Ben Forta

FOURTH EDITION
FULL COLOR

Sams Teach Yourself

SQL

in 10 Minutes

FREE SAMPLE CHAPTER

SHARE WITH OTHERS

Ben Forta

Sams Teach Yourself SQL in 10 Minutes

Fourth Edition

SAMS 800 East 96th Street, Indianapolis, Indiana 46240

Sams Teach Yourself SQL in 10 Minutes, Fourth Edition

ISBN-13: 9780672336072

ISBN-10: 0672336073

Library of Congress cataloging-in-Publication Data is on file.

Printed in the United States of America

Seventh Printing: December 2014

Acquisitions Editor
Mark Taber

Managing Editor
Kristy Hart

Project Editor
Andy Beaster

Copy Editor
Apostrophe Editing Services

Indexer
Tim Wright

Proofreader
Kathy Ruiz

Technical Editors
Chris McGee
Greg Wilson

Publishing Coordinator
Vanessa Evans

Designer
Anne Jones

Page Layout
TnT Design, Inc.

Table of Contents

About the Author

Ben Forta is Adobe Systems' Director of Developer Relations and has more than 20 years of experience in the computer industry in product development, support, training, and product marketing. He is the author of the best-selling *Sams Teach Yourself SQL in 10 Minutes*, spinoff titles on MySQL and SQL Server T-SQL, *ColdFusion Web Application Construction Kit* and *Advanced ColdFusion Application Development* (both published by Adobe Press), *Sams Teach Yourself Regular Expressions in 10 Minutes*, as well as books on Flash, Java, Windows, and other subjects. He has extensive experience in database design and development, has implemented databases for several highly successful commercial software programs and websites, and is a frequent lecturer and columnist on Internet and database technologies. Ben lives in Oak Park, Michigan, with his wife Marcy and their seven children. Ben welcomes your e-mail at ben@forta.com and invites you to visit his website at `http://forta.com/`.

Acknowledgments

Thanks to the team at Sams for all these years of support, dedication, and encouragement. A special thank-you to Mark Taber for encouraging this long awaited update and for suggesting and facilitating the code coloring, which significantly enhances the readability and value of this new edition.

Thanks to my colleague Greg Wilson for his thorough technical review.

Thanks to the many hundreds of you who provided feedback on the first three editions of this book. Fortunately, most of it was positive, and all of it was appreciated. The enhancements and changes in this edition are a direct response to your feedback, which I continue to welcome.

Thanks to the dozens of colleges and universities who have made this book part of their IT and computer science curriculums. Being included and trusted by professors and teachers this way is immensely rewarding and equally humbling.

And finally, thanks to the more than one-quarter million of you who bought the previous editions of this book, making it not just my best-selling title, but also one of the best-selling books on the subject. Your continued support is the highest compliment an author can ever be paid.

—Ben Forta

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.

You can e-mail 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 stronger.

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 reply to every message.

When you write, please be sure to include this book's title and author as well as your name and contact information.

E-mail: feedback@samspublishing.com

Mail: Reader Feedback
Sams Publishing/Pearson Education
800 East 96th Street
Indianapolis, IN 46240 USA

Reader Services

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

This page intentionally left blank

Introduction

SQL is the most widely used database language. Whether you are an application developer, database administrator, web application designer, mobile app developer, or Microsoft Office user, a good working knowledge of SQL is an important part of interacting with databases.

This book was born out of necessity. I had been teaching Web application development for several years, and students were constantly asking for SQL book recommendations. There are lots of SQL books out there. Some are actually very good. But they all have one thing in common: for most users they teach just too much information. Instead of teaching SQL itself, most books teach everything from database design and normalization to relational database theory and administrative concerns. And while those are all important topics, they are not of interest to most of us who just need to learn SQL.

And so, not finding a single book that I felt comfortable recommending, I turned that classroom experience into the book you are holding. *Sams Teach Yourself SQL in 10 Minutes* will teach you SQL you need to know, starting with simple data retrieval and working on to more complex topics including the use of joins, subqueries, stored procedures, cursors, triggers, and table constraints. You'll learn methodically, systematically, and simply—in lessons that will each take 10 minutes or less to complete.

Now in its fourth edition, this book has taught SQL to over a quarter million English speaking users, and has been translated into over a dozen other languages too so as to help users the globe over. And now it is your turn. So turn to Lesson 1, and get to work. You'll be writing world class SQL in no time at all.

Who Is the Teach Yourself SQL Book For?

This book is for you if

- You are new to SQL.
- You want to quickly learn how to get the most out of SQL.
- You want to learn how to use SQL in your own application development.
- You want to be productive quickly and easily in SQL without having to call someone for help.

DBMSs Covered in This Book

For the most part, the SQL taught in this book will apply to any Database Management System (DBMS). However, as all SQL implementations are not created equal, the following DBMSs are explicitly covered (and specific instructions or notes are included where needed):

- Apache Open Office Base
- IBM DB2
- Microsoft Access
- Microsoft SQL Server (including Microsoft SQL Server Express)
- MariaDB
- MySQL
- Oracle (including Oracle Express)
- PostgreSQL
- SQLite

Example databases (or SQL scripts to create the example databases) are available for all of these DBMSs on the book webpage at `http://forta.com/books/0672336073/`.

Conventions Used in This Book

This book uses different typefaces to differentiate between code and regular English, and also to help you identify important concepts.

Text that you type and text that should appear on your screen is presented in `monospace` type.

```
It will look like this to mimic the way text looks on your
screen.
```

Placeholders for variables and expressions appear in *`monospace italic`* font. You should replace the placeholder with the specific value it represents.

This arrow (➥) at the beginning of a line of code means that a single line of code is too long to fit on the printed page. Continue typing all the characters after the ➥ as though they were part of the preceding line.

NOTE:

A Note presents interesting pieces of information related to the surrounding discussion.

TIP:

A Tip offers advice or teaches an easier way to do something.

CAUTION:

A Caution advises you about potential problems and helps you steer clear of disaster.

PLAIN ENGLISH

New Term icons provide clear definitions of new, essential terms.

Input ▼

The Input icon identifies code that you can type in. It usually appears next to a listing.

Output ▼

The Output icon highlights the output produced by running a program. It usually appears after a listing.

Analysis ▼

The Analysis icon alerts you to the author's line-by-line analysis of a program.

LESSON 2

Retrieving Data

In this lesson, you'll learn how to use the `SELECT` *statement to retrieve one or more columns of data from a table.*

The `SELECT` Statement

As explained in Lesson 1, "Understanding SQL," SQL statements are made up of plain English terms. These terms are called keywords, and every SQL statement is made up of one or more keywords. The SQL statement that you'll probably use most frequently is the `SELECT` statement. Its purpose is to retrieve information from one or more tables.

> **Keyword**
> A reserved word that is part of the SQL language. Never name a table or column using a keyword. Appendix E, "SQL Reserved Words," lists some of the more common reserved words.

To use `SELECT` to retrieve table data you must, at a minimum, specify two pieces of information—what you want to select, and from where you want to select it.

NOTE: **Following Along with the Examples**

The sample SQL statements (and sample output) throughout the lessons in this book use a set of data files that are described in Appendix A, "Sample Table Scripts." If you'd like to follow along and try the examples yourself (I strongly recommend that you do so), refer to Appendix A which contains instructions on how to download or create these data files.

It is important to understand that SQL is a language, not an application. The way that you specify SQL statements and display statement output varies from one application to the next. To assist you in adapting the examples to your own environment, Appendix B, "Working in Popular Applications," explains how to issue the statements taught throughout this book using many popular applications and development environments. And if you need an application with which to follow along, Appendix B has recommendations for you too.

Retrieving Individual Columns

We'll start with a simple SQL `SELECT` statement, as follows:

Input ▼

```
SELECT prod_name
FROM Products;
```

Analysis ▼

The previous statement uses the `SELECT` statement to retrieve a single column called `prod_name` from the `Products` table. The desired column name is specified right after the `SELECT` keyword, and the `FROM` keyword specifies the name of the table from which to retrieve the data. The output from this statement is shown in the following:

Output ▼

```
prod_name
--------------------
Fish bean bag toy
Bird bean bag toy
Rabbit bean bag toy
8 inch teddy bear
12 inch teddy bear
18 inch teddy bear
Raggedy Ann
King doll
Queen doll
```

NOTE: **Unsorted Data**

If you tried this query yourself you might have discovered that the data was displayed in a different order than shown here. If this is the case, don't worry—it is working exactly as it is supposed to. If query results are not explicitly sorted (we'll get to that in the next lesson) then data will be returned in no order of any significance. It may be the order in which the data was added to the table, but it may not. As long as your query returned the same number of rows then it is working.

A simple `SELECT` statement similar to the one used above returns all the rows in a table. Data is not filtered (so as to retrieve a subset of the results), nor is it sorted. We'll discuss these topics in the next few lessons.

TIP: **Terminating Statements**

Multiple SQL statements must be separated by semicolons (the `;` character). Most DBMSs do not require that a semicolon be specified after single statements. But if your particular DBMS complains, you might have to add it there. Of course, you can always add a semicolon if you wish. It'll do no harm, even if it is, in fact, not needed.

TIP: **SQL Statement and Case**

It is important to note that SQL statements are case-insensitive, so `SELECT` is the same as `select`, which is the same as `Select`. Many SQL developers find that using uppercase for all SQL keywords and lowercase for column and table names makes code easier to read and debug. However, be aware that while the SQL language is case-insensitive, the names of tables, columns, and values may not be (that depends on your DBMS and how it is configured).

TIP: **Use of White Space**

All extra white space within a SQL statement is ignored when that statement is processed. SQL statements can be specified on one long line or broken up over many lines. So, the following three statements are functionality identical:

```
SELECT prod_name
FROM Products;

SELECT prod_name FROM Products;

SELECT
prod_name
FROM
Products;
```

Most SQL developers find that breaking up statements over multiple lines makes them easier to read and debug.

Retrieving Multiple Columns

To retrieve multiple columns from a table, the same `SELECT` statement is used. The only difference is that multiple column names must be specified after the `SELECT` keyword, and each column must be separated by a comma.

TIP: **Take Care with Commas**

When selecting multiple columns be sure to specify a comma between each column name, but not after the last column name. Doing so will generate an error.

The following SELECT statement retrieves three columns from the Products table:

Input ▼

```
SELECT prod_id, prod_name, prod_price
FROM Products;
```

Analysis ▼

Just as in the prior example, this statement uses the SELECT statement to retrieve data from the Products table. In this example, three column names are specified, each separated by a comma. The output from this statement is shown below:

Output ▼

```
prod_id      prod_name              prod_price
---------    --------------------   ----------
BNBG01       Fish bean bag toy      3.4900
BNBG02       Bird bean bag toy      3.4900
BNBG03       Rabbit bean bag toy    3.4900
BR01         8 inch teddy bear      5.9900
BR02         12 inch teddy bear     8.9900
BR03         18 inch teddy bear     11.9900
RGAN01       Raggedy Ann            4.9900
RYL01        King doll              9.4900
RYL02        Queen dool             9.4900
```

NOTE: **Presentation of Data**

As you will notice in the above output, SQL statements typically return raw, unformatted data. Data formatting is a presentation issue, not a retrieval issue. Therefore, presentation (for example, displaying the above price values as currency amounts with the correct number of decimal places) is typically specified in the application that displays the data. Actual retrieved data (without application-provided formatting) is rarely used.

Retrieving All Columns

In addition to being able to specify desired columns (one or more, as seen above), SELECT statements can also request all columns without having to list them individually. This is done using the asterisk (*) wildcard character in lieu of actual column names, as follows:

Input ▼

```
SELECT *
FROM Products;
```

Analysis ▼

When a wildcard (*) is specified, all the columns in the table are returned. The column order will typically, but not always, be the physical order in which the columns appear in the table definition. However, SQL data is seldom displayed as is. (Usually, it is returned to an application that formats or presents the data as needed). As such, this should not pose a problem.

> CAUTION: **Using Wildcards**
>
> As a rule, you are better off not using the * wildcard unless you really do need every column in the table. Even though use of wildcards may save you the time and effort needed to list the desired columns explicitly, retrieving unnecessary columns usually slows down the performance of your retrieval and your application.

> TIP: **Retrieving Unknown Columns**
>
> There is one big advantage to using wildcards. As you do not explicitly specify column names (because the asterisk retrieves every column), it is possible to retrieve columns whose names are unknown.

Retrieving Distinct Rows

As you have seen, SELECT returns all matched rows. But what if you do not want every occurrence of every value? For example, suppose you want the vendor ID of all vendors with products in your products table:

Input ▼

```
SELECT vend_id
FROM Products;
```

Output ▼

```
vend_id
----------
BRS01
BRS01
BRS01
DLL01
DLL01
DLL01
DLL01
FNG01
FNG01
```

The SELECT statement returned 9 rows (even though there are only four vendors in that list) because there are 9 products listed in the products table. So how could you retrieve a list of distinct values?

The solution is to use the DISTINCT keyword which, as its name implies, instructs the database to only return distinct values.

Input ▼

```
SELECT DISTINCT vend_id
FROM Products;
```

Analysis ▼

`SELECT DISTINCT vend_id` tells the DBMS to only return distinct (unique) `vend_id` rows, and so only three rows are returned, as seen in the following output. If used, the `DISTINCT` keyword must be placed directly in front of the column names.

Output ▼

```
vend_id
----------
BRS01
DLL01
FNG01
```

> CAUTION: **Can't Be Partially** `DISTINCT`
>
> The `DISTINCT` keyword applies to all columns, not just the one it precedes. If you were to specify `SELECT DISTINCT vend_id, prod_price`, all rows would be retrieved because *both* of the specified columns were distinct.

Limiting Results

`SELECT` statements return all matched rows, possibly every row in the specified table. What if you want to return just the first row or a set number of rows? This is doable, but unfortunately, this is one of those situations where all SQL implementations are not created equal.

In Microsoft SQL Server and Microsoft Access you can use the `TOP` keyword to limit the top number of entries, as seen here:

Input ▼

```
SELECT TOP 5 prod_name
FROM Products;
```

Output ▼

```
prod_name
----------------
8 inch teddy bear
12 inch teddy bear
18 inch teddy bear
Fish bean bag toy
Bird bean bag toy
```

Analysis ▼

The previous statement uses the `SELECT TOP 5` statement to retrieve just the first five rows.

If you are using DB2, well, then you get to use SQL unique to that DBMS, like this:

Input ▼

```
SELECT prod_name
FROM Products
FETCH FIRST 5 ROWS ONLY;
```

Analysis ▼

`FETCH FIRST 5 ROWS ONLY` does exactly what it suggests.

If you are using Oracle you need to count rows based on `ROWNUM` (a row number counter) like this:

Input ▼

```
SELECT prod_name
FROM Products
WHERE ROWNUM <=5;
```

If you are using MySQL, MariaDB, PostgreSQL, or SQLite, you can use the `LIMIT` clause, as follows:

Input ▼

```
SELECT prod_name
FROM Products
LIMIT 5;
```

Analysis ▼

The previous statement uses the `SELECT` statement to retrieve a single column. `LIMIT 5` instructs the supported DBMSs to return no more than five rows. The output from this statement is shown in the following code.

To get the next five rows, specify both where to start and the number of rows to retrieve, like this:

Input ▼

```
SELECT prod_name
FROM Products
LIMIT 5 OFFSET 5;
```

Analysis ▼

`LIMIT 5 OFFSET 5` instructs supported DBMSs to return five rows starting from row `5`. The first number is the number of rows to retrieve, and the second is where to start. The output from this statement is shown in the following code:

Output ▼

```
prod_name
------------------
Rabbit bean bag toy
Raggedy Ann
King doll
Queen doll
```

So, `LIMIT` specifies the number of rows to return. `LIMIT` with an `OFFSET` specifies where to start from. In our example there are only nine products in the `Products` table, so `LIMIT 5 OFFSET 5` returned just four rows (as there was no fifth).

CAUTION: **Row 0**

The first row retrieved is row `0`, not row `1`. As such, `LIMIT 1 OFFSET 1` will retrieve the second row, not the first one.

TIP: **MySQL, MariaDB, and SQLite Shortcut**

MySQL, MariaDB, and SQLite support a shorthand version of `LIMIT 4 OFFSET 3`, enabling you to combine them as `LIMIT 3,4`. Using this syntax, the value before the `,` is the `LIMIT` and the value after the `,` is the `OFFSET`.

NOTE: **Not ALL SQL Is Created Equal**

I included this section on limiting results for one reason only, to demonstrate that while SQL is usually quite consistent across implementations, you can't rely on it always being so. While very basic statements tend to be very portable, more complex ones tend to be less so. Keep that in mind as you search for SQL solutions to specific problems.

Using Comments

As you have seen, SQL statements are instructions that are processed by your DBMS. But what if you wanted to include text that you'd not want processed and executed? Why would you ever want to do this? Here are a few reasons:

- The SQL statements we've been using here are all very short and very simple. But, as your SQL statement grow (in length and complexity), you'll want to include descriptive comments (for your own future reference or for whoever has to work on the project next). These comments need to be embedded in the SQL scripts, but they are obviously not intended for actual DBMS processing. (For an example of this, see the `create.sql` and `populate.sql` files used in Appendix B).

- The same is true for headers at the top of SQL file, perhaps containing the programmer contact information and a description and notes. (This use case is also seen in the Appendix B `.sql` files.).
- Another important use for comments is to temporarily stop SQL code from being executed. If you were working with a long SQL statement, and wanted to test just part of it, you could *comment out* some of the code so that DBMS saw it as comments and ignored it.

Most DBMSs supports several forms of comment syntax. We'll Start with inline comments:

Input ▼

```
SELECT prod_name    -- this is a comment
FROM Products;
```

Analysis ▼

Comments may be embedded inline using `--` (two hyphens). Anything after the `--` is considered comment text, making this a good option for describing columns in a `CREATE TABLE` statement, for example.

Here is another form of inline comment (although less commonly supported):

Input ▼

```
# This is a comment
SELECT prod_name
FROM Products;
```

Analysis ▼

A `#` at the start of a line makes the entire line a comment. You can see this format comment used in the accompanying `create.sql` and `populate.sql` scripts.

You can also create multi line comments, and comments that stop and start anywhere within the script:

Input ▼

```
/* SELECT prod_name, vend_id
FROM Products; */
SELECT prod_name
FROM Products;
```

Analysis ▼

`/*` starts a comments, and `*/` ends it. Anything between `/*` and `*/` is comment text. This type of comment is often used to *comment out* code, as seen in this example. Here, two `SELECT` statements are defined, but the first won't execute because it has been commented out.

Summary

In this lesson, you learned how to use the SQL `SELECT` statement to retrieve a single table column, multiple table columns, and all table columns. You also learned how to return distinct values and how to comment your code. And unfortunately, you were also introduced to the fact that more complex SQL tends to be less portable SQL. Next you'll learn how to sort the retrieved data.

This page intentionally left blank

This page intentionally left blank

Index

D

E

F

G

H-I

J

K

L

M

N

S

T

U

V

W-X-Y-Z