SQL Crash Course

A Practical Guide From Beginner to Interview-Ready Expert

Vajo Lukic

Companion Code SL
Calle Martinez Campos 16
29001, Malaga,
Spain
info@companioncode.com

Table Of Contents

Terms of Use

By accessing, reading, or using this book, you agree to adhere to the following terms of use. If you don't agree with these terms, you're not allowed to use or access this book.

License: Companion Code SL grants you a non-exclusive, non-transferable, revocable license to use this book for personal, non-commercial purposes only. This means you may not use the book for business uses without direct written permission from the author/publisher.

Copyright: This book is protected by copyright and intellectual property laws. You may not reproduce, distribute, display, perform, publish, license, create derivative works from, transfer, or sell any information, software, products, or services obtained from this book.

Content Usage: You may not use the book's content for commercial purposes, to infringe upon the book's intellectual property rights, or in any way that harms or can be reasonably expected to harm the author or publisher.

Modifications: Any unauthorized modification, alteration, or use of the book or its content is strictly prohibited.

Limitation of Liability: The book is provided "as is," and Companion Code SL shall not be liable for any direct, indirect, consequential, incidental, or punitive damages that come from or are related to your use or inability to use the book. All information, references, and links are provided solely for informational purposes and do not guarantee the content's accuracy, reliability, or any other stated or implied objective. The author, company, and publisher do not guarantee the performance, effectiveness, or relevance of any sites or references mentioned in this book.

Governing Law: These terms will be governed by and construed in accordance with the laws of Spain without giving effect to any principles of conflicts of law.

Changes to Terms of Use: Companion Code SL reserves the right to modify these terms of use at any time. Your continued use of the book after any such changes indicates your acceptance of the new terms.

Contact Information: If you have questions or concerns about these terms of use, please reach out to info@companioncode.com

About This Book

SQL Crash Course: A Practical Guide From Beginner to Interview-Ready Expert is your comprehensive guide designed to take you through the world of SQL, from the fundamentals to advanced concepts and real-world applications.

Suppose you're a complete beginner looking to start your database journey, an experienced developer aiming to sharpen your SQL skills, or a job seeker preparing for technical interviews. In that case, this book has something valuable to offer you.

We have structured this book into three main parts:

1. **SQL Fundamentals**: Here, we lay the groundwork. You'll learn the basics of databases, SQL syntax, and essential operations. This section ensures that even those new to SQL can build a solid foundation.
2. **Advanced SQL Techniques**: This section goes deeper. You'll explore complex queries, performance optimization, and how SQL is used in modern data environments. This section is perfect for those looking to elevate their SQL expertise.
3. **Expert SQL Techniques**: This section takes your skills to the next level. You'll explore advanced topics like data generation, complex transformations, and intricate analytical queries. Expert-level concepts such as window functions, hierarchical data handling, and advanced aggregations will equip you to deal with

the most challenging database problems and stand out as an SQL expert.

4. **SQL Interview Preparation**: The final section is dedicated to helping you ace SQL job interviews. We cover common questions and advanced topics and provide practice problems to boost your confidence.

This book's practical approach sets it apart. We explain each concept with clear examples and real-world scenarios. You'll find exercises and projects throughout to reinforce what you've learned.

We've focused on the SQL skills that'll give you the biggest bang for your buck. There is no fluff - just the most relevant, frequently used aspects of SQL that'll make a real difference in your work.

For the data engineers out there, we've included advanced problem-solving scenarios. You'll explore data generation, manipulation, and complex time-based analyses - the challenges you'll face in your career.

By the time you finish, you'll have a solid grasp of SQL and the practical skills to back it up. You'll be ready to apply your knowledge in real-world situations and confidently face job interviews.

Be patient. Getting good at SQL takes practice. This book is your roadmap, but you'll need to put in the work to master it.

Ready to start your SQL journey? Just click the link below to get your copy:

Get 'SQL Crash Course' on Amazon

How To Use This Book

To get the most out of this book, we recommended the following:

1. **Read in order**: Each chapter builds on the last, so try not to skip around.
2. **Practice as you go**: Set up SQL on your computer (we'll show you how) and try every example. Mess with the code and see what happens.
3. **Do the exercises**: They're spread throughout each chapter. Take them seriously - they'll help you remember what you've learned.
4. **Use online extras**: We've got downloadable datasets and solutions online. Use them. **Download link** is in Appendix A: **Online Resources and Your Thoughts** at the end of the book.
5. **Reflect after each chapter**: Explain what you've learned in your own words.
6. **Apply to real life**: Think about how you can use SQL in your work or projects.
7. **Take mock interviews seriously**: When you reach that part, time yourself and work on solving problems without immediately checking earlier chapters.
8. However, if needed, revisiting earlier chapters is fine—that's how true learning happens.
9. **Take your time**: Learning SQL isn't a race. Set realistic goals and celebrate your progress.

Tips for different readers:

• **Beginners**: Focus on Part I. Make sure you're comfortable with each concept before moving on.

• **Experienced devs**: You can skim the basics, but don't skip them entirely. You might learn something new.

• **Job seekers**: The whole book is helpful, but pay extra attention to Part IV. Use Parts I, II, and III to fill in any gaps.

Eventually, this book is just a tool. Your progress depends on how you use it. Be persistent, practice regularly, and don't be afraid to make mistakes; just learn from them. Stick with it, and you'll master SQL.

Happy learning!

What Sets This Book Apart

"SQL Crash Course: A Practical Guide From Beginner to Interview-Ready Expert" stands out in the crowded field of SQL books for several key reasons:

1. **We focus on what matters**: We cover a lot, but we've zeroed in on the SQL skills you'll use most. It's all about giving you the biggest bang for your buck.
2. **Real-world stuff**: Our examples are based on actual scenarios you'll face at work. You'll learn to apply SQL to real problems, not just theory.
3. **Extra for data engineers**: We've included challenging scenarios specifically for data engineers. Think complex data manipulation and performance tweaks for big data.
4. **Interview prep**: We don't just teach SQL - we help you get ready for job interviews. We've got common questions, advanced topics, and mock interviews to boost your confidence.
5. **Step-by-step learning**: We start simple and build up. Each chapter builds on the last, so you constantly reinforce what you've learned.
6. **Hands-on approach**: We believe in learning by doing, so you'll have plenty of opportunities to practice your learning.
7. **Modern SQL use**: We cover how SQL is used today - in data warehousing, analytics, and big data tech.
8. **Works for different platforms**: We focus on standard SQL but highlight differences between major types (like MySQL and PostgreSQL) when it matters.
9. **Practice makes perfect**: We use plenty of SQL code to explain tricky concepts.

10. **Pro tips**: We share insider knowledge and best practices throughout. It's stuff you'd usually only learn on the job.

Bottom line: If you're just starting or looking to level up, we have designed this book to get you job-ready with SQL. We've packed in everything you need to go from beginner to pro.

PART I: SQL FUNDAMENTALS

Welcome to Part I of "SQL Crash Course: A Practical Guide From Beginner to Interview-Ready Expert." In this section, we'll lay the foundation for your SQL journey. We'll start with the basics and slowly build your understanding of core SQL concepts. By the end of this part, you'll have a solid grasp of SQL fundamentals, preparing you for the more advanced topics.

Introduction to Databases and SQL

In this chapter, we'll introduce you to the world of databases and SQL. We'll explore the fundamental concepts that underpin database systems and understand why SQL is such a crucial tool in modern data management.

What is a database?

Let's cut to the chase: a database is simply an organized data collection stored electronically. If you've ever used a spreadsheet, you're already familiar with the basic idea, but databases go much further.

Imagine trying to keep track of all your friends' contact info, birthdays, and favorite foods in a notebook. It might work for a while, but it'd get messy fast. A database solves this problem by storing information in a structured way that's easy to search, update, and analyze.

At their most basic, databases use tables to organize data. Each table is a lot like a single spreadsheet: It has rows and columns. Each row in a "customer" table might represent one customer, while the columns hold specific information like name, email, and purchase history.

Databases have come a long way since they first appeared in the 1960s. Today, they're the backbone of every app or website you use. When you check your bank balance, order a pizza online, or scroll through social media, you interact with databases behind the scenes.

Why should you care? If you want to work with data in any serious capacity – building apps, analyzing business trends, or just making sense of large amounts of information – you need to understand databases. They're the tool that makes working with big data possible and practical.

As we explore SQL more, you'll see how these basic concepts of tables, rows, and columns combine to create robust systems for managing information.

Just remember, a database is a structured way to store, retrieve, and manage data electronically. It's the foundation on which everything else is built.

The role of SQL in data management

So, we've got our data neatly stored in a database—great. But how do we actually work with it? That's where SQL comes in.

SQL stands for Structured Query Language. Don't let the fancy name fool you - it's basically just a special language for talking to databases. Think of SQL as a really efficient librarian. You tell it what information you need, and it knows exactly where to find it, how to organize it, and how to give it to you.

With SQL, you can do four main things with your data:

1. Create new data
2. Read existing data
3. Update data that's already there
4. Delete data you don't need anymore

Developers often call these CRUD operations. It's just a handy acronym to remember.

But SQL is for more than just moving data around. It's also a powerful tool for analysis. Want to know which products are selling best in each region? Or how many customers signed up last month? SQL queries can crunch those numbers for you in seconds.

SQL has existed since the 1970s, and it's still going strong. Why? Because it's really good at what it does. It's like a Swiss Army knife for data—versatile, reliable, and gets the job done.

Here's the bottom line: if you're serious about working with data, you need to know SQL. It doesn't matter if you're a data analyst, a software developer, or a business intelligence specialist. SQL is the language of data, and it's not going anywhere.

As we move forward, you'll learn how to write SQL queries to extract precisely the information you need from a database. It might seem tricky initially, but you'll be slicing and dicing data like a pro with practice.

SQL: Powering data-driven decisions

SQL (Structured Query Language) is the cornerstone of modern data management, designed to address real-world data challenges efficiently. Let's examine it.

Core principles and practical benefits of SQL are:

1. Structured Data Storage: SQL organizes data into tables, mirroring how we naturally categorize information. Practical benefit: Easily model business entities like customer, order, or inventory.
2. Relational Connections: Keys and relationships link different datasets. Practical benefit: Track order history per customer or calculate product popularity across regions.
3. Declarative Querying: Specify what you need, not how to get it. Practical benefit: Write complex queries without worrying about underlying data retrieval mechanisms.

4. Data Integrity: Enforce rules to maintain data consistency. Practical benefit: Ensure all orders have valid customer IDs, preventing orphaned records.
5. Concurrent Operations: Handle multiple users and operations simultaneously. Practical benefit: Support numerous customer transactions without data conflicts.
6. Transactional Reliability: Guarantee ACID properties for critical operations. Practical benefit: Ensure bank transfers are complete and accurate, even during system failures.

Some of the real-world applications of SQL are:

Business Intelligence: This query quickly identifies top-performing products.

```
SELECT product_name, SUM(quantity * price) as revenue
FROM customer_order
JOIN product ON order.product_id = product.id
GROUP BY product_name
ORDER BY revenue DESC
LIMIT 10;
```

Data Integration: Aggregate daily sales data for reporting.

```
INSERT INTO sale_summary (date, total_sale)
SELECT DATE(order_date), SUM(total_amount)
FROM customer_order
GROUP BY DATE(order_date);
```

Performance Optimization: Speed up date-based queries on large order tables.

```
CREATE INDEX idx_order_date ON order(order_date);
```

SQL's power lies in its ability to handle diverse data tasks, from simple lookups to complex analytics. Master these principles, and you'll have a versatile tool for tackling real-world industry data challenges.

Types of SQL databases

All right, now you know what a database is and why SQL matters. But here's the thing: not all SQL databases are created equal. There are several types, each with its own quirks and strengths.

First up, we've got **MySQL**. It's open-source, which means it's free and has a massive community behind it. If you're building a website or a web app, there's a good chance you'll run into MySQL. It's fast, reliable, and gets the job done for most projects.

Then there's **PostgreSQL**. Think of it as MySQL's more relaxed, more sophisticated cousin. It's also open-source but packs some advanced features that data scientists and large enterprises love. If you're dealing with complex queries or massive datasets, PostgreSQL might be your go-to.

For the big corporate world, there's **Oracle**. It's the heavyweight champion of databases, used by many Fortune 500 companies. It's powerful and can handle almost anything you throw at it, but comes with a hefty price tag.

Microsoft's offering is **SQL Server**. If your company is all-in on Microsoft products, SQL Server plays nice with the rest of the Microsoft ecosystem. It's robust and has some nifty business intelligence tools built-in.

Lastly, there's SQLite. It's like the Swiss Army knife of databases - small, portable, and doesn't need a separate server to run. You'll

often find SQLite in mobile apps or desktop applications where a full-blown database server would be overkill.

You might be wondering, "Which one should I learn?" Here's the good news: **the core SQL language is the same across all of these**. Sure, each has its own unique features, but if you know standard SQL, you can work with any of them.

As we go through this book, we'll focus on standard SQL that works across the board. Where there are significant differences, we'll point them out. The goal is to make you flexible - able to work with whatever database system you encounter in the wild.

Remember, choosing a database often comes down to the specific needs of your project. Size, speed, cost, and scalability all play a role. But don't stress about it too much right now. As you learn SQL and work on different projects, you'll get a feel for which database fits best in various situations.

Setting Up Your SQL Environment

Choosing a SQL database system

Let's discuss picking an SQL database. It's like choosing a car—you want something reliable that fits your needs. For this book, we're going with **PostgreSQL**. Why? It's free, powerful, and plays by the rules of standard SQL better than most.

Earlier we talked about other databases like MySQL, Oracle, or SQL Server. They're all excellent, but PostgreSQL is our sweet spot. It's advanced enough to handle serious work but not so complicated that it'll make your head spin when you're just starting.

Here's the kicker: once you learn PostgreSQL, you'll find it easy to work with other SQL databases, too. The core concepts are the same; it's just the little details that change.

But don't worry if your job or project requires a different database. The SQL you'll learn here will serve you well, no matter what. It's like learning to drive a stick shift—an automatic is a breeze once you've mastered it.

Remember, the goal here isn't to become a PostgreSQL expert. It's to learn SQL in a way that'll set you up for success with any database you encounter down the road.

Installation and configuration

Alright, let's get PostgreSQL up and running on your machine. Don't worry, it's not as scary as it sounds.

For Windows users:

1. Head over to **postgresql.org** and download the installer.
2. Run the installer and follow the prompts. When it asks about components, just stick with the defaults.
3. Choose a **password** for the database superuser. Don't forget to do this!
4. The standard port is **5432**. Unless you've got a good reason, leave it as is.

Mac users, you've got options:

1. You can download the installer from postgresql.org, just like Windows folks.
2. If you're comfortable with the command line, use Homebrew. Just type '**brew install postgresql**', and you're set.

Linux users, you probably know the drill:

1. Use your package manager. On Ubuntu, it's 'sudo apt-get install postgresql.'

Once installed, you'll want to check if it's running. Open a terminal or command prompt and type: **psql -U postgres**

You're in business if you see a **'postgres=#' prompt!**

Stay calm if you hit a snag. Installation hiccups are normal. If you get stuck, PostgreSQL's official docs are a goldmine. Or Google the error message - you're probably not the first to see it.

Luckily, this is a one-time setup. Once it's done, you're ready to start slinging SQL like a pro. And trust me, the payoff is worth the setup hassle.

Using online SQL environments for practice

Setting up PostgreSQL locally on your computer is great, but sometimes, you just want to jump right in without the hassle. That's where online SQL playgrounds come in handy.

My top pick is **DB Fiddle (db-fiddle.com)**. Here's why:

1. It's free and doesn't need an account.
2. You can choose PostgreSQL as your database.
3. It's dead simple to use.

Here's how to use DB Fiddle:

1. Open **db-fiddle.com** in your browser.
2. Pick PostgreSQL and choose the version from the dropdown.
3. In the left panel, write your SQL to create tables.
4. Hit "Run" to set up your database.
5. Now, use the right panel for your queries.
6. Hit "Run" again to see your results.

The cool thing is that you can save your work and share it with a URL. If you're stuck on a problem, send the link to a friend or post it in a forum.

Other options, like **sqliteonline.com** or **pgplayground.com,** are very similar to db-fiddle. Try them out and see what works best for you.

These online tools are great for learning and quick tests but have limits. They're slower than a local setup and might time out on long-running queries. Plus, you can't use them for real projects or sensitive data.

My advice? Use these online playgrounds when you're starting or want to test a quick idea. But as you get deeper into SQL, you'll want to switch to your local PostgreSQL setup for serious work.

Remember, the goal is to learn SQL, not to become an expert in any particular tool. These online environments let you focus on SQL itself without worrying about setup, and that's exactly what you need when you're starting.

Important: Throughout this book, we'll use an e-commerce platform as our primary example. It's a great choice because it's super relevant and has many interconnected parts. Plus, we can jump into more complex examples as we go along.

Our main tables will be:

1. customer
2. product
3. customer_order
4. order_item
5. employee
6. department

The complete e-commerce database and code examples mentioned throughout this book are available online. You'll find the link to access these materials in **Appendix A: Online Resources and Your Thoughts** at the end of the book. This includes all the SQL queries, sample datasets, and printable cheat sheets to help boost your learning. Be sure to check it out to get the most from this book.

Data Definition Language (DDL)

DDL is how we shape our database. It's the SQL commands we use to create, change, or delete the structure of our database objects. Let's focus on the three main DDL commands:

1. CREATE TABLE: This is how we make new tables.
2. ALTER TABLE: Use this to change existing tables.
3. DROP TABLE: This is your go-to when deleting a table.

Now, let's see these in action with our e-commerce database:

CREATE TABLE

```
CREATE TABLE IF NOT EXISTS customer (
    customer_id SERIAL PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    email VARCHAR(100) UNIQUE,
    registration_date DATE DEFAULT CURRENT_DATE
);
```

This command creates our "customer" table. Notice how we've set up a primary key, made some fields required with NOT NULL, and even set a default value for registration_date.

The IF NOT EXISTS clause is super handy. It tells PostgreSQL only to create the table if a table with that name doesn't already exist in the current schema. If the table exists, PostgreSQL will skip this command without throwing an error.

It's a great practice to use this when setting up a database or running scripts you might need to execute multiple times. It helps prevent errors and makes your DDL scripts more robust.

ALTER TABLE

Let's say we want to add a phone number to our customer table:

```
ALTER TABLE customer
ADD COLUMN phone_number VARCHAR(20);
```

Or maybe we want to change the size of the email field:

```
ALTER TABLE customer
ALTER COLUMN email TYPE VARCHAR(150);
```

Adding a constraint which prevents invalid emails being inserted:

```
ALTER TABLE customer
ADD CONSTRAINT check_email_format
CHECK (email ~*
'^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z]{2,}$');
```

DROP TABLE

If we need to remove a table altogether:

```
DROP TABLE IF EXISTS customer;
```

Be careful with this one - it'll delete the table and all its data!

These commands give you the power to structure your database exactly how you need it. As you work with databases, you'll use these DDL commands often to set up and maintain your data structure.

DDL Tips and Hints

Here are some valuable tips about naming conventions, best practices, and common pitfalls for CREATE TABLE statements.

Naming Conventions:

1. Use snake_case for table and column names (e.g., order_item, first_name).
2. Be descriptive but concise (e.g., customer_address, not cust_addr).
3. Avoid SQL keywords as names (e.g., don't name a column "order" or "table").
4. Use singular nouns for table names (e.g., "product" not "products").
5. Prefix tables with a short app name for larger systems (e.g., ecom_customer).

Best Practices:

1. Always specify a primary key.
2. Use appropriate data types (e.g., DATE for dates, not VARCHAR).
3. Set default values where it makes sense (e.g., registration_date DEFAULT CURRENT_DATE).
4. Use NOT NULL for columns that should always have a value.
5. Create indexes on columns you'll frequently search or join on.

Good examples:

```
CREATE TABLE product (
 product_id SERIAL PRIMARY KEY,
 product_name VARCHAR(100) NOT NULL,
 product_code VARCHAR(50) NOT NULL,
 category VARCHAR(50) NOT NULL,
 price DECIMAL(10,2) NOT NULL,
 stock_quantity INT NOT NULL,
 description TEXT,
 created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE customer_order (
    order_id SERIAL PRIMARY KEY,
    customer_id INT,
    order_date DATE DEFAULT CURRENT_DATE,
    total_amount DECIMAL(10,2),
    FOREIGN KEY (customer_id) REFERENCES
customer(customer_id)
);
```

Common Mistakes and Gotchas:

1. Forgetting to set a primary key.
2. Using VARCHAR without specifying a length.
3. Choosing inappropriate data types (e.g., FLOAT for currency).
4. Not considering future needs (e.g., making a phone number field too short).
5. Overusing NULL when a default value would be better.

Bad example:

```
CREATE TABLE Products (
    ID INT,
    Name VARCHAR,
    price FLOAT
);
```

This table's name is in uppercase and plural. It lacks a primary key, uses an unbounded VARCHAR, and uses FLOAT for price (which can lead to rounding errors with currency).

A well-designed table makes querying and maintaining your database much easier in the long run. Before creating it, take the time to think through your table structure.

Thanks for reading this sample! If these ideas resonated with you, there's plenty more where that came from. The complete book is available on Amazon, packed with additional tips, real-world examples, and practical exercises to help you master SQL. Ready to continue your journey? Just click the link below to get your copy:

Get 'SQL Crash Course' on Amazon

Your path to becoming SQL Expert starts now!