



# THE LANGUAGE **of SQL**

SECOND EDITION

LARRY ROCKOFF

FREE SAMPLE CHAPTER

SHARE WITH OTHERS



# 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**

Copyright © 2017 by Pearson Education, Inc.

All rights reserved. No part of this book shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the publisher. No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions. Nor is any liability assumed for damages resulting from the use of the information contained herein.

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](mailto:corpsales@pearsoned.com) or (800) 382-3419.

For government sales inquiries, please contact [governmentsales@pearsoned.com](mailto:governmentsales@pearsoned.com).

For questions about sales outside the U.S., please contact [intlcs@pearson.com](mailto: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

Introduction	xiii
1 Relational Databases and SQL	1
2 Basic Data Retrieval	11
3 Calculated Fields and Aliases	19
4 Using Functions	27
5 Sorting Data	41
6 Selection Criteria	49
7 Boolean Logic	61
8 Conditional Logic	73
9 Summarizing Data	81
10 Subtotals and Crosstabs	101
11 Inner Joins	115
12 Outer Joins	123
13 Self Joins and Views	135
14 Subqueries	143
15 Set Logic	155
16 Stored Procedures and Parameters	163
17 Modifying Data	171
18 Maintaining Tables	181
19 Principles of Database Design	189
20 Strategies for Displaying Data	199
A Getting Started with Microsoft SQL Server	209
B Getting Started with MySQL	211
C Getting Started with Oracle	215
Index	217

# Table of Contents

<b>Introduction</b>	<b>xiii</b>
<b>1 Relational Databases and SQL</b>	<b>1</b>
What Is SQL?	2
Microsoft SQL Server, MySQL, and Oracle	3
Relational Databases	4
Primary and Foreign Keys	6
Datatypes	6
NULL Values	8
The Significance of SQL	8
Looking Ahead	9
<b>2 Basic Data Retrieval</b>	<b>11</b>
A Simple SELECT	11
Syntax Notes	12
Comments	13
Specifying Columns	14
Column Names with Embedded Spaces	15
Preview of the Full SELECT	16
Looking Ahead	17
<b>3 Calculated Fields and Aliases</b>	<b>19</b>
Literal Values	20
Arithmetic Calculations	21
Concatenating Fields	22
Column Aliases	23
Table Aliases	24
Looking Ahead	25
<b>4 Using Functions</b>	<b>27</b>
What Is a Function?	27
Character Functions	28
Composite Functions	32
Date/Time Functions	33

Numeric Functions	35
Conversion Functions	36
Looking Ahead	39

## **5 Sorting Data 41**

Sorting in Ascending Order	41
Sorting in Descending Order	43
Sorting by Multiple Columns	43
Sorting by a Calculated Field	44
Sort Sequences	45
Looking Ahead	47

## **6 Selection Criteria 49**

Applying Selection Criteria	49
WHERE Clause Operators	50
Limiting Rows	51
Limiting Rows with a Sort	53
Pattern Matching	54
Wildcards	56
Looking Ahead	58

## **7 Boolean Logic 61**

Complex Logical Conditions	61
The AND Operator	62
The OR Operator	62
Using Parentheses	63
Multiple Sets of Parentheses	65
The NOT Operator	66
The BETWEEN Operator	68
The IN Operator	69
Boolean Logic and NULL Values	70
Looking Ahead	72

## **8 Conditional Logic 73**

The CASE Expression	73
The Simple CASE Format	74
The Searched CASE Format	76

Conditional Logic in ORDER BY Clauses 78  
Conditional Logic in WHERE Clauses 79  
Looking Ahead 80

**9 Summarizing Data 81**

Eliminating Duplicates 81  
Aggregate Functions 83  
The COUNT Function 84  
Grouping Data 86  
Multiple Columns and Sorting 87  
Selection Criteria on Aggregates 89  
Conditional Logic in GROUP BY Clauses 91  
Conditional Logic in HAVING Clauses 92  
Ranking Functions 93  
Partitions 97  
Looking Ahead 100

**10 Subtotals and Crosstabs 101**

Adding Subtotals with ROLLUP 102  
Adding Subtotals with CUBE 106  
Creating Crosstab Layouts 110  
Looking Ahead 114

**11 Inner Joins 115**

Joining Two Tables 116  
The Inner Join 118  
Table Order in Inner Joins 119  
An Alternate Specification of Inner Joins 119  
Table Aliases Revisited 120  
Looking Ahead 121

**12 Outer Joins 123**

The Outer Join 123  
Left Joins 125  
Testing for NULL Values 127  
Right Joins 128

Table Order in Outer Joins	129
Full Joins	129
Cross Joins	131
Looking Ahead	134
<b>13 Self Joins and Views</b>	<b>135</b>
Self Joins	135
Creating Views	137
Referencing Views	139
Benefits of Views	140
Modifying and Deleting Views	141
Looking Ahead	142
<b>14 Subqueries</b>	<b>143</b>
Types of Subqueries	143
Using a Subquery as a Data Source	144
Using a Subquery in Selection Criteria	147
Correlated Subqueries	148
The EXISTS Operator	150
Using a Subquery as a Calculated Column	151
Common Table Expressions	152
Looking Ahead	153
<b>15 Set Logic</b>	<b>155</b>
Using the UNION Operator	156
Distinct and Non-Distinct Unions	158
Intersecting Queries	159
Looking Ahead	161
<b>16 Stored Procedures and Parameters</b>	<b>163</b>
Creating Stored Procedures	164
Parameters in Stored Procedures	165
Executing Stored Procedures	167
Modifying and Deleting Stored Procedures	167
Functions Revisited	168
Looking Ahead	169



**17 Modifying Data 171**

- Modification Strategies 171
- Inserting Data 172
- Deleting Data 175
- Updating Data 176
- Correlated Subquery Updates 177
- Looking Ahead 179

**18 Maintaining Tables 181**

- Data Definition Language 181
- Table Attributes 182
- Table Columns 183
- Primary Keys and Indexes 183
- Foreign Keys 184
- Creating Tables 185
- Creating Indexes 187
- Looking Ahead 187

**19 Principles of Database Design 189**

- Goals of Normalization 190
- How to Normalize Data 191
- The Art of Database Design 195
- Alternatives to Normalization 196
- Looking Ahead 197

**20 Strategies for Displaying Data 199**

- Crosstab Layouts Revisited 199
- Excel and External Data 200
- Excel Pivot Tables 203
- Looking Ahead 207

**A Getting Started with Microsoft SQL Server 209**

- Installing SQL Server 2016 Express 209
- Installing SQL Server 2016 Management Studio Express 210
- Using SQL Server 2016 Management Studio Express 210

**B Getting Started with MySQL 211**

Installing MySQL on Windows 211

Installing MySQL on a Mac 212

Using MySQL Workbench 213

**C Getting Started with Oracle 215**

Installing Oracle Database Express Edition 215

Using Oracle Database Express Edition 216

**Index 217**

## 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](http://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](https://facebook.com/larryrockoff)

[twitter.com/larryrockoff](https://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](http://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](mailto: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](http://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](http://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](http://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.

# 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:

<u>The Answer</u>
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:

<u>Convert to Uppercase</u>	<u>Convert to Lowercase</u>
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:

<u>5 Squared</u>
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:

<u>Square Root of 25</u>
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

---

- comments, using, 13
- "
  - in Oracle, 24
  - using with column names, 16, 140
- ' , using with literal values, 20-21
- ' , ' meaning of, 22
- operator, 22
- / operator, 22
- /\* \*/ comment format, 14
- ;, using with statements, 13
- \_ wildcard, effect of, 56-57
- ` , using with column names, 16, 140
- || in Oracle, 23
- + operator, 22
- =, using with WHERE clause, 50
- # comment format, 14
- % wildcard, effect of, 55-56
- () used in Boolean logic, 63-65
- \*
  - meaning of, 11, 21-22
  - using with COUNT function, 84-85
- \*\* operator in Oracle, 22
- [], using with wildcards, 56, 58
- ^ symbol, effect of, 56
- < operator, using AND expression with, 68
- < operator, using with WHERE clause, 50
- <= operator, using with WHERE clause, 50
- <>, using with WHERE clause, 50
- > operator, using AND expression with, 68
- > operator, using with WHERE clause, 50-51
- >= operator, using with WHERE clause, 50

## A

---

- accent grave, using with column names, 16, 140
- Access, 4
- Actors table, wildcards examples, 57
- adding subtotals
  - with CUBE keyword, 106-109
  - with ROLLUP keyword, 102-106
- addition operator, 22

- aggregate functions
  - explained, 28
  - selection criteria on, 89-90
  - using, 83-84
- aliases, 23-25
- alphabetic order, sorting data in, 42, 47
- ALTER FUNCTION keyword, 168
- ALTER PROCEDURE keyword, 167-168
- ALTER TABLE statement, 186-187
- ALTER VIEW statement, 141-142
- AND expression and operator, 62, 68, 159
- angle brackets, using with WHERE clause, 50
- arguments
  - for functions, 29
  - separating, 36
- arithmetic calculations, 21-22. *See also* calculated fields
- AS keyword
  - and column aliases, 121
  - for columns, 23-25
  - and crosstab layouts, 111
  - and table aliases, 120
  - for tables, 24-25
  - using to separate arguments, 36
- ASC keyword, 43
- ascending order, sorting in, 41-43, 45
- asterisk
  - meaning of, 11, 21-22
  - using with COUNT function, 84-85
- auto-increment, 6, 183
- AVG function, 84

## B

---

- BEGIN keyword and stored procedures, 164
- BETWEEN operator, 68-69
- Boolean logic
  - explained, 61
  - NOT operator, 66-67
  - and NULL values, 70-72
  - AND operator, 62
  - OR operator, 62-63
  - BETWEEN operator, 68-69
  - IN operator, 69-70
  - parentheses, 63-65



## C

---

**C++, comparing to SQL, 2**
**calculated columns**

creating with views, 141

subqueries as, 151–152

**calculated fields. See also arithmetic calculations**

explained, 19

and literal values, 20

sorting by, 44–45

**CALL keyword, 167****caret symbol, effect of, 56****Cascade foreign key, 185****CASE expression**

effect of, 74

format of, 74–75

keywords, 74

placement of, 74

searched format, 76–77

**CASE statement**

and CUBE keyword, 109

and GROUPING function, 104–106

using with GROUP BY clause, 91

**CAST function, effect of, 36–38****character data, joining, 22****character datatypes, 7****character functions**

explained, 28

LEFT, 28–29

LOWER, 31

LTRIM, 31

RIGHT, 29–30

RTRIM, 31

sort sequence, 45

SUBSTRING, 30

UPPER, 31

**[characterlist] wildcards, effects of, 56****collation settings, customizing, 45****ColorInventory table and CROSS JOINS, 132****column aliases, 23–24, 28****column names, lack of case sensitivity, 15****columns. See also rows; tables**

adding indexes to, 183–184

designating as foreign keys, 184–185

explained, 5, 183

names with embedded spaces, 15–16

and PIVOT operator, 114

renaming, 141

sorting, 87–88

sorting by, 43–44

specifying, 14–15, 52

specifying values for, 184

**comments, using with statements, 13–14****common table expressions, 152–153****Compact Form pivot tables, 206****composite functions, 32****computer languages, procedural, 2****concatenating fields, 22****concatenation, using with calculated fields, 44****conditional logic**

CASE expression, 73–77

explained, 73

in GROUP BY clauses, 91

in HAVING clauses, 92

in ORDER BY clauses, 78

in WHERE clauses, 79

**conversion functions, 36–38****correlated subqueries, 148–149**

EXISTS operator, 150–151

updating, 177–179

**COUNT function, 84–85, 151****CREATE FUNCTION keyword, 168****CREATE INDEX statement, 187****CREATE PROCEDURE keyword, 164, 167–168, 181****CREATE TABLE statement, 185–187****CREATE VIEW keyword, 138–139, 142, 181****cross joins, 131–133. See also joining****crosstab layouts, 110–114, 199–200****CUBE keyword, 106–109****CURRENT\_DATE function, 33****Customers table**

columns in, 5

in outer join examples, 124

SELECT statement, 12

values, 124

## D

**data. See also displaying data; sorting**

creating subsets of, 141

deleting, 175–176

eliminating redundancies, 191–192

exporting to Excel, 201

formatting with views, 141

grouping, 86–88

inserting, 172–175

modification strategies, 171–172

normalizing, 191–195

obtaining from relational databases, 201

partitioning, 97–99

selecting from expressions, 30

summarizing, 17

updating, 176–177

**data sources, subqueries as, 144–147**

**database design**

- alternatives to normalization, 196–197
- art of, 195
- normalization, 190–195

**datatypes**

- of columns, 6–8
- converting, 36

**DATE\_FORMAT function, 34****DATEDIFF function, effect of, 34****DATEPART function, effect of, 33****date/time functions and datatypes, 7–8, 33–34****DBMS (database management systems), 3****DCL (Data Control Language), 2, 181****DDL (Data Definition Language), 2, 181–182****declarative language, explained, 2****delete anomalies, eliminating, 191****deleting**

- data, 175–176
- rows, 171, 175–176
- stored procedures, 167–168
- views, 141–142

**denominator value, testing, 77****DENSE\_RANK function, 93, 95****dependencies, eliminating, 192****descending order, sorting in, 43****displaying data. *See also* data**

- crosstab layouts, 199–200
- Excel and external data, 200–203
- Excel pivot tables, 203–207

**DISTINCT keyword, 85****division operator, 22****DML (Data Manipulation Language), 2, 181****does not equal operator, using with WHERE clause, 50****double dash comments, using, 13****double quote, using with column names, 16, 140****drillthrough in pivot tables, explained, 207****DROP FUNCTION keyword, 168****DROP INDEX statement, 187****DROP PROCEDURE statement, 168****DROP VIEW statement, 142****duplicates, eliminating, 81–82, 190**

---

**E**

---

**ELSE keyword, using with CASE expression, 74–75****END keyword**

- and stored procedures, 164
- using with CASE expression, 74–75

**English sentences, comparing SQL syntax to, 1–2****entity-relationship diagram, explained, 117****equals sign, using with WHERE clause, 50****Excel**

- drillthrough in pivot tables, 207
- DSN (Data Source Name) files, 202
- exporting data to, 201
- and external data, 200–203
- From Data Connection Wizard, 201
- Import Data options, 201
- Microsoft Query window, 202–203
- pivot tables, 203–207

**EXCEPT operator, format of, 160–161****EXEC keyword and stored procedures, 167****EXISTS operator**

- and subqueries, 150–151
- and updating data, 178

**exponentiation operator, 22****exporting data to Excel, 201****expressions**

- selecting data from, 30
- using italics with, 13

---

**F**

---

**false condition, evaluating, 61****fields, concatenating, 22****first normal form, 191****FirstName, sorting by, 42–43****foreign keys**

- designating columns as, 184–185
- and joining tables, 117
- and primary keys, 6

**Formats table, normalizing, 193–194****FROM clause**

- explained, 17
- and ORDER BY clause, 41–42
- specifying inner joins with, 119–120
- and subqueries, 144
- and table aliases, 120

**full joins, 129–131. *See also* joining****functions**

- arguments for, 29
- creating, 168–169
- explained, 27–28
- versus procedures, 169
- saving, 168
- scalar and aggregate, 28

---

**G**

---

**GETDATE function, effect of, 33****Grades table**

- normalizing, 193–195
- star schema, 196–197

**greater than operators**

- using AND expression with, 68
- using with WHERE clause, 50–51

**GROUP BY clause**

- and calculated columns, 151
- and CASE statement, 91
- conditional logic in, 91
- and crosstab layouts, 112
- explained, 17
- versus ORDER BY, 88
- versus partitions, 98
- ROLLUP keyword in, 102–103
- and subqueries, 149
- using, 86–87

**grouping data, 86–90****GROUPING function**

- and CUBE keyword, 109
- effect of, 104, 108

---

**H**

---

**HAVING clause**

- conditional logic in, 92
- explained, 17
- and subqueries, 149
- using, 89–90
- versus WHERE clause, 90

**header row and literal values, 20**

---

**I**

---

**IFNULL function, 38****importing data into Excel, 201–202****IN operator, 69–70, 147, 150****indexes**

- adding and modifying, 187
- creating, 187
- and primary keys, 183–184

**inner joins. See also joining**

- and ON, 120
- alternate specification, 119–120
- effect of, 118–119
- versus LEFT JOIN, 126
- and subqueries, 150–151
- and table aliases, 120
- table order, 119

**insert anomalies, eliminating, 191****inserting data, 172–175****installing**

- MySQL on Macs, 212–213
- MySQL on Windows, 211–212
- Oracle Database Express Edition, 215–216
- SQL Server 2016 Express, 209

- SQL Server 2016 Management Studio Express, 210

**INTERSECT operator, 159–161****IS NOT NULL keyword, 71****IS NULL keyword, 71–72****ISNULL function, effect of, 37–38, 46, 71–72, 86, 105****italics, using with expressions, 13**

---

**J**

---

**JOIN keyword, effect of, 17, 116, 123**

- joining. See also cross joins; full joins; inner joins; left joins; outer joins; right joins; self joins**
- character data, 22
- tables, 116–117

---

**K**

---

**keys. See foreign keys; primary keys****keywords**

- explained, 11
- lack of case sensitivity, 15
- treatment of, 15

---

**L**

---

**languages**

- versus logic, 1
- procedural versus declarative, 2

**LastName, sorting by, 42–43****LEFT function**

- arguments, 29
- effect of, 28
- format of, 29
- formula for, 29

**left joins. See also joining**

- and calculated columns, 151
- effect of, 125–127
- versus INNER JOIN keyword, 126
- versus RIGHT JOIN, 128
- and subqueries, 146
- switching with right joins, 129
- testing for NULL values, 127

**less than operators**

- using AND expression with, 68
- using with WHERE clause, 50

**LIKE operator**

- and percent wildcard, 56
- using in WHERE clause, 54

**LIMIT keyword, 52****literal space, indicating, 22**

**literal values**

- and calculated fields, 20–21
- header row, 20

**logical conditions**

- explained, 61
- NOT operator, 66–67
- and NULL values, 70–72
- AND operator, 62
- OR operator, 62–63
- BETWEEN operator, 68–69
- IN operator, 69–70
- parentheses, 63–65

**LOWER function, effect of, 31****LTRIM function, effect of, 28, 31**

---

**M**

---

**Macs, installing MySQL on, 212–213****math calculations, 21–22. See also****calculated fields****MAX function, 84****Microsoft Access, 4****Microsoft Query window, 202–203****Microsoft SQL Server**

- concatenating fields, 22
- datatypes, 7–8
- downloading free versions, 4
- versions and editions, 3

**MIN function, 84****MINUS operator, 161****Movies table**

- and FULL JOINS, 130–131
- pattern matching example, 54–56

**multiplication operator, 22****MySQL open-source database**

- ; used with statements, 13
- ` and column names, 16
- ALTER PROCEDURE command, 168
- CALL keyword, 167
- comments, 14
- CONCAT function, 23
- CREATE statements, 182
- CREATE TABLE statement, 186
- datatypes, 7–8
- DATE\_FORMAT function, 34
- DATEDIFF function, 34
- downloading free versions, 4
- IFNULL function, 38
- installing, 211–213
- LIMIT keyword, 52
- literal values, 20
- NOW function, 33
- parameters in stored procedures, 166
- platforms and editions, 3

- ROLLUP keyword, 106
- spaces and functions, 31
- stored procedures, 164–165

**MySQL Workbench, using, 213**

---

**N**

---

**No Action foreign key, 185****normalization**

- alternatives, 196–197
- of data, 191–195
- goals of, 190–191
- grades example, 195

**NOT NULL keyword, 183****NOT operator, 66–67****NOW function, 33****NTILE ranking function, 93, 96****NULL values**

- allowing, 183
- and Boolean logic, 70–72
- converting, 37–38
- and crosstab layouts, 113
- and CUBE keyword, 108
- explained, 8
- in Grade column, 85
- for Homework GradeType, 86
- in Homework row, 87
- and ROLLUP keyword, 103–104
- sort sequence, 45–47
- testing in LEFT JOIN, 127

**NULL word, suppressing printing of, 105****numeric datatypes, 7****numeric functions, 35–36****numeric values, sort sequence of, 47****NVL function, 38**

---

**O**

---

**ON clause, in self joins, 137****ON keyword and INNER JOIN, 120****OR expression and operator, 62–64, 68****Oracle databases**

- ; used with statements, 13
- " and column names, 16
- case sensitivity, 55
- column aliases, 24
- columns as sequences, 183
- concatenation, 23
- CREATE commands, 182
- CURRENT\_DATE function, 33
- datatypes, 7–8
- date display format, 126
- downloading free versions, 4
- DROP INDEX statement, 187

**Oracle databases** (*continued*)

- DROP VIEW, 142
- editions, 3
- exponentiation operator, 22
- literal values, 20
- MINUS operator, 161
- NVL function, 38
- PIVOT keyword, 114
- ROWNUM keyword, 52
- stored procedures, 164
- SUBSTR function, 31
- table aliases, 25, 121

**Oracle Database Express Edition, installing and using, 215–216****ORDER BY clause**

- conditional logic in, 78
- and crosstab layouts, 112–113
- and CUBE keyword, 109
- effect of, 41, 43–44, 47
- explained, 17
- versus GROUP BY, 88
- and ROLLUP keyword, 103–104
- switching, 88
- and TOP keyword, 53–54
- and UNION operator, 157
- and views, 139

**Orders table**

- columns in, 5
- INTERSECT operator, 160
- joining, 116–117
- in outer join examples, 124
- UNION operator, 156
- values, 125

outer joins, table order in, 129. *See also* joining

Outline Form pivot tables, 206–207

OVER keyword, using with ranking functions, 94–95

---

**P**


---

parameters in stored procedures, 165–166

parent table, explained, 184

parentheses used in Boolean logic, 63–65

partitions, 97–99

pattern matching, 54–56

PE Ratio column, 94–95

percent wildcard, effect of, 55–56

Personnel table and self joins, 136

PI function, effect of, 35

PIVOT operator, using with crosstab layouts, 111–112, 114, 200

pivot tables, using in Excel, 203–207

POWER function, effect of, 35–36

**primary keys**

- contents, 8
- and foreign keys, 6
- and indexes, 183–184
- and normalization, 190
- specifying, 12

procedural language, explained, 2

procedures versus functions, 169

Products table, NULL and Boolean logic, 70–71

**Purchases table**

- AND operator example, 62
- OR operator example, 63

---

**Q**


---

Quartile column, 96

queries. *See also* subqueries

- combining, 155
- intersecting, 159–161

quotation mark ('), using with literal values, 20–21

---

**R**


---

RANK function, 93, 95

ranking functions, 93–99

records. *See* rows

**Refunds table**

- in outer join examples, 124
- values, 125

relating tables, 6

**relational databases**

- availability, 4
- features, 4–5
- obtaining data from, 201

**Returns table**

- INTERSECT operator, 160
- UNION operator, 156

**RIGHT function, 32**

- effect of, 29, 32
- problem with spaces, 30

right joins, 128–129. *See also* joining

ROLLUP keyword, 102–108

ROUND function, effect of, 35

ROW\_NUMBER ranking function, 93–94, 98

ROWNUM keyword, 52

rows. *See also* columns; tables

- counting, 84–85
- deleting, 175–176
- deleting from tables, 171
- explained, 5
- identifying by primary keys, 6
- inserting, 172
- limiting with selection criteria, 51–52
- limiting with sort, 53–54

selecting using parentheses, 65  
 using NOT operator with, 66

**RTRIM function, 30–32**

## S

### Sales table

calculated fields, 19  
 joining, 116–117

**Salespeople table, sorting examples in, 42**

**saving stored procedures and functions, 168**

**scalar function, explained, 28**

**second normal form, 191**

**security restrictions, enforcing with views, 141**

**SELECT clause, explained, 16**

**SELECT mainquery, order of fields in, 113**

**SELECT queries and set logic, 155**

### SELECT statements

and ALTER VIEW, 141–142  
 with ASC keyword, 43  
 and Boolean logic, 63  
 and CASE expression, 75  
 format of, 73–74  
 and INNER JOIN, 118  
 introducing, 11  
 for joining tables, 117  
 with LEFT function, 28  
 and LEFT JOIN, 126  
 with LOWER function, 31  
 with ORDER BY clause, 41  
 parameters in stored procedures, 165–166  
 and parentheses, 65  
 with RIGHT function, 30, 32  
 with RTRIM function, 32  
 and self joins, 137  
 separating with UNION operator, 157  
 specifying clauses in, 17  
 specifying columns, 14–15  
 and stored procedures, 165  
 and subqueries, 144, 146  
 with SUBSTRING function, 30  
 FROM TableForSort, 46  
 with UPPER function, 31  
 using with Customers table, 12  
 as views, 139  
 views as, 137  
 with WHERE clause, 49

### selection criteria

on aggregate functions, 89–90  
 applying, 49–50  
 limiting rows, 51–52  
 limiting rows with sort, 53–54  
 pattern matching, 54–56

subqueries in, 147–148  
 TOP keyword, 52

**selection logic, indicating, 17**

**self joins, effect of, 135–137. See also joining; views**

**semicolon (;), using with statements, 13**

**SET A and B, selecting data in, 156**

**SET keyword, using to update data, 176–178**

### set logic

distinct unions, 158–159  
 explained, 155  
 intersecting queries, 159–161  
 non-distinct unions, 158–159  
 UNION operator, 156–158

**Set Null foreign key, 185**

**Shakespeare example, 53–54**

**soft delete technique, 171**

**sorting. See also data**

in ascending order, 41–43  
 by calculated field, 44–45  
 by columns, 43–44  
 columns, 87–88  
 in descending order, 43  
 sequences, 45–47

### spaces

including in columns, 15–16  
 problem in RIGHT function, 30

### SQL (Structured Query Language)

components, 2  
 explained, 2  
 pronunciation, 2  
 significance, 8–9  
 versus Visual Basic and C++, 2

**SQL database, explained, 3**

### SQL Server

concatenating fields, 22  
 datatypes, 7–8  
 downloading free versions, 4  
 versions and editions, 3

### SQL Server 2016

Express installation, 209  
 Management Studio Express installation, 210

**SQL statements, 13–14**

### SQL syntax

relating to English sentences, 1–3  
 for writing statements, 12–13

**square brackets, using with wildcards, 56, 58**

**square root, determining, 36**

**star schema, 196–197**

**statements, 13–14**

### stored procedures

creating, 164–165, 167  
 deleting, 167–168

**stored procedures** (*continued*)

- executing, 167, 169
- explained, 163–164
- parameters, 169
- parameters in, 165–166
- saving, 168

**string datatypes**, 7**string functions**. See **character functions****Students table, normalizing**, 193**subqueries**. See *also* **queries**

- analyzing, 146
- as calculated columns, 151–152
- correlated and uncorrelated, 148–151
- as data sources, 144–147
- EXISTS operator, 150–151
- explained, 70, 143
- in selection criteria, 147–148
- specifying, 144
- types of, 143–144

**SUBSTR function**, 31**SUBSTRING function, effect of**, 30**subtotals**

- adding with CUBE keyword, 106–109
- adding with ROLLUP, 102–106

**subtraction operator**, 22**SUM function**

- and CUBE keyword, 109
- effect of, 28, 84

**summarizing data**, 17**summing values in pivot tables**, 206**syntax**

- relating to English sentences, 1–3
- for writing statements, 12–13

---

## T

---

**table aliases**, 24–25**table expressions**, 152–153**table names, lack of case sensitivity**, 15**tables**. See *also* **columns**; **rows**

- attributes, 182
- creating, 185–187
- DDL (Data Definition Language), 181–182
- displaying, 5
- joining, 116–117, 125–127
- order in INNER JOINS, 119
- relating, 6

**Tabular Form pivot tables**, 206**Teachers table, normalizing**, 193–194**Tests table, normalizing**, 193–194**THEN keyword, using with CASE expression**, 74**third normal form**, 191**time**. See **date/time functions and datatypes****TOP keyword, effect of**, 52–54**Transactions table, subqueries**, 145, 147–149**true condition, evaluating**, 61**TRUNCATE TABLE statement**, 176

---

## U

---

**uncorrelated subqueries**, 148–149**underscore wildcard, effect of**, 56–57**UNION and UNION ALL**, 156–159**uniqueness, ensuring with primary keys**, 6**update anomalies, eliminating**, 191**updating data**, 176–179**UPPER function, effect of**, 31**Users table, subqueries**, 144–145, 147–149

---

## V

---

**values, summation in pivot tables**, 206**VALUES keyword**, 173–174**VARCHAR**, effect of, 46**Vendors table, updating**, 178**views**. See *also* **self joins**

- benefits, 140–141
- creating, 137–139
- modifying and deleting, 141–142
- referencing, 139–140
- as SELECT statements, 137, 139
- storing, 138

**Visual Basic, comparing to SQL**, 2

---

## W

---

**WHEN keyword, using with CASE expression**, 74–75**WHERE clause**

- AND operator, 62
- applying selection criteria, 49–50
- conditional logic in, 79
- and correlated subqueries, 179
- effect of, 17, 50
- example, 58
- versus HAVING clause, 90
- interpreting in Boolean logic, 64
- LEFT JOIN with NULL values, 127
- LIKE operator, 54
- operators, 50–51
- parameters in stored procedures, 166
- placement of, 49
- sample output, 50
- specifying inner joins with, 119–120
- subqueries, 146
- TOP keyword, 54

**wildcards, arranging**, 56–58**Windows, installing MySQL on**, 211–212**WITH keyword, effect of**, 152–153