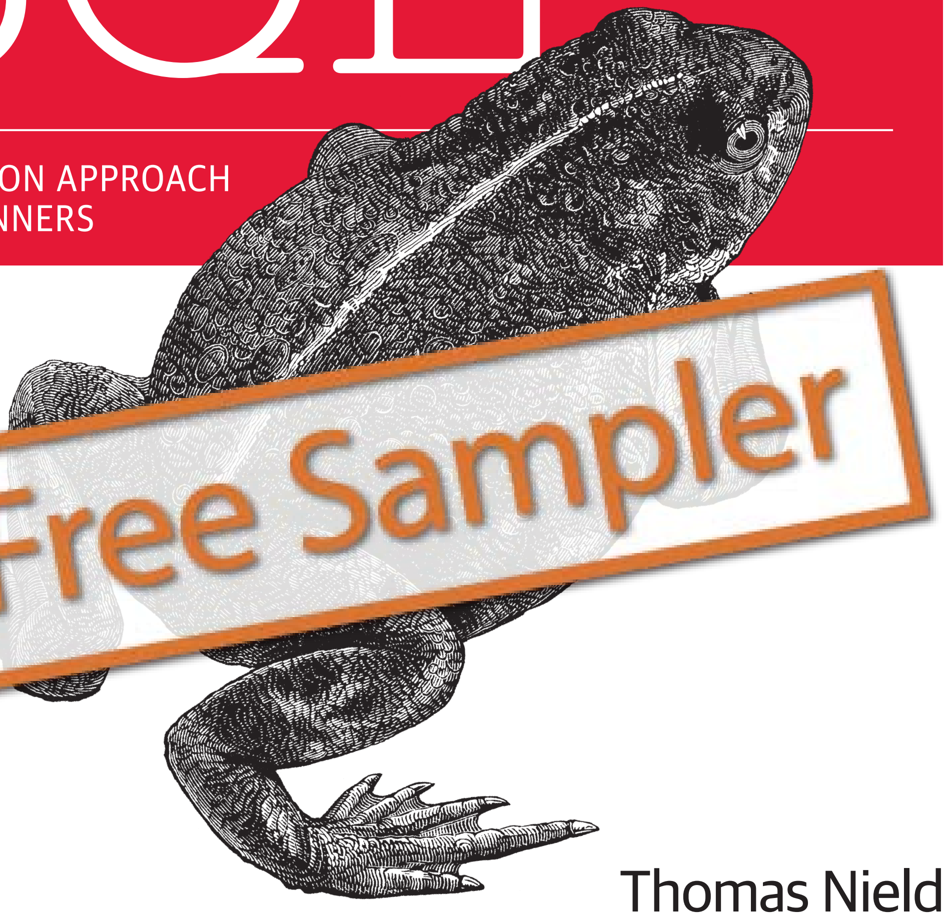
O'REILLY®

Getting Started with SQL

A HANDS-ON APPROACH FOR BEGINNERS



Thomas Nield

Getting Started with SQL
by Thomas Nield

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://safaribooksonline.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Editor: Shannon Cutt
Production Editor: Shiny Kalapurakkel
Copyeditor: Jasmine Kwityn
Proofreader: Rachel Head
Indexer: Ellen Troutman-Zaig
Interior Designer: David Futato
Cover Designer: Randy Comer
Illustrator: Rebecca Demarest

February 2016: First Edition

Revision History for the First Edition
2016-02-08: First Release

See *http://oreilly.com/catalog/errata.csp?isbn=9781491938614* for release details.

978-1-491-93861-4

[LSI]

Table of Contents

CHAPTER 1

Why Learn SQL?

What Is SQL and Why Is It Marketable?

It is an obvious statement that the business landscape is shifting rapidly. A lot of this is enabled by technology and the explosion of business data. Companies are investing vast amounts of capital to gather and warehouse data. But what many business leaders and managers currently struggle with is how to make sense of this data and use it. This is where *SQL*, which stands for *Structured Query Language*, comes in. It provides a means to access and manipulate this data in meaningful ways and provide business insights not possible before.

Businesses are gathering data at exponential rates, and there is an equally growing need for people who know how to analyze and manage it. Stack Overflow, the most active programming community in the world, performed a comprehensive survey on its members in 2015. Apple coding was the most in-demand technology and had an average salary nearing six figures. But SQL came in in fifth place, with a salary that was not far behind. In recent years, data has suddenly become ubiquitous—yet few people know how to access it meaningfully, which has put SQL talent in high demand.

Who Is SQL For?

One misperception about SQL is that it is an IT skill and therefore only applicable to technology (not business) professionals. In the world as it exists today, this is hardly the truth. Businesspeople, managers, IT professionals, and engineers can all reap benefits from learning SQL to better position their careers. SQL can open many career paths because it enables individuals to know their businesses better through the data that is driving them. On the business side, interest in SQL can lead to roles that are analytical, managerial, strategic, and research- or project-based. On the IT front, it can lead to roles in database design, database administration, systems engineering, IT project management, and even software development.

CHAPTER 2

Databases

What Is a Database?

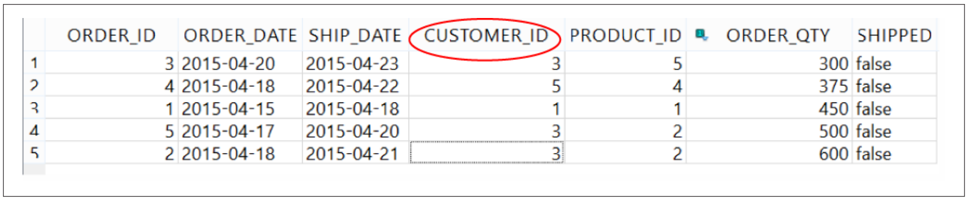
In the broadest definition, a *database* is anything that collects and organizes data. A spreadsheet holding customer bookings is a database, and so is a plain-text file containing flight schedule data. Plain-text data itself can be stored in a variety of formats, including XML and CSV.

Professionally, however, when one refers to a "database" they likely are referring to a *relational database management system* (RDBMS). This term may sound technical and intimidating, but an RDBMS is simply a type of database that holds one or more tables that may have relationships to each other.

Exploring Relational Databases

A table should be a familiar concept. It has columns and rows to store data, much like a spreadsheet. These tables can have relationships to each other, such as an `ORDER` table that refers to a `CUSTOMER` table for customer information.

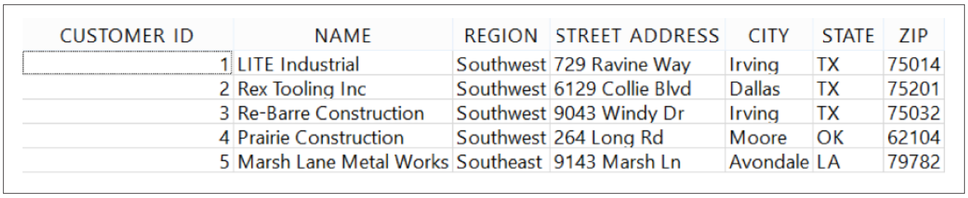
For example, suppose we have an `ORDER` table with a field called `CUSTOMER_ID` (Figure 2-1).



	ORDER_ID	ORDER_DATE	SHIP_DATE	CUSTOMER_ID	PRODUCT_ID	ORDER_QTY	SHIPPED
1	3	2015-04-20	2015-04-23	3	5	300	false
2	4	2015-04-18	2015-04-22	5	4	375	false
3	1	2015-04-15	2015-04-18	1	1	450	false
4	5	2015-04-17	2015-04-20	3	2	500	false
5	2	2015-04-18	2015-04-21	3	2	600	false

Figure 2-1. An ORDER table with a CUSTOMER_ID

We can reasonably expect there to be another table, maybe called CUSTOMER (Figure 2-2), which holds the customer information for each CUSTOMER_ID.



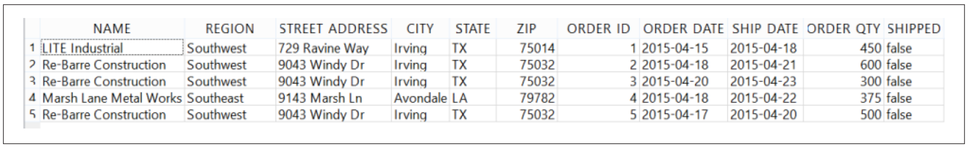
CUSTOMER ID	NAME	REGION	STREET ADDRESS	CITY	STATE	ZIP
1	LITE Industrial	Southwest	729 Ravine Way	Irving	TX	75014
2	Rex Tooling Inc	Southwest	6129 Collie Blvd	Dallas	TX	75201
3	Re-Barre Construction	Southwest	9043 Windy Dr	Irving	TX	75032
4	Prairie Construction	Southwest	264 Long Rd	Moore	OK	62104
5	Marsh Lane Metal Works	Southeast	9143 Marsh Ln	Avondale	LA	79782

Figure 2-2. A CUSTOMER table

When we go through the ORDER table, we can use the CUSTOMER_ID to look up the customer information in the CUSTOMER table. This is the fundamental idea behind a "relational database," where tables may have fields that point to information in other tables. This concept may sound familiar if you've used VLOOKUP in Excel to retrieve information in one sheet from another sheet in a workbook.

Why Separate Tables?

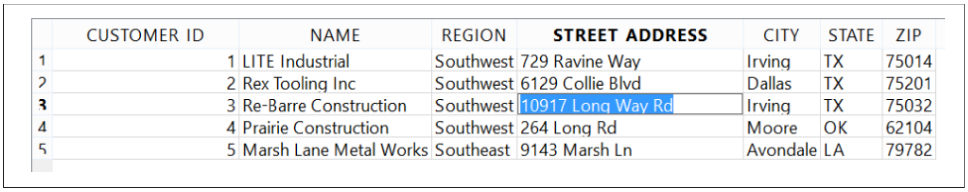
But why are these tables separated and designed this way? The motivation is *normalization*, which is separating the different types of data into their own tables rather than putting them in one table. If we had all information in a single table, it would be redundant, bloated, and very difficult to maintain. Imagine if we stored customer information in the ORDER table. Figure 2-3 shows what it would look like.



	NAME	REGION	STREET ADDRESS	CITY	STATE	ZIP	ORDER ID	ORDER DATE	SHIP DATE	ORDER QTY	SHIPPED
1	LITE Industrial	Southwest	729 Ravine Way	Irving	TX	75014	1	2015-04-15	2015-04-18	450	false
2	Re-Barre Construction	Southwest	9043 Windy Dr	Irving	TX	75032	2	2015-04-18	2015-04-21	600	false
3	Re-Barre Construction	Southwest	9043 Windy Dr	Irving	TX	75032	3	2015-04-20	2015-04-23	300	false
4	Marsh Lane Metal Works	Southeast	9143 Marsh Ln	Avondale	LA	79782	4	2015-04-18	2015-04-22	375	false
5	Re-Barre Construction	Southwest	9043 Windy Dr	Irving	TX	75032	5	2015-04-17	2015-04-20	500	false

Figure 2-3. A table that is not normalized

Notice that for the Re-Barre Construction orders someone had to populate the customer information three times for all three orders (the name, region, street address, city, state, and zip). This is very redundant, takes up unnecessary storage space, and is difficult to maintain. Imagine if a customer had an address change and you had to update all the orders to reflect that. This is why it is better to separate CUSTOMERS and ORDERS into two separate tables. If you need to change a customer's address, you only need to change one record in the CUSTOMER table (Figure 2-4).



	CUSTOMER ID	NAME	REGION	STREET ADDRESS	CITY	STATE	ZIP
1	1	LITE Industrial	Southwest	729 Ravine Way	Irving	TX	75014
2	2	Rex Tooling Inc	Southwest	6129 Collie Blvd	Dallas	TX	75201
3	3	Re-Barre Construction	Southwest	10917 Long Way Rd	Irving	TX	75032
4	4	Prairie Construction	Southwest	264 Long Rd	Moore	OK	62104
5	5	Marsh Lane Metal Works	Southeast	9143 Marsh Ln	Avondale	LA	79782

Figure 2-4. A normalized table

We will explore table relationships again in Chapter 8, and learn how to use the `JOIN` operator to merge tables in a query so the customer information can be viewed alongside the order.

Choosing a Database Solution

Relational databases and SQL are not proprietary. However, there are several companies and communities that have developed their own relational database software, all of which use tables and leverage SQL. Some database solutions are lightweight and simple, storing data in a single file accessible to a small number of users. Other database solutions are massive and run on a server, supporting thousands of users and applications simultaneously. Some database solutions are free and open source, while others require commercial licenses.

For the sake of practicality, we will divide database solutions into two categories: *lightweight* and *centralized*. These are not necessarily the industry vernacular, but they will help clarify the distinction.

Lightweight Databases

If you are seeking a simple solution for one user or a small number of users (e.g., your coworkers), a lightweight database is a good place to start. Lightweight databases have little to no overhead, meaning they have no servers and are very nimble. Databases are typically stored in a file you can share with others, although it starts to break down when multiple people make edits to the file simultaneously. When you run into this problem, you may want to consider migrating to a centralized database.

The two most common lightweight databases are SQLite and Microsoft Access. SQLite is what we will use in this book. It is free, lightweight, and intuitive to use. It is used in most of the devices we touch and can be found in smartphones, satellites, aircraft, and car systems. It has virtually no size limitation and is ideal for environments where it is not used by more than one person (or at most a few people). Among many other uses, SQLite is ideal to learn SQL due to its ease of installation and simplicity.

Microsoft Access has been around for a while and is inferior to SQLite in terms of scalability and performance. But it is heavily used in business environments and worth being familiar with. It has many visual tools for writing queries without using SQL, as well as visual form designers and macro abilities. There are many jobs available to take ownership of Microsoft Access databases and maintain them, as well as migrating them to better database platforms such as MySQL.

Centralized Databases

If you expect tens, hundreds, or thousands of users and applications to use a database simultaneously, lightweight databases are not going to cut it. You need a centralized database that runs on a server and handles a high volume of traffic efficiently. There is a wide array of centralized database solutions to choose from, including the following:

- MySQL
- Microsoft SQL Server
- Oracle
- PostgreSQL
- Teradata
- IBM DB2
- MariaDB

You can install some of these solutions on any computer and turn that computer into a *server*. You can then connect users' computers (also known as *clients*) to the server so they can access the data. The client can send a SQL statement requesting specific data, and the server processes the request and returns the answer. This is a classic *client–server setup*. The client requests something, and the server gives it.

While you can turn any MacBook or cheap PC into a MySQL server, larger traffic volumes require more specialized computers (called *server computers*) optimized for server tasks. These are typically maintained by an IT department whose members administrate and control databases formally deemed critical to the business.

Do not be confused by the term "SQL" being used to brand database platforms such as MySQL, Microsoft SQL Server, and SQLite. SQL is the universal language to work with data on all these platforms. They merely used "SQL" in their names for marketing.

As you enter a workplace, chances are an existing centralized database might exist with information you need, and you will need to request access to it. While we will not be covering centralized databases in this book, the experience between different database solutions should largely be the same. Across all database solutions, you use

SQL to interact with tables in a pretty uniform way, and even the SQL editor tools are somewhat similar. Each solution may have nuances to its implementation of SQL, such as date functionalities, but everything in this book should be universally applicable.

If you ever do need to create a centralized database solution, I would highly recommend MySQL. It is open source, free to use, and straightforward to install and set up. It is used by Facebook, Google, eBay, Twitter, and hundreds of other Silicon Valley companies.

With a conceptual understanding of databases, we can now start working with them. Although we will use SQLite in this book, keep in mind it uses SQL, so the knowledge you gain is applicable to all database platforms.