

O'REILLY®

SQL for Data Analysis

Advanced Techniques for Transforming Data
into Insights



Cathy Tanimura



SQL for Data Analysis

With the explosion of data, computing power, and cloud data warehouses, SQL has become an even more indispensable tool for the savvy analyst or data scientist. This practical book reveals new and hidden ways to improve your SQL skills, solve problems, and make the most of SQL as part of your workflow.

You'll learn how to use both common and exotic SQL functions such as joins, window functions, subqueries, and regular expressions in new, innovative ways—as well as how to combine SQL techniques to accomplish your goals faster, with understandable code. If you work with SQL databases, this is a must-have reference.

- Learn the key steps for preparing your data for analysis
- Perform time series analysis using SQL's date and time manipulations
- Use cohort analysis to investigate how groups change over time
- Use SQL's powerful functions and operators for text analysis
- Detect outliers in your data and replace them with alternate values
- Establish causality using experiment analysis, also known as A/B testing

Cathy Tanimura has been analyzing data for over 20 years across a wide range of industries, from finance to B2B software to consumer services. With a passion for connecting people and organizations to the data they need, Cathy has built and managed data teams and data infrastructure at several leading tech companies. She has experience analyzing data with SQL across most of the major proprietary and open source databases.

"I can't count the number of 'AHA!' moments I had, despite my 20 years of doing analysis in various SQL environments. I will be buying a copy for every current and future member of my team."

—Stuart Kim-Brown
PhD, B2C and SaaS Product Analytics Leader

"So great to finally see a SQL book written specifically for those looking to do data analysis. Any aspiring analyst or data scientist would benefit from reading about the various methods of data analysis that can be tackled with SQL. Detailed examples and code make this book both a great introduction and reference."

—Dan Voorhies
Director of Analytics, Zillow

Shared by TRS

DATA

US \$59.99 CAN \$79.99

ISBN: 978-1-492-08878-3



Twitter: @oreillymedia
facebook.com/oreilly

SQL for Data Analysis

*Advanced Techniques for Transforming
Data into Insights*

Cathy Tanimura

Beijing • Boston • Farnham • Sebastopol • Tokyo

O'REILLY®

SQL for Data Analysis

by Cathy Tanimura

Copyright © 2021 Cathy Tanimura. All rights reserved.

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://oreilly.com>). For more information, contact our corporate/institutional sales department: 800-998-9938 or corporate@oreilly.com.

Acquisitions Editor: Andy Kwan

Development Editors: Amelia Blevins and Shira Evans

Production Editor: Kristen Brown

Copyeditor: Arthur Johnson

Proofreader: Paula L. Fleming

Indexer: Ellen Troutman-Zaig

Interior Designer: David Futato

Cover Designer: Karen Montgomery

Illustrator: Kate Dullea

September 2021: First Edition

Revision History for the First Edition

2021-09-09: First Release

See <http://oreilly.com/catalog/errata.csp?isbn=9781492088783> for release details.

The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. *SQL for Data Analysis*, the cover image, and related trade dress are trademarks of O'Reilly Media, Inc.

The views expressed in this work are those of the author, and do not represent the publisher's views. While the publisher and the author have used good faith efforts to ensure that the information and instructions contained in this work are accurate, the publisher and the author disclaim all responsibility for errors or omissions, including without limitation responsibility for damages resulting from the use of or reliance on this work. Use of the information and instructions contained in this work is at your own risk. If any code samples or other technology this work contains or describes is subject to open source licenses or the intellectual property rights of others, it is your responsibility to ensure that your use thereof complies with such licenses and/or rights.

978-1-492-08878-3

[LSI]

Table of Contents

Preface.....	ix
1. Analysis with SQL.....	1
What Is Data Analysis?	1
Why SQL?	4
What Is SQL?	4
Benefits of SQL	7
SQL Versus R or Python	8
SQL as Part of the Data Analysis Workflow	9
Database Types and How to Work with Them	12
Row-Store Databases	13
Column-Store Databases	15
Other Types of Data Infrastructure	16
Conclusion	17
2. Preparing Data for Analysis.....	19
Types of Data	20
Database Data Types	20
Structured Versus Unstructured	22
Quantitative Versus Qualitative Data	22
First-, Second-, and Third-Party Data	23
Sparse Data	24
SQL Query Structure	25
Profiling: Distributions	27
Histograms and Frequencies	28
Binning	31
n-Tiles	33

Profiling: Data Quality	35
Detecting Duplicates	36
Deduplication with GROUP BY and DISTINCT	38
Preparing: Data Cleaning	39
Cleaning Data with CASE Transformations	39
Type Conversions and Casting	42
Dealing with Nulls: coalesce, nullif, nvl Functions	45
Missing Data	47
Preparing: Shaping Data	52
For Which Output: BI, Visualization, Statistics, ML	52
Pivoting with CASE Statements	53
Unpivoting with UNION Statements	55
pivot and unpivot Functions	57
Conclusion	59
3. Time Series Analysis.....	61
Date, Datetime, and Time Manipulations	62
Time Zone Conversions	62
Date and Timestamp Format Conversions	64
Date Math	68
Time Math	71
Joining Data from Different Sources	72
The Retail Sales Data Set	74
Trending the Data	75
Simple Trends	75
Comparing Components	77
Percent of Total Calculations	86
Indexing to See Percent Change over Time	90
Rolling Time Windows	95
Calculating Rolling Time Windows	97
Rolling Time Windows with Sparse Data	102
Calculating Cumulative Values	104
Analyzing with Seasonality	107
Period-over-Period Comparisons: YoY and MoM	109
Period-over-Period Comparisons: Same Month Versus Last Year	112
Comparing to Multiple Prior Periods	116
Conclusion	119
4. Cohort Analysis.....	121
Cohorts: A Useful Analysis Framework	122
The Legislators Data Set	125

Retention	127
SQL for a Basic Retention Curve	128
Adjusting Time Series to Increase Retention Accuracy	131
Cohorts Derived from the Time Series Itself	137
Defining the Cohort from a Separate Table	142
Dealing with Sparse Cohorts	146
Defining Cohorts from Dates Other Than the First Date	151
Related Cohort Analyses	153
Survivorship	154
Returnship, or Repeat Purchase Behavior	158
Cumulative Calculations	163
Cross-Section Analysis, Through a Cohort Lens	166
Conclusion	174
5. Text Analysis.....	175
Why Text Analysis with SQL?	175
What Is Text Analysis?	176
Why SQL Is a Good Choice for Text Analysis	176
When SQL Is Not a Good Choice	177
The UFO Sightings Data Set	178
Text Characteristics	179
Text Parsing	182
Text Transformations	187
Finding Elements Within Larger Blocks of Text	195
Wildcard Matches: LIKE, ILIKE	195
Exact Matches: IN, NOT IN	200
Regular Expressions	203
Constructing and Reshaping Text	218
Concatenation	218
Reshaping Text	222
Conclusion	226
6. Anomaly Detection.....	227
Capabilities and Limits of SQL for Anomaly Detection	228
The Data Set	229
Detecting Outliers	230
Sorting to Find Anomalies	231
Calculating Percentiles and Standard Deviations to Find Anomalies	234
Graphing to Find Anomalies Visually	241
Forms of Anomalies	250
Anomalous Values	250

Anomalous Counts or Frequencies	254
Anomalies from the Absence of Data	258
Handling Anomalies	260
Investigation	260
Removal	260
Replacement with Alternate Values	262
Rescaling	264
Conclusion	266
7. Experiment Analysis.....	267
Strengths and Limits of Experiment Analysis with SQL	269
The Data Set	270
Types of Experiments	272
Experiments with Binary Outcomes: The Chi-Squared Test	272
Experiments with Continuous Outcomes: The t-Test	274
Challenges with Experiments and Options for Rescuing Flawed Experiments	276
Variant Assignment	277
Outliers	278
Time Boxing	279
Repeated Exposure Experiments	280
When Controlled Experiments Aren't Possible: Alternative Analyses	282
Pre-/Post-Analysis	282
Natural Experiment Analysis	284
Analysis of Populations Around a Threshold	286
Conclusion	286
8. Creating Complex Data Sets for Analysis.....	287
When to Use SQL for Complex Data Sets	287
Advantages of Using SQL	288
When to Build into ETL Instead	288
When to Put Logic in Other Tools	290
Code Organization	292
Commenting	292
Capitalization, Indentation, Parentheses, and Other Formatting Tricks	293
Storing Code	296
Organizing Computations	296
Understanding Order of SQL Clause Evaluation	296
Subqueries	300
Temporary Tables	302
Common Table Expressions	303
grouping sets	305

Managing Data Set Size and Privacy Concerns	308
Sampling with %, mod	308
Reducing Dimensionality	310
PII and Data Privacy	314
Conclusion	316
9. Conclusion.....	317
Funnel Analysis	317
Churn, Lapse, and Other Definitions of Departure	319
Basket Analysis	323
Resources	325
Books and Blogs	325
Data Sets	326
Final Thoughts	327
Index.....	329

Preface

Over the past 20 years, I've spent many of my working hours manipulating data with SQL. For most of those years, I've worked in technology companies spanning a wide range of consumer and business-to-business industries. In that time, volumes of data have increased dramatically, and the technology I get to use has improved by leaps and bounds. Databases are faster than ever, and the reporting and visualization tools used to communicate the meaning in the data are more powerful than ever. One thing that has remained remarkably constant, however, is SQL being a key part of my toolbox.

I remember when I first learned SQL. I started my career in finance, where spreadsheets rule, and I'd gotten pretty good at writing formulas and memorizing all those keyboard shortcuts. One day I totally geeked out and Ctrl- and Alt-clicked every key on my keyboard just to see what would happen (and then created a cheat sheet for my peers). That was part fun and part survival: the faster I was with my spreadsheets, the more likely I would be to finish my work before midnight so I could go home and get some sleep. Spreadsheet mastery got me in the door at my next role, a startup where I was first introduced to databases and SQL.

Part of my role involved crunching inventory data in spreadsheets, and thanks to early internet scale, the data sets were sometimes tens of thousands of rows. This was “big data” at the time, at least for me. I got in the habit of going for a cup of coffee or for lunch while my computer’s CPU was occupied with running its vlookup magic. One day my manager went on vacation and asked me to tend to the data warehouse he’d built on his laptop using Access. Refreshing the data involved a series of steps: running SQL queries in a portal, loading the resulting csv files into the database, and then refreshing the spreadsheet reports. After the first successful load, I started tinkering, trying to understand how it worked, and pestering the engineers to show me how to modify the SQL queries.

I was hooked, and even when I thought I might change directions with my career, I've kept coming back to data. Manipulating data, answering questions, helping my

colleagues work better and smarter, and learning about businesses and the world through sets of data have never stopped feeling fun and exciting.

When I started working with SQL, there weren't many learning resources. I got a book on basic syntax, read it in a night, and from there mostly learned through trial and error. Back in the days when I was learning, I queried production databases directly and brought the website down more than once with my overly ambitious (or more likely just poorly written) SQL. Fortunately my skills improved, and over the years I learned to work forward from the data in tables, and backward from the output needed, solving technical and logic challenges and puzzles to write queries that returned the right data. I ended up designing and building data warehouses to gather data from different sources and avoid bringing down critical production databases. I've learned a lot about when and how to aggregate data before writing the SQL query and when to leave data in a more raw form.

I've compared notes with others who got into data around the same time, and it's clear we mostly learned in the same ad hoc way. The lucky among us had peers with whom to share techniques. Most SQL texts are either introductory and basic (there's definitely a place for these!) or else aimed at database developers. There are few resources for advanced SQL users who are focused on analysis work. Knowledge tends to be locked up in individuals or small teams. A goal of this book is to change that, giving practitioners a reference for how to solve common analysis problems with SQL, and I hope inspiring new inquiries into data using techniques you might not have seen before.

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic

Indicates new terms, URLs, email addresses, filenames, file extensions, and keywords.

Constant width

Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, environment variables, and statements.

Constant width bold

Shows commands or other text that should be typed literally by the user.

Constant width italic

Shows text that should be replaced with user-supplied values or by values determined by context.



This element signifies a tip or suggestion.



This element signifies a general note.



This element indicates a warning or caution.

Using Code Examples

Supplemental material (code examples, exercises, etc.) is available for download at https://github.com/cathytanimura/sql_book.

If you have a technical question or a problem using the code examples, please send email to bookquestions@oreilly.com.

This book is here to help you get your job done. In general, if example code is offered with this book, you may use it in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but generally do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: “*SQL for Data Analysis* by Cathy Tanimura (O'Reilly). Copyright 2021 Cathy Tanimura, 978-1-492-08878-3.”

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at permissions@oreilly.com.

O'Reilly Online Learning



For more than 40 years, *O'Reilly Media* has provided technology and business training, knowledge, and insight to help companies succeed.

Our unique network of experts and innovators share their knowledge and expertise through books, articles, and our online learning platform. O'Reilly's online learning platform gives you on-demand access to live training courses, in-depth learning paths, interactive coding environments, and a vast collection of text and video from O'Reilly and 200+ other publishers. For more information, visit <http://oreilly.com>.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at <https://oreil.ly/sql-data-analysis>.

Email bookquestions@oreilly.com to comment or ask technical questions about this book.

For news and information about our books and courses, visit <http://oreilly.com>.

Find us on Facebook: <http://facebook.com/oreilly>

Follow us on Twitter: <http://twitter.com/oreillymedia>

Watch us on YouTube: <http://www.youtube.com/oreillymedia>

Acknowledgments

This book wouldn't have been possible without the efforts of a number of people at O'Reilly. Andy Kwan recruited me to this project. Amelia Blevins and Shira Evans guided me through the process and gave helpful feedback along the way. Kristen Brown shepherded the book through the production process. Arthur Johnson improved the quality and clarity of the text and inadvertently made me think more deeply about SQL keywords.

Many colleagues over the years played an important role in my SQL journey, and I'm grateful for their tutorials, tips, and shared code, and the time spent brainstorming ways to solve analysis problems over the years. Sharon Lin opened my eyes to regular expressions. Elyse Gordon gave me lots of book-writing advice. Dave Hoch and our conversations about experiment analysis inspired [Chapter 7](#). Dan, Jim, and Stu from the Star Chamber have long been my favorite guys to geek out with. I'm also grateful for all of the colleagues who asked hard questions over the years—and once those were answered, asked even harder ones.

I'd like to thank my husband Rick, son Shea, daughters Lily and Fiona, and mom Janet for their love, encouragement, and most of all the gift of time to work on this book. Amy, Halle, Jessi, and the Den of Slack kept me sane and laughing through months of writing and pandemic lockdown.

CHAPTER 1

Analysis with SQL

If you're reading this book, you're probably interested in data analysis and in using SQL to accomplish it. You may be experienced with data analysis but new to SQL, or perhaps you're experienced with SQL but new to data analysis. Or you may be new to both topics entirely. Whatever your starting point, this chapter lays the groundwork for the topics covered in the rest of the book and makes sure we have a common vocabulary. I'll start with a discussion of what data analysis is and then move on to a discussion of SQL: what it is, why it's so popular, how it compares to other tools, and how it fits into data analysis. Then, since modern data analysis is so intertwined with the technologies that have enabled it, I'll conclude with a discussion of different types of databases that you may encounter in your work, why they're used, and what all of that means for the SQL you write.

What Is Data Analysis?

Collecting and storing data for analysis is a very human activity. Systems to track stores of grain, taxes, and the population go back thousands of years, and the [roots of statistics](#) date back hundreds of years. Related disciplines, including statistical process control, operations research, and cybernetics, exploded in the 20th century. Many different names are used to describe the discipline of data analysis, such as business intelligence (BI), analytics, data science, and decision science, and practitioners have a range of job titles. Data analysis is also done by marketers, product managers, business analysts, and a variety of other people. In this book, I'll use the terms *data analyst* and *data scientist* interchangeably to mean the person working with SQL to understand data. I will refer to the software used to build reports and dashboards as *BI tools*.

Data analysis in the contemporary sense was enabled by, and is intertwined with, the history of computing. Trends in both research and commercialization have shaped it,

and the story includes a who's who of researchers and major companies, which we'll talk about in the section on SQL. Data analysis blends the power of computing with techniques from traditional statistics. Data analysis is part data discovery, part data interpretation, and part data communication. Very often the purpose of data analysis is to improve decision making, by humans and increasingly by machines through automation.

Sound methodology is critical, but analysis is about more than just producing the right number. It's about curiosity, asking questions, and the "why" behind the numbers. It's about patterns and anomalies, discovering and interpreting clues about how businesses and humans behave. Sometimes analysis is done on a data set gathered to answer a specific question, as in a scientific setting or an online experiment. Analysis is also done on data that is generated as a result of doing business, as in sales of a company's products, or that is generated for analytics purposes, such as user interaction tracking on websites and mobile apps. This data has a wide range of possible applications, from troubleshooting to planning user interface (UI) improvements, but it often arrives in a format and volume such that the data needs processing before yielding answers. [Chapter 2](#) will cover preparing data for analysis, and [Chapter 8](#) will discuss some of the ethical and privacy concerns with which all data practitioners should be familiar.

It's hard to think of an industry that hasn't been touched by data analysis: manufacturing, retail, finance, health care, education, and even government have all been changed by it. Sports teams have employed data analysis since the early years of Billy Beane's term as general manager of the Oakland Athletics, made famous by Michael Lewis's book *Moneyball* (Norton). Data analysis is used in marketing, sales, logistics, product development, user experience design, support centers, human resources, and more. The combination of techniques, applications, and computing power has led to the explosion of related fields such as data engineering and data science.

Data analysis is by definition done on historical data, and it's important to remember that the past doesn't necessarily predict the future. The world is dynamic, and organizations are dynamic as well—new products and processes are introduced, competitors rise and fall, sociopolitical climates change. Criticisms are leveled against data analysis for being backward looking. Though that characterization is true, I have seen organizations gain tremendous value from analyzing historical data. Mining historical data helps us understand the characteristics and behavior of customers, suppliers, and processes. Historical data can help us develop informed estimates and predicted ranges of outcomes, which will sometimes be wrong but quite often will be right. Past data can point out gaps, weaknesses, and opportunities. It allows organizations to optimize, save money, and reduce risk and fraud. It can also help organizations find opportunity, and it can become the building blocks of new products that delight customers.



Organizations that don't do some form of data analysis are few and far between these days, but there are still some holdouts. Why do some organizations not use data analysis? One argument is the cost-to-value ratio. Collecting, processing, and analyzing data takes work and some level of financial investment. Some organizations are too new, or they're too haphazard. If there isn't a consistent process, it's hard to generate data that's consistent enough to analyze. Finally, there are ethical considerations. Collecting or storing data about certain people in certain situations may be regulated or even banned. Data about children and health-care interventions is sensitive, for example, and there are extensive regulations around its collection. Even organizations that are otherwise data driven need to take care around customer privacy and to think hard about what data should be collected, why it is needed, and how long it should be stored. Regulations such as the European Union's General Data Protection Regulation, or GDPR, and the California Consumer Privacy Act, or CCPA, have changed the way businesses think about consumer data. We'll discuss these regulations in more depth in [Chapter 8](#). As data practitioners, we should always be thinking about the ethical implications of our work.

When working with organizations, I like to tell people that data analysis is not a project that wraps up at a fixed date—it's a way of life. Developing a data-informed mindset is a process, and reaping the rewards is a journey. Unknowns become known, difficult questions are chipped away at until there are answers, and the most critical information is embedded in dashboards that power tactical and strategic decisions. With this information, new and harder questions are asked, and then the process repeats.

Data analysis is both accessible for those looking to get started and hard to master. The technology can be learned, particularly SQL. Many problems, such as optimizing marketing spend or detecting fraud, are familiar and translate across businesses. Every organization is different and every data set has quirks, so even familiar problems can pose new challenges. Communicating results is a skill. Learning to make good recommendations and becoming a trusted partner to an organization take time. In my experience, simple analysis presented persuasively has more impact than sophisticated analysis presented poorly. Successful data analysis also requires partnership. You can have great insights, but if there is no one to execute on them, you haven't really made an impact. Even with all the technology, it's still about people, and relationships matter.

Why SQL?

This section describes what SQL is, the benefits of using it, how it compares to other languages commonly used for analysis, and finally how SQL fits into the analysis workflow.

What Is SQL?

SQL is the language used to communicate with databases. The acronym stands for Structured Query Language and is pronounced either like “sequel” or by saying each letter, as in “ess cue el.” This is only the first of many controversies and inconsistencies surrounding SQL that we’ll see, but most people will know what you mean regardless of how you say it. There is some debate as to whether SQL is or isn’t a programming language. It isn’t a general purpose language in the way that C or Python are. SQL without a database and data in tables is just a text file. SQL can’t build a website, but it is powerful for working with data in databases. On a practical level, what matters most is that SQL can help you get the job of data analysis done.

IBM was the first to develop SQL databases, from the relational model invented by Edgar Codd in the 1960s. The relational model was a theoretical description for managing data using relationships. By creating the first databases, IBM helped to advance the theory, but it also had commercial considerations, as did Oracle, Microsoft, and every other company that has commercialized a database since. From the beginning, there has been tension between computer theory and commercial reality. SQL became an International Organization for Standards (ISO) standard in 1987 and an American National Standards Institute (ANSI) standard in 1986. Although all major databases start from these standards in their implementation of SQL, many have variations and functions that make life easier for the users of those databases. These come at the cost of making SQL more difficult to move between databases without some modifications.

SQL is used to access, manipulate, and retrieve data from objects in a database. Databases can have one or more *schemas*, which provide the organization and structure and contain other objects. Within a schema, the objects most commonly used in data analysis are tables, views, and functions. Tables contain fields, which hold the data. Tables may have one or more *indexes*; an index is a special kind of data structure that allows data to be retrieved more efficiently. Indexes are usually defined by a database administrator. Views are essentially stored queries that can be referenced in the same way as a table. Functions allow commonly used sets of calculations or procedures to be stored and easily referenced in queries. They are usually created by a database administrator, or DBA. [Figure 1-1](#) gives an overview of the organization of databases.

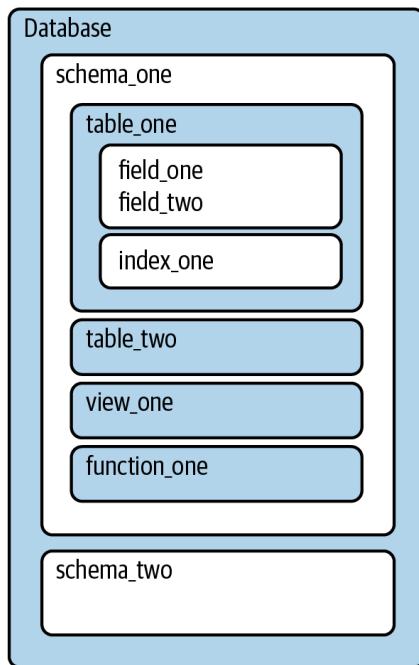


Figure 1-1. Overview of database organization and objects in a database

To communicate with databases, SQL has four sublanguages for tackling different jobs, and these are mostly standard across database types. Most people who work in data analysis don't need to recall the names of these sublanguages on a daily basis, but they might come up in conversation with database administrators or data engineers, so I'll briefly introduce them. The commands all work fluidly together, and some may coexist in the same SQL statement.

DQL, or *data query language*, is what this book is mainly about. It's used for *querying* data, which you can think of as using code to ask questions of a database. DQL commands include *SELECT*, which will be familiar to prior users of SQL, but the acronym DQL is not frequently used in my experience. SQL queries can be as short as a single line or span many tens of lines. SQL queries can access a single table (or view), can combine data from multiple tables through the use of joins, and can also query across multiple schemas in the same database. SQL queries generally cannot query across databases, but in some cases clever network settings or additional software can be used to retrieve data from multiple sources, even databases of different types. SQL queries are self-contained and, apart from tables, do not reference variables or outputs from previous steps not contained in the query, unlike scripting languages.

DDL, or *data definition language*, is used to create and modify tables, views, users, and other objects in the database. It affects the structure but not the contents. There are three common commands: *CREATE*, *ALTER*, and *DROP*. *CREATE* is used to

make new objects. *ALTER* changes the structure of an object, such as by adding a column to a table. *DROP* deletes the entire object and its structure. You might hear DBAs and data engineers talk about working with DDLs—this is really just shorthand for the files or pieces of code that do the creates, alters, or drops. An example of how DDL is used in the context of analysis is the code to create temporary tables.

DCL, or *data control language*, is used for access control. Commands include *GRANT* and *REVOKE*, which give permission and remove permission, respectively. In an analysis context, *GRANT* might be needed to allow a colleague to query a table you created. You might also encounter such a command when someone has told you a table exists in the database but you can't see it—permissions might need to be *GRANTED* to your user.

DML, or *data manipulation language*, is used to act on the data itself. The commands are *INSERT*, *UPDATE*, and *DELETE*. *INSERT* adds new records and is essentially the “load” step in extract, transform, load (ETL). *UPDATE* changes values in a field, and *DELETE* removes rows. You will encounter these commands if you have any kind of self-managed tables—temp tables, sandbox tables—or if you find yourself in the role of both owner and analyzer of the database.

These four sublanguages are present in all major databases. In this book, I'll focus mainly on DQL. We will touch on a few DDL and DML commands in [Chapter 8](#), and you will also see some examples in the [GitHub site for the book](#), where they are used to create and populate the data used in examples. Thanks to this common set of commands, SQL code written for any database will look familiar to anyone used to working with SQL. However, reading SQL from another database may feel a bit like listening to someone who speaks the same language as you but comes from another part of the country or the world. The basic structure of the language is the same, but the slang is different, and some words have different meanings altogether. Variations in SQL from database to database are often termed *dialects*, and database users will reference Oracle SQL, MSSQL, or other dialects.

Still, once you know SQL, you can work with different database types as long as you pay attention to details such as the handling of nulls, dates, and timestamps; the division of integers; and case sensitivity.

This book uses PostgreSQL, or Postgres, for the examples, though I will try to point out where the code would be meaningfully different in other types of databases. You can install [Postgres](#) on a personal computer in order to follow along with the examples.

Benefits of SQL

There are many good reasons to use SQL for data analysis, from computing power to its ubiquity in data analysis tools and its flexibility.

Perhaps the best reason to use SQL is that much of the world's data is already in databases. It's likely your own organization has one or more databases. Even if data is not already in a database, loading it into one can be worthwhile in order to take advantage of the storage and computing advantages, especially when compared to alternatives such as spreadsheets. Computing power has exploded in recent years, and data warehouses and data infrastructure have evolved to take advantage of it. Some newer cloud databases allow massive amounts of data to be queried in memory, speeding things up further. The days of waiting minutes or hours for query results to return may be over, though analysts may just write more complex queries in response.

SQL is the de facto standard for interacting with databases and retrieving data from them. A wide range of popular software connects to databases with SQL, from spreadsheets to BI and visualization tools and coding languages such as Python and R (discussed in the next section). Due to the computing resources available, performing as much data manipulation and aggregation as possible in the database often has advantages downstream. We'll discuss strategies for building complex data sets for downstream tools in depth in [Chapter 8](#).

The basic SQL building blocks can be combined in an endless number of ways. Starting with a relatively small number of building blocks—the syntax—SQL can accomplish a wide array of tasks. SQL can be developed iteratively, and it's easy to review the results as you go. It may not be a full-fledged programming language, but it can do a lot, from transforming data to performing complex calculations and answering questions.

Last, SQL is relatively easy to learn, with a finite amount of syntax. You can learn the basic keywords and structure quickly and then hone your craft over time working with varied data sets. Applications of SQL are virtually infinite, when you take into account the range of data sets in the world and the possible questions that can be asked of data. SQL is taught in many universities, and many people pick up some skills on the job. Even employees who don't already have SQL skills can be trained, and the learning curve may be easier than that for other programming languages. This makes storing data for analysis in relational databases a logical choice for organizations.

SQL Versus R or Python

While SQL is a popular language for data analysis, it isn't the only choice. R and Python are among the most popular of the other languages used for data analysis. R is a statistical and graphing language, while Python is a general-purpose programming language that has strengths in working with data. Both are open source, can be installed on a laptop, and have active communities developing packages, or extensions, that tackle various data manipulation and analysis tasks. Choosing between R and Python is beyond the scope of this book, but there are many discussions online about the relative advantages of each. Here I will consider them together as coding-language alternatives to SQL.

One major difference between SQL and other coding languages is where the code runs and, therefore, how much computing power is available. SQL always runs on a database server, taking advantage of all its computing resources. For doing analysis, R and Python are usually run locally on your machine, so computing resources are capped by whatever is available locally. There are, of course, lots of exceptions: databases can run on laptops, and R and Python can be run on servers with more resources. When you are performing anything other than the simplest analysis on large data sets, pushing work onto a database server with more resources is a good option. Since databases are usually set up to continually receive new data, SQL is also a good choice when a report or dashboard needs to update periodically.

A second difference is in how data is stored and organized. Relational databases always organize data into rows and columns within tables, so SQL assumes this structure for every query. R and Python have a wider variety of ways to store data, including variables, lists, and dictionaries, among other options. These provide more flexibility, but at the cost of a steeper learning curve. To facilitate data analysis, R has data frames, which are similar to database tables and organize data into rows and columns. The pandas package makes DataFrames available in Python. Even when other options are available, the table structure remains valuable for analysis.

Looping is another major difference between SQL and most other computer programming languages. A *loop* is an instruction or a set of instructions that repeats until a specified condition is met. SQL aggregations implicitly loop over the set of data, without any additional code. We will see later how the lack of ability to loop over fields can result in lengthy SQL statements when pivoting or unpivoting data. While deeper discussion is beyond the scope of this book, some vendors have created extensions to SQL, such as PL/SQL in Oracle and T-SQL in Microsoft SQL Server, that allow functionality such as looping.

A drawback of SQL is that your data must be in a database,¹ whereas R and Python can import data from files stored locally or can access files stored on servers or websites. This is convenient for many one-off projects. A database can be installed on a laptop, but this does add an extra layer of overhead. In the other direction, packages such as `dbplyr` for R and `SQLAlchemy` for Python allow programs written in those languages to connect to databases, execute SQL queries, and use the results in further processing steps. In this sense, R or Python can be complementary to SQL.

R and Python both have sophisticated statistical functions that are either built in or available in packages. Although SQL has, for example, functions to calculate average and standard deviation, calculations of p-values and statistical significance that are needed in experiment analysis (discussed in [Chapter 7](#)) cannot be performed with SQL alone. In addition to sophisticated statistics, machine learning is another area that is better tackled with one of these other coding languages.

When deciding whether to use SQL, R, or Python for an analysis, consider:

- Where is the data located—in a database, a file, a website?
- What is the volume of data?
- Where is the data going—into a report, a visualization, a statistical analysis?
- Will it need to be updated or refreshed with new data? How often?
- What does your team or organization use, and how important is it to conform to existing standards?

There is no shortage of debate around which languages and tools are best for doing data analysis or data science. As with many things, there's often more than one way to accomplish an analysis. Programming languages evolve and change in popularity, and we're lucky to live and work in a time with so many good choices. SQL has been around for a long time and will likely remain popular for years to come. The ultimate goal is to use the best available tool for the job. This book will help you get the most out of SQL for data analysis, regardless of what else is in your toolkit.

SQL as Part of the Data Analysis Workflow

Now that I've explained what SQL is, discussed some of its benefits, and compared it to other languages, we'll turn to a discussion of where SQL fits in the data analysis process. Analysis work always starts with a question, which may be about how many new customers have been acquired, how sales are trending, or why some users stick around for a long time while others try a service and never return. Once the question is framed, we consider where the data originated, where the data is stored, the

¹ There are some newer technologies that allow SQL queries on data stored in nonrelational sources.

analysis plan, and how the results will be presented to the audience. [Figure 1-2](#) shows the steps in the process. Queries and analysis are the focus of this book, though I will discuss the other steps briefly in order to put the queries and analysis stage into a broader context.

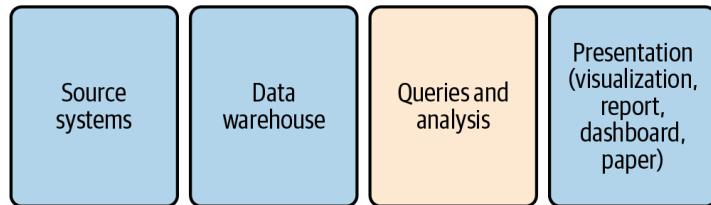


Figure 1-2. Steps in the data analysis process

First, data is generated by *source systems*, a term that includes any human or machine process that generates data of interest. Data can be generated by people by hand, such as when someone fills out a form or takes notes during a doctor's visit. Data can also be machine generated, such as when an application database records a purchase, an event-streaming system records a website click, or a marketing management tool records an email open. Source systems can generate many different types and formats of data, and [Chapter 2](#) will discuss them, and how the type of source may impact the analysis, in more detail.

The second step is moving the data and storing it in a database for analysis. I will use the terms *data warehouse*, which is a database that consolidates data from across an organization into a central repository, and *data store*, which refers to any type of data storage system that can be queried. Other terms you might come across are *data mart*, which is typically a subset of a data warehouse, or a more narrowly focused data warehouse; and *data lake*, a term that can mean either that data resides in a file storage system or that it is stored in a database but without the degree of data transformation that is common in data warehouses. Data warehouses range from small and simple to huge and expensive. A database running on a laptop will be sufficient for you to follow along with the examples in this book. What matters is having the data you need to perform an analysis together in one place.



Usually a person or team is responsible for getting data into the data warehouse. This process is called *ETL*, or extract, transform, load. Extract pulls the data from the source system. Transform optionally changes the structure of the data, performs data quality cleaning, or aggregates the data. Load puts the data into the database. This process can also be called *ELT*, for extract, load, transform—the difference being that, rather than transformations being done before data is loaded, all the data is loaded and then transformations are performed, usually using SQL. You might also hear the terms *source* and *target* in the context of ETL. The source is where the data comes from, and the target is the destination, i.e., the database and the tables within it. Even when SQL is used to do the transforming, another language such as Python or Java is used to glue the steps together, coordinate scheduling, and raise alerts when something goes wrong. There are a number of commercial products as well as open source tools available, so teams don't have to create an ETL system entirely from scratch.

Once the data is in a database, the next step is performing queries and analysis. In this step, SQL is applied to explore, profile, clean, shape, and analyze the data. **Figure 1-3** shows the general flow of the process. Exploring the data involves becoming familiar with the topic, where the data was generated, and the database tables in which it is stored. Profiling involves checking the unique values and distribution of records in the data set. Cleaning involves fixing incorrect or incomplete data, adding categorization and flags, and handling null values. Shaping is the process of arranging the data into the rows and columns needed in the result set. Finally, analyzing the data involves reviewing the output for trends, conclusions, and insights. Although this process is shown as linear, in practice it is often cyclical—for example, when shaping or analysis reveals data that should be cleaned.

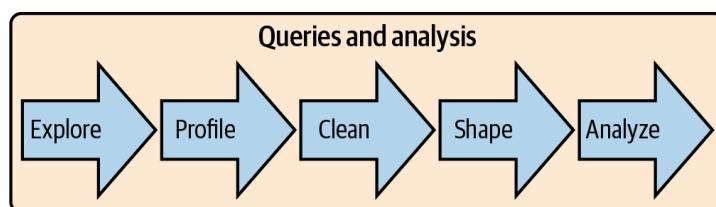


Figure 1-3. Stages within the queries and analysis step of the analysis workflow

Presentation of the data into a final output form is the last step in the overall workflow. Businesspeople won't appreciate receiving a file of SQL code; they expect you to present graphs, charts, and insights. Communication is key to having an impact with analysis, and for that we need a way to share the results with other people. At other times, you may need to apply more sophisticated statistical analysis than is possible in SQL, or you may want to feed the data into a machine learning (ML) algorithm. Fortunately, most reporting and visualization tools have SQL connectors that allow you to pull in data from entire tables or prewritten SQL queries. Statistical software and languages commonly used for ML also usually have SQL connectors.

Analysis workflows encompass a number of steps and often include multiple tools and technologies. SQL queries and analysis are at the heart of many analyses and are what we will focus on in the following chapters. [Chapter 2](#) will discuss types of source systems and the types of data they generate. The rest of this chapter will take a look at the types of databases you are likely to encounter in your analysis journey.

Database Types and How to Work with Them

If you're working with SQL, you'll be working with databases. There is a range of database types—open source to proprietary, row-store to column-store. There are on-premises databases and cloud databases, as well as hybrid databases, where an organization runs the database software on a cloud vendor's infrastructure. There are also a number of data stores that aren't databases at all but can be queried with SQL.

Databases are not all created equal; each database type has its strengths and weaknesses when it comes to analysis work. Unlike tools used in other parts of the analysis workflow, you may not have much say in which database technology is used in your organization. Knowing the ins and outs of the database you have will help you work more efficiently and take advantage of any special SQL functions it offers. Familiarity with other types of databases will help you if you find yourself working on a project to build or migrate to a new data warehouse. You may want to install a database on your laptop for personal, small-scale projects, or get an instance of a cloud warehouse for similar reasons.

Databases and data stores have been a dynamic area of technology development since they were introduced. A few trends since the turn of the 21st century have driven the technology in ways that are really exciting for data practitioners today. First, data volumes have increased incredibly with the internet, mobile devices, and the Internet of Things (IoT). In 2020 [IDC predicted](#) that the amount of data stored globally will grow to 175 zettabytes by 2025. This scale of data is hard to even think about, and not all of it will be stored in databases for analysis. It's not uncommon for companies to have data in the scale of terabytes and petabytes these days, a scale that would have been impossible to process with the technology of the 1990s and earlier. Second, decreases in data storage and computing costs, along with the advent of the cloud,

have made it cheaper and easier for organizations to collect and store these massive amounts of data. Computer memory has gotten cheaper, meaning that large amounts of data can be loaded into memory, calculations performed, and results returned, all without reading and writing to disk, greatly increasing the speed. Third, distributed computing has allowed the breaking up of workloads across many machines. This allows a large and tunable amount of computing to be pointed to complex data tasks.

Databases and data stores have combined these technological trends in a number of different ways in order to optimize for particular types of tasks. There are two broad categories of databases that are relevant for analysis work: row-store and column-store. In the next section I'll introduce them, discuss what makes them similar to and different from each other, and talk about what all of this means as far as doing analysis with data stored in them. Finally, I'll introduce some additional types of data infrastructure beyond databases that you may encounter.

Row-Store Databases

Row-store databases—also called *transactional* databases—are designed to be efficient at processing transactions: *INSERTs*, *UPDATEs*, and *DELETEs*. Popular open source row-store databases include MySQL and Postgres. On the commercial side, Microsoft SQL Server, Oracle, and Teradata are widely used. Although they're not really optimized for analysis, for a number of years row-store databases were the only option for companies building data warehouses. Through careful tuning and schema design, these databases can be used for analytics. They are also attractive due to the low cost of open source options and because they're familiar to the database administrators who maintain them. Many organizations replicate their production database in the same technology as a first step toward building out data infrastructure. For all of these reasons, data analysts and data scientists are likely to work with data in a row-store database at some point in their career.

We think of a table as rows and columns, but data has to be serialized for storage. A query searches a hard disk for the needed data. Hard disks are organized in a series of blocks of a fixed size. Scanning the hard disk takes both time and resources, so minimizing the amount of the disk that needs to be scanned to return query results is important. Row-store databases approach this problem by serializing data in a row. [Figure 1-4](#) shows an example of row-wise data storage. When querying, the whole row is read into memory. This approach is fast when making row-wise updates, but it's slower when making calculations across many rows if only a few columns are needed.