant. The various arguments that are defined for each function are what truly define the meaning of the function. In the case of the LEFT function, the *CharacterValue* and *NumberOfCharacters* arguments are both needed to define what will happen when the LEFT function is invoked.

The LEFT function has two arguments, and both are required. As mentioned, other functions may have more or fewer arguments. Functions are even permitted to have no arguments. But regardless of the number of arguments, even if zero, all functions have a set of parentheses following the function name. The presence of the parentheses tells you that the expression is a function and not something else.

The formula for the LEFT function says: Take the specified *CharacterValue*, look at the specified *NumberOfCharacters* on the left, and bring back the result. In the previous example, it looks at the CharacterValue 'sunlight' and brings back the left three characters. The result is "sun".

The main point to remember is that for any function you want to use, you'll need to look up the function in the database's reference guide and determine how many arguments are required and what they mean.

The second character function we'll cover is the RIGHT function. This is the same as the LEFT function, except that characters are now specified for the right side of the input value. The general format of the RIGHT function is:

```
RIGHT(CharacterValue, NumberOfCharacters)
```

As an example:

```
SELECT
RIGHT('sunlight',5) AS 'The Answer'
```

returns:

| The Answer |
|---|
| light |

In this case, the *NumberOfCharacters* argument needed to have a value of 5 in order to return the value "light". A value of 3 would have only returned "ght".

One problem that often arises with the use of the RIGHT function is that character data often contains spaces on the right-hand side. Let's look at an example in which a table with only one row of data contains a column named President, where the column is defined as being 20 characters long. The table looks like:

| President |
|---|
| George Washington |

If we issue this SELECT statement against the table:

```
SELECT
RIGHT(President,10) AS 'Last Name'
FROM table1
```

we get back this data:

| Last Name |
|---|
| hington |

We expected to get back "Washington" but only got "hington." The problem is that the entire column is 20 characters long. In this example, there are three spaces to the right of the value "George Washington". Therefore, when we ask for the rightmost 10 characters, SQL will take the three spaces, plus another seven characters from the original expression. As will soon be seen, the function RTRIM must be used to remove the ending spaces before using the RIGHT function.

You might be wondering how to select data from the middle of an expression. This is accomplished by using the SUBSTRING function. The general format of that function is:

```
SUBSTRING(CharacterValue, StartingPosition, NumberOfCharacters)
```

For example:

```
SELECT
SUBSTRING('thewhitegoat',4,5) AS 'The Answer'
```

returns this data:

| The Answer |
|---|
| white |

This function is saying to take five characters, starting with position 4. This results in the display of the word "white".

### Database Differences: MySQL and Oracle

MySQL sometimes requires that there be no space between the function name and the left parenthesis. It depends on the specific function used. For example, the previous statement in MySQL must be written exactly as shown. Unlike in Microsoft SQL Server, you can't type in an extra space after SUBSTRING.

In Oracle, the equivalent of the SUBSTRING function is SUBSTR. One difference in the Oracle version of SUBSTR is that the second argument (*StartingPosition*) can have a negative value. A negative value for this argument means that you need to count that number of positions backward from the right side of the column.

As mentioned, Oracle doesn't permit you to write a SELECT statement without a FROM clause. However, Oracle does provide a dummy table called DUAL for this type of situation. The Oracle equivalent of the SELECT with a SUBSTRING function is:

```
SELECT
SUBSTR('thewhitegoat',4,5) AS "The Answer"
FROM DUAL;
```

Our next two character functions enable us to remove all spaces, either on the left or the right side of an expression. The LTRIM function trims characters from the left side of a character expression. For example:

```
SELECT
LTRIM('     the apple') AS 'The Answer'
```

returns this data:

| The Answer |
|---|
| the apple |

Note that LTRIM is smart enough not to eliminate spaces in the middle of a phrase. It only removes the spaces to the very left of a character value.

Similar to LTRIM, the RTRIM function removes any spaces to the right of a character value. An example of RTRIM will be given in the next section, on composite functions.

The final two character functions to be covered are UPPER and LOWER. These functions convert any word or phrase to upper- or lowercase. The syntax is simple and straightforward. Here's an example that covers both functions:

```
SELECT
UPPER('Abraham Lincoln') AS 'Convert to Uppercase',
LOWER('ABRAHAM LINCOLN') AS 'Convert to Lowercase'
```

The output is:

| Convert to Uppercase | Convert to Lowercase |
|---|---|
| ABRAHAM LINCOLN | abraham lincoln |

## Composite Functions

An important characteristic of functions, whether they are character, mathematical, or date/time, is that two or more functions can be combined to create composite functions. A composite function with two functions can be said to be a function of a function. Let's go back to the George Washington query to illustrate. Again, we're working from this data:

| President |
|---|
| George Washington |

Remember that the President column is 20 characters long. In other words, there are three spaces to the right of the value "George Washington". In addition to illustrating composite functions, this next example will also cover the RTRIM function mentioned in the previous section. The statement:

```
SELECT
RIGHT(RTRIM (President),10) AS 'Last Name'
FROM table1
```

returns this data:

| Last Name |
|---|
| Washington |

Why does this now produce the correct value? Let's examine how this composite function works. There are two functions involved: RIGHT and RTRIM. When evaluating composite functions, you always start from the inside and work your way out. In this example, the innermost function is:

```
RTRIM(President)
```

This function takes the value in the President column and eliminates all spaces on the right. After this is done, the RIGHT function is applied to the result to bring back the desired value. Because

```
RTRIM(President)
```

equals "George Washington", we can say that:

```
SELECT
RIGHT(RTRIM (President), 10)
```

is the same as saying:

```
SELECT
RIGHT('George Washington', 10)
```

In other words, we can obtain the desired result by first applying the RTRIM function to the input data and then adding the RIGHT function to the expression to produce the final results.

## Date/Time Functions

Date/Time functions allow for the manipulation of date and time values. The names of these functions differ, depending on the database used. In Microsoft SQL Server, the functions we'll cover are called GETDATE, DATEPART, and DATEDIFF.

The simplest of the date/time functions is one that returns the current date and time. In Microsoft SQL Server, the function is named GETDATE. This function has no arguments. It merely returns the current date and time. For example:

```
SELECT GETDATE()
```

brings back an expression with the current date and time. Since the GETDATE function has no arguments, there is nothing specified between the parentheses. Remember that a date/time field is a special datatype that contains both a date and a time in a single field. An example of such a value is:

```
2017-05-15 08:48:30
```

This value refers to the 15th of May 2017, at 48 minutes and 30 seconds past 8 AM.

### Database Differences: MySQL and Oracle

In MySQL, the equivalent of GETDATE is NOW. The above statement would be written as:

```
SELECT NOW()
```

The equivalent of GETDATE in Oracle is CURRENT_DATE. The statement is written as:

```
SELECT CURRENT_DATE
```

The next date/time function enables us to analyze any specified date and return a value to represent such elements as the day or week of the date. Again, the name of this function differs, depending on the database. In Microsoft SQL Server, this function is called DATEPART. The general format is:

```
DATEPART(DatePart, DateValue)
```

The *DateValue* argument is any date. The *DatePart* argument can have many different values, including year, quarter, month, dayofyear, day, week, weekday, hour, minute, and second.

The following chart shows how the DATEPART function evaluates the date '5/6/2017', with different values for the *DatePart* argument:

| DATEPART Function Expression | Resulting Value |
|---|---|
| DATEPART(month, '5/6/2017') | 5 |
| DATEPART(day, '5/6/2017') | 6 |
| DATEPART(week, '5/6/2017') | 18 |
| DATEPART(weekday, '5/6/2017') | 7 |

Looking at the values in the previous chart, you can see that the month of 5/6/2017 is 5 (May). The day is 2 (Monday). The week is 18, because 5/6/2017 is in the 18th week of the year. The weekday is 7 because 5/6/2017 falls on a Saturday, which is the seventh day of the week.

> **Database Differences: MySQL and Oracle**
>
> In MySQL, the equivalent of the DATEPART function is named DATE_FORMAT, and it utilizes different values for the *DateValue* argument. For example, to return the day of the date '5/6/2017', you would issue this SELECT in MySQL:
>
> ```
> SELECT DATE_FORMAT('2017-05-06', '%d');
> ```
>
> Oracle doesn't have a function comparable to DATEPART.

The final date/time function we'll cover, DATEDIFF, enables you to determine quantities such as the number of days between any two dates. The general format is:

```
DATEDIFF (DatePart, StartDate, EndDate)
```

Valid values for the *DatePart* argument for this function include year, quarter, month, dayofyear, day, month, hour, minute, and second. Here's a chart that shows how the DATEDIFF function evaluates the difference between the dates 7/8/2017 and 8/14/2017, with different values for the DatePart argument:

| DATEPART Function Expression | Resulting Value |
|---|---|
| DATEDIFF(day, '7/8/2017', '8/14/2017') | 37 |
| DATEDIFF(week, '7/8/2017', '8/14/2017') | 6 |
| DATEDIFF(month, '7/8/2017', '8/14/2017') | 1 |
| DATEDIFF(year, '7/8/2017', '8/14/2017') | 0 |

The above chart indicates that there are 37 days, or 6 weeks, or 1 month, or 0 years between the two dates.

> **Database Differences: MySQL and Oracle**
>
> In MySQL, the DATEDIFF function only allows you to calculate the number of days between the two dates, and the end date must be listed first to return a positive value. The general format is:
>
> ```
> DATEDIFF(EndDate, StartDate)
> ```
>
> Oracle doesn't have a function comparable to DATEDIFF.

## Numeric Functions

Numeric functions allow for manipulation of numeric values. Numeric functions are sometimes called *mathematical functions*. The functions we'll cover are ROUND, RAND, PI, and POWER.

The ROUND function allows you to round any numeric value. The general format is:

```
ROUND(NumericValue, DecimalPlaces)
```

The *NumericValue* argument can be any positive or negative number, with or without decimal places, such as 712.863 or –42. The *DecimalPlaces* argument is trickier. It can contain a positive or negative integer, or zero. If *DecimalPlaces* is a positive integer, it means to round to that many decimal places. If *DecimalPlaces* is a negative integer, it means to round to that number of positions to the left of the decimal place. The following chart shows how the number 712.863 is rounded, with different values for the *DecimalPlaces* argument.

| ROUND Function Expression | Resulting Value |
|---|---|
| ROUND(712.863, 3) | 712.863 |
| ROUND(712.863, 2) | 712.860 |
| ROUND(712.863, 1) | 712.900 |
| ROUND(712.863, 0) | 713.000 |
| ROUND(712.863, –1) | 710.000 |
| ROUND(712.863, –2) | 700.000 |

The PI function merely returns the value of the mathematical number pi. As you may remember from high school geometry, the number pi is an irrational number approximated by the value 3.14. This function is seldom used, but nicely illustrates the point that numeric functions need not have any arguments. For example, the statement:

```
SELECT PI()
```

returns the value 3.14159265358979. To take this example a little further, let's say that we want the value of pi rounded to two decimal places. This can be accomplished by creating a composite function with the PI and ROUND functions. The PI function is used to get the initial value, and the ROUND function is added to round it to two decimal places. The following statement returns a value of 3.14:

```
SELECT ROUND(PI(),2)
```

> **Database Differences: Oracle**
>
> Unlike Microsoft SQL Server and MySQL, Oracle doesn't have a PI function.

The final numeric function we'll cover, which is much more commonly used than PI, is POWER. The POWER function is used to specify a numeric value that includes exponents. The general format of the function is:

```
POWER(NumericValue, Exponent)
```

Let's start with an example that illustrates how to display the number 5 raised to the second power. This is commonly referred to as "5 squared." The SELECT statement:

```
SELECT POWER(5,2) AS '5 Squared'
```

returns this data:

| 5 Squared |
|---|
| 25 |

In this example, 5 is the numeric value to be evaluated, and 2 is the value of the exponent. Remembering that the square root of a number can be expressed as an exponent with a decimal value less than 1, we can calculate the square root of 25 as follows. The statement:

```
SELECT POWER(25,.5) AS 'Square Root of 25'
```

returns this data:

| Square Root of 25 |
|---|
| 5 |

In algebraic terms, the calculation takes 25 to the 1/2 (or .5) power. This is the same as taking the square root of 25.

## Conversion Functions

All of the aforementioned functions pertain to specific ways to manipulate character, date/time, or numeric datatypes. We now want to address the need to convert data from one datatype to another, or to convert NULL values to something meaningful. The remainder of this chapter will cover two special functions that can be used in these situations.

The CAST function converts data from one datatype to another. The general format of the function is:

```
CAST(Expression AS DataType)
```

The format of this function is slightly different from other functions previously seen, as it uses the word AS to separate the two arguments, rather than a comma. Looking at the usage of the function, it turns out that the CAST function is unnecessary in most situations. Let's take the situation where we want to execute this statement, where the Quantity column is defined as a character datatype:

```
SELECT
2 * Quantity
FROM table
```

Your first impression might be that the statement would fail, due to the fact that Quantity is not defined as a numeric column. However, most SQL databases are smart enough to automatically convert the Quantity column to a numeric value so that it can be multiplied by 2.

Here's an example where the CAST function becomes necessary. Let's say we have dates stored in a column with a character datatype. We'd like to convert those dates to a true date/time column. This statement illustrates how the CAST function can handle that conversion:

```
SELECT
'2017-04-11' AS 'Original Date',
CAST('2017-04-11' AS DATETIME) AS 'Converted Date'
```

The output is:

| Original Date | Converted Date |
|---|---|
| 2017-04-11 | 2017-04-11 00:00:00 |

The Original Date column looks like a date, but it is really just character data. In contrast, the Converted Date column is a true date/time column, as evidenced by the time value now shown.

A second useful conversion function is one that converts NULL values to a meaningful value. In Microsoft SQL Server, the function is called ISNULL. As mentioned in Chapter 1, "Relational Databases and SQL," NULL values are those for which there is an absence of data. A NULL value is not the same as a space or zero. Let's say we have this table of products:

| ProductID | Description | Weight |
|---|---|---|
| 1 | Printer A | NULL |
| 2 | Printer B | 0 |
| 3 | Monitor C | 2 |
| 4 | Laptop D | 4 |

Notice that Printer A has a value of NULL in the Weight column. This indicates that a weight for this printer has not yet been provided. Let's say we want to produce a list of all products. When this SELECT is issued:

```
SELECT
Description,
Weight
FROM Products
```

It will show:

| Description | Weight |
|---|---|
| Printer A | NULL |
| Printer B | 0 |
| Monitor C | 2 |
| Laptop D | 4 |

There's nothing inaccurate about this display. However, users may prefer to see something such as "Unknown" rather than NULL for missing values. Here's the solution:

```
SELECT
Description,
ISNULL(CAST(Weight AS VARCHAR),'Unknown') AS Weight
FROM Products
```

The following data is displayed:

| Description | Weight |
|---|---|
| Printer A | Unknown |
| Printer B | 0 |
| Monitor C | 2 |
| Laptop D | 4 |

Notice that the solution requires the use of both the ISNULL and CAST functions. The ISNULL function handles the display of the weight as "Unknown" when NULL values are encountered. Assuming the Weight column is defined as an integer, the CAST function is needed to convert the weight to a Varchar datatype, so both integer and character values can be displayed in a single column.

### Database Differences: MySQL and Oracle

The ISNULL function is called IFNULL in MySQL. Furthermore, MySQL doesn't require the use of the CAST function in this example. The equivalent of the above statement in MySQL is:

```
SELECT
Description,
IFNULL(Weight,'Unknown') AS Weight
FROM Products;
```

The ISNULL function is called NVL (Null Value) in Oracle. The equivalent Oracle statement is:

```
SELECT
Description,
NVL(CAST(Weight AS CHAR),'Unknown') AS Weight
FROM Products;
```

Additionally, unlike Microsoft SQL Server and MySQL, Oracle displays a dash rather than the word NULL when it encounters NULL values.

## Looking Ahead

This chapter described a wide variety of functions. Functions are basically predefined rules for transforming a set of values into another value. Just as spreadsheets provide built-in functions for manipulating data, SQL provides similar capabilities. In addition to covering basic character, date/time, numeric, and conversion functions, we also explained how to create composite functions from two or more of these functions.

Because there are simply so many available functions with widely varying possibilities, it's impossible to discuss every nuance of every available function. The thing to remember is that functions can be easily looked up in a database's help system or reference guide when they need to be used. Online reference material will provide details on exactly how each function works and the proper syntax.

In our next chapter, we'll take a break from *columnlist* issues and talk about something a little more interesting: how to sort data. Sorts can serve lots of useful purposes and satisfy the basic desire of users to view data in some type of order. With the sort, we will begin to think of the entire way in which information is presented, rather than with just bits and pieces of individual data items.

*This page intentionally left blank*

# Index

## Symbols



## A



## B

## C



## D

## P



## Q



## R