Ben Forta

Sams Teach Yourself

SQL

in 10 Minutes

SAMS

Sams Teach Yourself SQL in 10 Minutes

Fourth Edition

Ben Forta

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

First Printing: October 2012

Trademarks

Warning and Disclaimer

Bulk Sales

Pearson offers excellent discounts on this book when ordered in quantity for bulk purchases or specia sales. For more information, please contact

> **U.S. Corporate and Government Sales**
> **1-800-382-3419**
> **corpsales@pearsontechgroup.com**

For sales outside of the U.S., please contact

> **International Sales**
> **international@pearsoned.com**

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 marketin 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 Expression 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 subjec 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 opinio 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 a 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 tha 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.

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 SQ 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 tim 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 replac 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 1. Understanding SQL

In this lesson, you'll learn exactly what SQL is and what it will do for you.

Database Basics

The fact that you are reading a book on SQL indicates that you, somehow, need to interact with databases. SQL is a language used to do just this, so before looking at SQL itself, it is important that you understand some basic concepts about databases and database technologies.

Whether you are aware of it or not, you use databases all the time. Each time you select a name from your email address book, you are using a database. If you conduct a search on an Internet search site, you are using a database. When you log into your network at work, you are validating your name and password against a database. Even when you use your ATM card at a cash machine, you are using databases for PIN number verification and balance checking.

But even though we all use databases all the time, there remains much confusion over what exactly a database is. This is especially true because different people use the same database terms to mean different things. Therefore, a good place to start our study is with a list and explanation of the most important database terms.

> **Tip: Reviewing Basic Concepts**
>
> What follows is a very brief overview of some basic database concepts. It is intended to either jolt your memory if you already have some database experience, or to provide you with the absolute basics, if you are new to databases. Understanding databases is an important part of mastering SQL, and you might want to find a good book on database fundamentals to brush up on the subject if needed.

Databases

The term *database* is used in many different ways, but for our purposes (and indeed, from SQL's perspective) a database is a collection of data stored in some organized fashion. The simplest way to think of it is to imagine a database as a filing cabinet. The filing cabinet is simply a physical location to store data, regardless of what that data is or how it is organized.

> **Database**
>
> A container (usually a file or set of files) to store organized data.

> **Caution: Misuse Causes Confusion**
>
> People often use the term *database* to refer to the database software they are running. This is incorrect, and it is a source of much confusion. Database software is actually called the *Database Management System* (or DBMS). The database is the container created and manipulated via the DBMS, and exactly what the database is and what form it takes varies from one database to the next.

Tables

When you store information in your filing cabinet, you don't just toss it in a drawer. Rather, you create files within the filing cabinet, and then you file related data in specific files.

In the database world, that file is called a table. A *table* is a structured file that can store data of a specific type. A table might contain a list of customers, a product catalog, or any other list of information.

> **Table**
> A structured list of data of a specific type.

The key here is that the data stored in the table is one type of data or one list. You would never store list of customers and a list of orders in the same database table. Doing so would make subsequent retrieval and access difficult. Rather, you'd create two tables, one for each list.

Every table in a database has a name that identifies it. That name is always unique—meaning no othe table in that database can have the same name.

> **Note: Table Names**
> What makes a table name unique is actually a combination of several things including the database name and table name. Some databases also use the name of the database owner as part of the unique name. This means that while you cannot use the same table name twice in the same database, you definitely can reuse table names in different databases.

Tables have characteristics and properties that define how data is stored in them. These include information about what data may be stored, how it is broken up, how individual pieces of informatio are named, and much more. This set of information that describes a table is known as a *schema*, and schemas are used to describe specific tables within a database, as well as entire databases (and the relationship between tables in them, if any).

> **Schema**
> Information about database and table layout and properties.

Columns and Datatypes

Tables are made up of columns. A column contains a particular piece of information within a table.

> **Column**
> A single field in a table. All tables are made up of one or more columns.

The best way to understand this is to envision database tables as grids, somewhat like spreadsheets. Each column in the grid contains a particular piece of information. In a customer table, for example, one column contains the customer number, another contains the customer name, and the address, city state, and ZIP code are all stored in their own columns.

> **Tip: Breaking Up Data**
>
> It is extremely important to break data into multiple columns correctly. For example, city, state, and ZIP code should always be separate columns. By breaking these out, it becomes possible to sort or filter data by specific columns (for example, to find all customers in a particular state or in a particular city). If city and state are combined into one column, it would be extremely difficult to sort or filter by state.
>
> When you break up data, the level of granularity is up to you and your specific requirements. For example, addresses are typically stored with the house number and street name together. This is fine, unless you might one day need to sort data by street name, in which case splitting house number and street name would be preferable.

Each column in a database has an associated datatype. A datatype defines what type of data the column can contain. For example, if the column were to contain a number (perhaps the number of items in an order), the datatype would be a numeric datatype. If the column were to contain dates, tex notes, currency amounts, and so on, the appropriate datatype would be used to specify this.

> **Datatype**
>
> A type of allowed data. Every table column has an associated datatype that restricts (or allows) specific data in that column.

Datatypes restrict the type of data that can be stored in a column (for example, preventing the entry c alphabetical characters into a numeric field). Datatypes also help sort data correctly and play an important role in optimizing disk usage. As such, special attention must be given to picking the right datatype when tables are created.

> **Caution: Datatype Compatibility**
>
> Datatypes and their names are one of the primary sources of SQL incompatibility. While most basic datatypes are supported consistently, many more advanced datatypes are not. And worse, occasionally you'll find that the same datatype is referred to by different names in different DBMSs. There is not much you can do about this, but it is important to keep in mind when you create table schemas.

Rows

Data in a table is stored in rows; each record saved is stored in its own row. Again, envisioning a tabl as a spreadsheet style grid, the vertical columns in the grid are the table columns, and the horizontal rows are the table rows.

For example, a customers table might store one customer per row. The number of rows in the table is the number of records in it.

> **Row**
>
> A record in a table.

> **Note: Records or Rows?**
>
> You may hear users refer to database *records* when referring to *rows*. For the most part the two terms are used interchangeably, but *row* is technically the correct term.

Primary Keys

Every row in a table should have some column (or set of columns) that uniquely identifies it. A table containing customers might use a customer number column for this purpose, whereas a table containing orders might use the order ID. An employee list table might use an employee ID or the employee Social Security number column.

> **Primary key**
>
> A column (or set of columns) whose values uniquely identify every row in a table.

This column (or set of columns) that uniquely identifies each row in a table is called a primary key. The primary key is used to refer to a specific row. Without a primary key, updating or deleting specific rows in a table becomes extremely difficult as there is no guaranteed safe way to refer to jus the rows to be affected.

> **Tip: Always Define Primary Keys**
>
> Although primary keys are not actually required, most database designers ensure that every table they create has a primary key so that future data manipulation is possible and manageable.

Any column in a table can be established as the primary key, as long as it meets the following conditions:

- No two rows can have the same primary key value.
- Every row must have a primary key value. (Primary key columns may not allow NULL values.)
- Values in primary key columns should never be modified or updated.
- Primary key values should never be reused. (If a row is deleted from the table, its primary key may not be assigned to any new rows in the future.)

Primary keys are usually defined on a single column within a table. But this is not required, and multiple columns may be used together as a primary key. When multiple columns are used, the rules listed above must apply to all columns, and the values of all columns together must be unique (individual columns need not have unique values).

There is another very important type of key called a foreign key, but I'll get to that later on in Lesson 12, "Joining Tables."

What Is SQL?

SQL (pronounced as the letters S-Q-L or as *sequel*) is an abbreviation for Structured Query Language SQL is a language designed specifically for communicating with databases.

Unlike other languages (spoken languages like English, or programming languages like Java, C, or PHP), SQL is made up of very few words. This is deliberate. SQL is designed to do one thing and do well—provide you with a simple and efficient way to read and write data from a database.

What are the advantages of SQL?

- SQL is not a proprietary language used by specific database vendors. Almost every major DBMS supports SQL, so learning this one language will enable you to interact with just about every database you'll run into.
- SQL is easy to learn. The statements are all made up of descriptive English words, and there aren't that many of them.
- Despite its apparent simplicity, SQL is actually a very powerful language, and by cleverly usin its language elements you can perform very complex and sophisticated database operations.

And with that, let's learn SQL.

> **Note: SQL Extensions**
>
> Many DBMS vendors have extended their support for SQL by adding statements or instructions to the language. The purpose of these extensions is to provide additional functionality or simplified ways to perform specific operations. And while often extremely useful, these extensions tend to be very DBMS specific, and they are rarely supported by more than a single vendor.
>
> Standard SQL is governed by the ANSI standards committee, and is thus called ANSI SQL. All major DBMSs, even those with their own extensions, support ANSI SQL. Individual implementations have their own names (PL-SQL, Transact-SQL, and so forth).
>
> For the most part, the SQL taught in this book is ANSI SQL. On the odd occasion where DBMS specific SQL is used it is so noted.

Try It Yourself

Like any language, the best way to learn SQL is to try it for yourself. To do this you'll need a databas and an application with which to test your SQL statements.

All of the lessons in this book use real SQL statements and real database tables. Appendix A, "The Example Tables," explains what the example tables are, and provides details on how to obtain (or create) them so that you may follow along with the instructions in each lesson. Appendix B, "Workin in Popular Applications," describes the steps needed to execute your SQL in a variety of applications Before proceeding to the next lesson, I'd strongly suggest that you turn to these two appendixes so th you'll be ready to follow along.

Summary

In this first lesson, you learned what SQL is and why it is useful. Because SQL is used to interact wit databases, you also reviewed some basic database terminology.

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—wh 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 nex 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:

Click here to view code image

```
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

Click here to view code image

```
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

Click here to view code image

```
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 o 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.