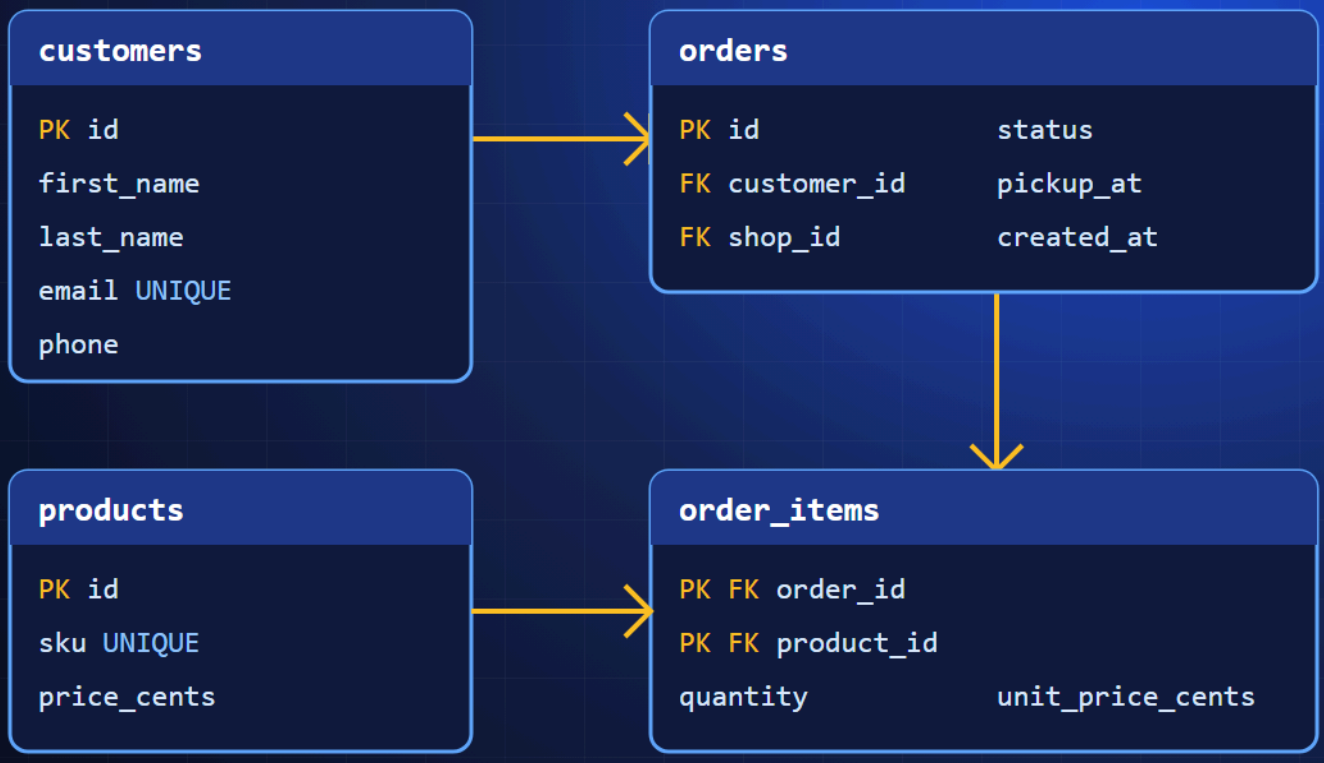




1NF · 2NF · 3NF · keys · joins · indexes

A PRACTICAL GUIDE

Database Design for Beginners

Model, normalize, and build relational databases — no experience needed

ER diagrams · Keys and relationships · Normalization
Constraints · SQL joins · Indexes · PostgreSQL

MODELING · NORMALIZATION · INTEGRITY · SQL

THENEXBOOKS

Database Design for Beginners

A Practical Guide to Modeling, Normalizing, and Building Relational Databases
With PostgreSQL examples throughout

First Edition

THENEXBOOKS

Database Design for Beginners
A Practical Guide to Modeling, Normalizing, and Building Relational Databases

First Edition, 2026

Copyright © 2026 thenexbooks (thenexbooks.gumroad.com). All rights reserved.

No part of this publication may be reproduced, distributed, or transmitted in any form or by any means without the prior written permission of the publisher, except for brief quotations in reviews and certain other non-commercial uses permitted by copyright law. SQL examples may be used freely in your own projects.

The information in this book is provided "as is" without warranty of any kind. While every precaution has been taken in its preparation, the publisher assumes no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein. Database software evolves; always consult the official documentation for the versions you use.

PostgreSQL is a trademark of the PostgreSQL Community Association of Canada. MySQL is a trademark of Oracle Corporation and/or its affiliates. MariaDB is a trademark of MariaDB plc. Microsoft SQL Server is a trademark of Microsoft Corporation. Other product names are trademarks of their respective owners. Their use in this book is for identification only and does not imply endorsement. This book is not affiliated with, sponsored by, or endorsed by any of these companies or projects.

Ovenly, the bakery used as the running example in this book, is fictional, as are all shops, customers, staff, and suppliers mentioned in its examples. Any resemblance to real businesses or people is coincidental.

Preface

Almost every app, website, and business system you have ever used keeps its information in a database. Your online orders, your bank balance, the songs in your playlist, the appointments at your doctor's office — all of it lives in tables that somebody designed. When that design is good, nobody notices: the data is correct, reports add up, and new features are easy to build. When it is poor, everyone feels it: duplicate customers, totals that disagree, "impossible" records, and changes that take weeks.

The good news is that database design is a learnable skill, not a mysterious talent. A small set of ideas — tables and keys, relationships, normalization, and constraints — explains most of what makes a design good or bad. You do not need a computer science degree to understand them. You need clear explanations, realistic examples, and a chance to watch a design grow step by step.

That is what this book offers. We start from zero, assuming no prior knowledge of databases or SQL. We take a small business that is outgrowing its spreadsheet and design a proper database for it, one decision at a time, explaining *why* each choice is made. By the end, you will be able to design a relational database on your own, recognize common mistakes, and talk confidently with developers and data teams.

Who This Book Is For

- **Complete beginners** who want to understand how databases work and how to design one well.
- **Students** taking their first database course who want practical examples alongside the theory.
- **Junior developers** who can write code but have never designed a schema from scratch.
- **Analysts, founders, and makers** whose spreadsheets have grown too big and who need something more reliable.

What You Will Learn

- What a relational database is, how tables, rows, columns, and keys work, and how to set up PostgreSQL.
- Just enough SQL to create tables, add data, and read it back.
- How to turn real-world requirements into entities, attributes, and relationships, and draw them as ER diagrams.
- One-to-many, many-to-many, one-to-one, and self-referencing relationships, with foreign keys.
- Normalization from first to third normal form and beyond — and when deliberate denormalization makes sense.
- Constraints, referential integrity, NULL handling, and everyday patterns for money, dates, status, and history.
- Joins, reporting queries, indexes, views, transactions, and the basics of performance.

- Changing a schema safely, basic database security, common anti-patterns, and a design review checklist.

How This Book Is Organized

Part I — Database Foundations (Chapters 1–5) explains what databases are, gets PostgreSQL running, and covers tables, data types, basic SQL, and keys. **Part II — Modeling Data** (Chapters 6–10) turns requirements into a data model and covers every kind of relationship. **Part III — Normalization** (Chapters 11–15) shows how to remove redundancy step by step and when to add it back on purpose. **Part IV — Integrity and Patterns** (Chapters 16–19) makes the database protect its own data and covers patterns you will use in almost every project. **Part V — Querying and Performance** (Chapters 20–24) reads the design back with joins and reports and keeps it fast. **Part VI — Real-World Design** (Chapters 25–28) covers changing a live schema, security, design reviews, and a complete case study. The appendices provide a cheat sheet, templates and a data type reference, a review checklist, and a glossary.

Each chapter ends with a **Key Takeaways** box for quick review, and many include short exercises so you can practice.

The Running Example: Ovenly

Throughout the book we design the database for **Ovenly**, a fictional neighborhood bakery with three shops. Ovenly bakes bread, pastries, and cakes, takes online pre-orders for pickup, runs a loyalty program, buys ingredients from several suppliers, and schedules staff shifts. Until now, everything has lived in a spreadsheet — and the spreadsheet is starting to hurt: customers appear twice with different phone numbers, a price change makes old order totals wrong, and nobody is sure which shop an order belongs to. Over the book we replace that spreadsheet with a clean, well-designed PostgreSQL database, and Chapter 28 presents the complete design.

Conventions

Examples use PostgreSQL, a free and widely used open-source database. The design ideas apply to every relational database; where MySQL, SQLite, or SQL Server do something noticeably differently, a Note points it out.

> **TIP**
> A recommended practice or a technique that makes your work easier.

> **NOTE**
> Background information, differences between database systems, or a useful aside.

> **WARNING**
>
> A common mistake or something that can lose or corrupt data.

SQL appears in `monospace`. Table and column names use lowercase words joined by underscores, such as `order_items` and `price_cents`. Money is stored as whole numbers of cents, never as floating-point numbers.

Contents

Preface

PART I — DATABASE FOUNDATIONS

1 What Is a Database? Data, Tables, and Why Design Matters

2 Setting Up: PostgreSQL, a Database Client, and Your First Database

3 Tables, Rows, Columns, and Choosing Data Types

4 SQL Basics for Designers: CREATE, INSERT, SELECT, UPDATE, and DELETE

5 Keys: Primary Keys, Natural and Surrogate Keys, and Uniqueness

PART II — MODELING DATA

6 From Requirements to Data: Stories, Nouns, and Business Rules

7 Entity-Relationship Modeling: Entities, Attributes, and Relationships

8 One-to-Many Relationships and Foreign Keys

9 Many-to-Many Relationships and Junction Tables

10 One-to-One, Optional Relationships, and Self-References

PART III — NORMALIZATION

11 Why Normalize? Redundancy and Update Anomalies

12 First Normal Form: Atomic Values and No Repeating Groups

13 Second and Third Normal Form

14 Beyond Third Normal Form: BCNF and Knowing When to Stop

15 Denormalization: Deliberate Trade-offs

PART IV — INTEGRITY AND PATTERNS

16 Constraints: NOT NULL, UNIQUE, CHECK, and DEFAULT

17 Referential Integrity: ON DELETE, ON UPDATE, and Cascades

18 NULL, Missing Data, and Optional Values

19 Everyday Modeling Patterns: Lookup Tables, Status, Money, Dates, Addresses, and History

PART V — QUERYING AND PERFORMANCE

20 Joins: Reading Your Design Back

21 Aggregation and Reporting Queries

22 Indexes Explained

23 Views, Transactions, and Keeping Data Consistent

24 Designing for Performance: Query Plans and Common Mistakes

PART VI — REAL-WORLD DESIGN

25 Changing a Schema Safely: Migrations and Versioning

26 Security Basics: Users, Roles, Permissions, and Sensitive Data

27 Design Review: Anti-Patterns and a Checklist

28 Case Study: Designing the Ovenly Database from Scratch

APPENDICES

A SQL and Design Cheat Sheet

B Templates and Data Type Reference

C Design Review Checklist

D Glossary

PART I

Database Foundations

What a database is and why its design matters, a working PostgreSQL setup, tables and data types, just enough SQL to build and query them, and the keys that give every row its identity.

1 · What Is a Database?
2 · Setting Up
3 · Tables, Columns, and Data Types
4 · SQL Basics for Designers
5 · Keys and Uniqueness

CHAPTER 1 · PART I — DATABASE FOUNDATIONS

What Is a Database? Data, Tables, and Why Design Matters

Before we write a single line of SQL, we need a clear picture of what a database is, how it differs from a spreadsheet, and why a little design work up front saves an enormous amount of pain later. In this chapter we meet Ovenly, the small bakery whose data we will organize throughout the book.

Meet Ovenly

Ovenly is a fictional neighborhood bakery. It started as a single shop and has grown to three: **Ovenly Riverside**, **Ovenly Old Town**, and **Ovenly Station**. It bakes sourdough loaves, butter croissants, chocolate cakes, cookies, and seasonal treats. Customers can walk in and buy something off the shelf, or they can place a pre-order online and collect it later. Regular customers can join a loyalty program and earn points. Behind the counter, bakers, cashiers, managers, and delivery drivers work in shifts, and the bakery buys flour, butter, chocolate, and other ingredients from a handful of suppliers.

Every one of those activities produces information. Who ordered what? Which shop will they collect it from, and when? How much did they pay? Which products contain nuts? Who is working at Riverside on Saturday morning? A bakery this size cannot keep all of that in anyone's head. It needs somewhere to store the information reliably, and it needs to be able to ask questions of that information quickly.

That "somewhere" is a database, and this book is about designing one well. We will use Ovenly as our running example from the first chapter to the last. By the end, in Chapter 28, you will have seen the complete Ovenly database built from scratch, and you will understand every decision that went into it.

Data and Information

Let us start with two words that people often use interchangeably. **Data** means raw facts: a name, a number, a date, a yes-or-no answer. "Maya Chen", "650", "2026-03-14", and "true" are all pieces of data. On their own they do not say much.

Information is data that has been given context so that it means something. "Maya Chen ordered one Sourdough Loaf priced at 650 cents on March 14, 2026, and has already paid" is information. It connects several pieces of data together so a person can act on it.

A database stores data in a structured way so that we can turn it into information whenever we need it. If the data is stored carelessly, that becomes slow, error-prone, or impossible. That is the central idea of this whole book.

So What Is a Database?

A **database** is an organized collection of data, stored so that it can be easily searched, updated, and kept correct over time. That definition is broad on purpose. A paper card index in a library is, in a loose sense, a database. So is the contacts list on your phone. In software, though, when people say "database" they usually mean data managed by a special program designed for the job.

That program is called a **database management system**, or **DBMS** for short. The DBMS is the software that actually stores the data on disk, lets many people and applications use it at the same time, enforces the rules you define, and answers your questions. Some well-known DBMS products are PostgreSQL, MySQL, MariaDB, SQLite, Microsoft SQL Server, and Oracle Database. In this book we use **PostgreSQL** (often shortened to "Postgres"), which is free, open source, widely used, and follows the SQL standard closely.

People are sometimes loose with these terms, and you will hear "the database" used to mean the DBMS software, the server machine it runs on, or the actual collection of data. In this book we try to be precise:

- **Database**: a named collection of related data, such as the `ovenly` database.
- **DBMS**: the software that manages databases, such as PostgreSQL.
- **Database server**: a running copy of the DBMS, usually on a computer that other programs connect to. One server can hold many databases.

> **NOTE**
>
> You may also see the term **RDBMS**, which stands for *relational* database management system. PostgreSQL, MySQL, SQL Server, SQLite, and Oracle are all relational. We explain what "relational" means later in this chapter.

Spreadsheets: Where Most Data Starts

Almost every small business starts by tracking its data in a spreadsheet, and Ovenly was no exception. Spreadsheets are wonderful tools. They are easy to open, easy to share, and anyone can type into them. You see all your data at once in a grid, and you can add a new column whenever you think of something new to record.

When Ovenly opened its first shop, the owner created a spreadsheet called `orders_sheet` to track pre-orders. Here is what a few rows looked like. We will refer back to this sheet many times, so take a moment to read it. **This is the deliberately bad version**, the design we will spend much of the book improving.

customer_name	customer_email	items	shop	total	phone1
Maya Chen	maya@example.com	2x Sourdough, 1x Croissant	Riverside	16.25	555-0101

Tomás Rivera	tomas@example.com	1x Choc Cake 20cm	Old Town	32	555-0102
maya chen	maya.chen@example.com	3 croissants	riverside	9.75	
Priya Nair	priya@example.com	1x Sourdough	Station	$6.50	555-0103

The real sheet also had a `phone2` column for customers who gave a second number, but we have left it out so the table fits on the page. Look closely and you will already spot trouble:

- Maya Chen appears twice, once with a capital "M" and once in lowercase, and with two different email addresses. Is that one customer or two?
- The `items` column crams several products and quantities into a single piece of text. The same croissant is written as "Croissant" in one row and "croissants" in another. Asking "how many croissants did we sell this week?" means reading and interpreting every cell by hand.
- The `total` column mixes plain numbers (`16.25`, `32`) with text that includes a dollar sign (`$6.50`). A formula that adds up the column may silently skip the text value.
- The shop name is typed freely, so "Riverside" and "riverside" might be counted as different shops in a report.
- Nothing stops someone from typing a total that does not match the items, or leaving the email blank.

None of these problems is anyone's fault. They are the natural result of a tool that lets you type anything anywhere. With three shops and thousands of orders, they become expensive mistakes: the wrong cake baked, or sales reports that nobody trusts.

Spreadsheets Versus Databases

A database is not simply "a bigger spreadsheet." It is built on different assumptions. The table below compares the two side by side. Neither is better at everything; they are designed for different jobs.

Question	Spreadsheet	Relational database
Who uses it?	Usually one person or a small team, often by hand	Many people and applications at the same time
What can go in a cell?	Anything anyone types	Only values that match the column's declared type and rules
How are related things linked?	By copying text, or by fragile lookup formulas	By keys that the DBMS checks for you
How do you ask questions?	Filters, formulas, pivot tables	Queries written in SQL
What happens with lots of data?	Tends to get slow and hard to manage	Designed to handle large volumes efficiently

What if two people edit at once?	Conflicts or overwritten changes are common	Handles concurrent changes safely

The most important difference is the second row. A database lets you declare **rules** about your data, and then it enforces those rules every single time, for every person and every program. If you say that every order must belong to a real shop, the database will refuse an order for a shop that does not exist. If you say that a price cannot be negative, a negative price will be rejected. A spreadsheet generally trusts whatever is typed; a database checks.

> **TIP**
>
> Spreadsheets are still great for quick analysis and one-off lists. A good rule of thumb: when several people or programs depend on the same data, when mistakes would be costly, or when the data keeps growing, it is time for a database. Many teams keep using spreadsheets for analysis by exporting data *from* the database.

The Relational Model in Plain Words

In 1970, a computer scientist named E. F. Codd, working at IBM, published a paper describing the **relational model** of data. His idea was to store all data in simple two-dimensional tables and to describe questions about that data mathematically, instead of forcing programmers to follow pointers from record to record. Almost every mainstream database you will use today, including PostgreSQL, is built on his ideas. You do not need to know the mathematics to design good databases, but a few words are worth learning now.

Tables

A **table** holds data about one kind of thing. Ovenly will have a `customers` table, a `products` table, a `shops` table, an `orders` table, and so on. Each table has a name and a fixed list of columns. In formal writing, a table is called a *relation*, which is where the word "relational" comes from. (It does not come from "relationships," although relational databases are very good at relationships too.)

Columns

A **column** describes one property that every item in the table has. The `products` table might have columns called `sku`, `name`, and `price_cents`. Each column has a **data type**, which says what kind of value it holds: whole numbers, text, dates, true/false values, and so on. Chapter 3 covers data types in detail. Columns are sometimes called *fields* or, in formal writing, *attributes*.

Rows

A **row** is one item in the table: one customer, one product, one order. Every row in a table has the same columns, and each column holds one value for that row. Rows are sometimes called *records* or, formally, *tuples*. Here are three rows from Ovenly's `products` table:

id	sku	name	price_cents	is_active
1	BRD-SOUR-800	Sourdough Loaf	650	true
2	PST-CROI-01	Butter Croissant	325	true
3	CAK-CHOC-20	Chocolate Cake (20 cm)	3200	true

Notice a few things. The price is stored as a whole number of cents (650 cents is $6.50), which avoids rounding problems we will discuss in Chapter 3. The names are consistent. Each product has an `id`, a number that identifies that one row and no other. And each cell holds exactly one value; there is no cell that says "650 or 700 on weekends."

Keys

A **key** is a column, or a set of columns, used to identify rows. The most important one is the **primary key**: a column whose value is unique for every row and never empty. In the products table above, `id` is the primary key. If someone says "product 2," there is exactly one row they could mean: the Butter Croissant. Chapter 5 is entirely about keys, because choosing them well is one of the most important design decisions you will make.

Relationships

Real-world things are connected. A customer places orders. An order is collected at a shop. An order contains products. In a relational database we express these connections by storing the key of one row inside another row. A column that holds another table's primary key is called a **foreign key**.

For example, instead of typing "Riverside" into each order, an order row stores `shop_id = 1`, which points to the row with `id = 1` in the `shops` table. Here are the three Ovenly shops:

id	name	city	opened_on
1	Ovenly Riverside	Riverton	2019-04-06
2	Ovenly Old Town	Riverton	2021-09-18
3	Ovenly Station	Riverton	2024-02-10

And here are a few rows from the `orders` table (showing only some of its columns):

id	customer_id	shop_id	status	pickup_at
1001	1	1	collected	2026-03-14 09:30
1002	2	2	confirmed	2026-03-15 16:00
1003	3	3	pending	2026-03-16 08:15

Order 1001 has `shop_id` 1, so it will be collected at Ovenly Riverside. Order 1002 has `shop_id` 2, so it belongs to Ovenly Old Town. The shop's name is stored in exactly one place. If Ovenly ever renames a shop, we change one row in `shops`, and every order automatically "sees" the new name, because orders only store the number. Compare that with the spreadsheet, where a rename would mean hunting down and editing every row that mentions the old name, and hoping nobody missed one.

The database also checks that `shop_id` values really exist. If someone tries to create an order with `shop_id = 7`, PostgreSQL will refuse, because there is no shop 7. This check is called **referential integrity**, and Chapters 8 and 17 explore it in depth.

> **NOTE**
>
> The `pickup_at` values in the table above are shown in a short, friendly format. In the real database this column stores a full date, time, and time zone. We will see exactly how PostgreSQL displays such values in Chapter 3.

A Quick Look at SQL

To talk to a relational database, we use **SQL**, which stands for *Structured Query Language*. People pronounce it either as the letters "S-Q-L" or as the word "sequel"; both are fine. SQL is a language for describing what you want, not step-by-step instructions for how to get it. You say "give me the names and prices of all active products, cheapest first," and the DBMS works out the fastest way to find them. SQL has been standardized for decades, so the basics work the same way in PostgreSQL, MySQL, SQL Server, SQLite, and others, although each product adds its own extensions and small differences.

Here is a small taste. Do not worry about the details yet; Chapter 4 teaches each piece properly.

ovenly/first-look.sql

```
-- List active products from cheapest to most expensive
SELECT name, price_cents
FROM products
WHERE is_active = true
ORDER BY price_cents;
```

Given the three products shown earlier, PostgreSQL would answer:

```
          name          | price_cents
------------------------+-------------
 Butter Croissant       |         325
 Sourdough Loaf         |         650
 Chocolate Cake (20 cm) |        3200
(3 rows)
```

Even if you have never seen SQL before, you can probably guess what that query does. That readability is one of SQL's great strengths. SQL is also used to *define* the structure of the database: to create tables, declare columns and their types, and set up rules. The part of SQL that defines structure is often called **DDL** (data definition language), and the part that reads and changes data is called **DML** (data manipulation language). You will see both throughout this book.

Why Design Matters

You might be wondering: if the database checks rules for us, why do we need a whole book about design? The answer is that the database can only enforce the rules you give it, using the structure you give it. A badly designed database will faithfully store bad data forever. Design is where you decide which tables exist, which columns they have, what types those columns use, how the tables connect, and which rules apply. Get those decisions right, and the database quietly protects you. Get them wrong, and you fight the database every day.

Let us return to Ovenly's spreadsheet and list the specific problems that good design will solve. We will name them now and fix each one properly in later chapters.

Problem 1: The same fact is stored many times

Maya's email address is typed into every order she places. If she changes her email, somebody has to find and update every one of those rows. Miss one, and the data now disagrees with itself. Storing the same fact in several places is called **redundancy**, and the mistakes it causes are called **anomalies**. Chapter 11 shows these anomalies in detail, and Chapters 12 through 14 introduce **normalization**, a set of step-by-step techniques for removing harmful redundancy.

Problem 2: Several values crammed into one cell

The `items` column holds text like "2x Sourdough, 1x Croissant." A computer cannot reliably count, sum, or search inside that text, because it is just a sentence that a human wrote. A good design stores each product in an order as its own row, with separate columns for the product and the quantity. This is the idea of **atomic values**, covered in Chapter 12.

Problem 3: Repeating columns

The `phone1` and `phone2` columns are a guess about how many phone numbers a customer might have. What about a customer with three? What about a report that must search both columns every time? Columns that repeat the same kind of thing are called a **repeating group**, another problem Chapter 12 solves.

Problem 4: Inconsistent values

"Riverside" versus "riverside," "32" versus "$6.50." Without rules, people type what they like. In a well-designed database, the shop is a reference to a row in the `shops` table, the total is a number of a specific type, and constraints (rules attached to columns and tables) reject anything that does not fit.

Chapter 16 covers constraints.

Problem 5: Questions the design cannot answer

Suppose the owner asks: "Which products contain nuts, and how many of them did we sell last month at each shop?" The spreadsheet has no ingredient information at all, and the sales are locked inside free text. A good design anticipates the questions the business will ask and makes sure the data needed to answer them is stored in a usable form. Chapter 6 is about discovering those questions before you build anything.

> **WARNING**
>
> It is tempting to skip design and "just start building," then fix things later. Changing the structure of a database that already holds real data is much harder than changing a plan on paper, because every existing row has to be moved or converted and every program that uses the database may need to change too. A few hours of design thinking up front can save weeks of painful fixes later. Chapter 25 shows how to change a live schema safely, but the best change is the one you never have to make.

What a Good Design Looks Like

We will spend the whole book refining what "good" means, but here is a preview of the qualities we aim for. A well-designed database:

1. **Stores each fact once.** A customer's email lives in one row of one table.
2. **Uses clear, consistent names.** Anyone reading `price_cents` knows it is a price, in cents.
3. **Uses the right data type for every column.** Dates are dates, money is an exact number, true/false values are booleans.
4. **Protects itself with rules.** Prices cannot be negative; every order belongs to a real shop.
5. **Makes relationships explicit.** Orders point to customers and shops through keys the database checks.
6. **Answers the business's real questions** with straightforward queries.
7. **Can grow and change** without constant rework.

If we redesign Ovenly's spreadsheet with those qualities in mind, the single messy grid becomes several focused tables. Here is a small preview of how Maya's first order looks once it is split up properly. First, the `customers` table (some columns omitted):

id	first_name	last_name	email	phone
1	Maya	Chen	maya@example.com	555-0101
2	Tomás	Rivera	tomas@example.com	555-0102
3	Priya	Nair	priya@example.com	555-0103

Then the `order_items` table, which records each product in an order on its own row:

order_id	product_id	quantity	unit_price_cents
1001	1	2	650
1001	2	1	325
1002	3	1	3200
1003	1	1	650

Order 1001 now reads clearly: two of product 1 (Sourdough Loaf) at 650 cents each and one of product 2 (Butter Croissant) at 325 cents. The total, 2 × 650 + 325 = 1625 cents, or $16.25, matches the spreadsheet, but now the database can calculate it rather than trusting a hand-typed number. Counting croissants sold is a simple query. Maya exists exactly once, with one email address. The shop name is stored once in `shops`.

You may notice that `order_items` stores `unit_price_cents` even though the product table already has a price. That is deliberate: prices change over time, and an order must remember the price the customer actually paid. Chapter 15 explains why this is correct history rather than harmful redundancy. Good design is full of small decisions like this, and we will explain the reasoning behind each one.

Splitting data across several tables might look like more work, but a query can **join** the tables back together whenever we want to see everything at once. Chapter 8 introduces joins and Chapter 20 covers them in full.

Other Kinds of Databases

You may have heard of databases that are not relational, often grouped under the label **NoSQL**: document databases, key-value stores, graph databases, and others. Each has real strengths for particular problems. This book focuses on relational databases because they are an excellent general-purpose default for business data like Ovenly's, and because the design skills they teach (thinking clearly about entities, relationships, and rules) transfer to every other kind of database.

How This Book Is Organized

The book is divided into six parts, and each part builds on the one before. You can read it straight through, which we recommend for complete beginners, or dip into a specific chapter later when you need it.

Part	Chapters	What you will learn
I — Database Foundations	1–5	What databases are, installing PostgreSQL, tables and data types, basic SQL, and keys

II — Modeling Data	6–10	Turning requirements into entities and relationships, drawing diagrams, and building one-to-many, many-to-many, and one-to-one links
III — Normalization	11–15	Spotting redundancy, the normal forms, and when to deliberately bend the rules
IV — Integrity and Patterns	16–19	Constraints, foreign key actions, NULL, and everyday patterns for money, dates, status, and history
V — Querying and Performance	20–24	Joins, reporting queries, indexes, views, transactions, and performance basics
VI — Real-World Design	25–28	Changing schemas safely, security, reviewing designs, and the complete Ovenly case study

The appendices at the back include a SQL and design cheat sheet, templates and a data type reference, a design review checklist, and a glossary. Whenever you meet a term you have forgotten, the glossary is a good place to look it up.

How to get the most from this book

- **Type the examples yourself.** Reading SQL is useful; typing it, running it, and seeing the result is far more memorable. Chapter 2 gets PostgreSQL running on your computer.
- **Make mistakes on purpose.** Try inserting a negative price or an order for a shop that does not exist. Seeing the database refuse is the best way to understand why rules matter.
- **Do the "Try It Yourself" exercises.** Most chapters end with a few small tasks. They are short, and they turn reading into skill.

> **TIP**
>
> Keep a notebook, paper or digital, for design sketches. Many design mistakes are easiest to catch when you draw your tables as boxes and your relationships as lines, before you write any SQL. Chapter 7 teaches a standard way to draw these diagrams.

A Word on Vocabulary

Database people use several words for the same ideas, depending on whether they are speaking formally, casually, or about a particular product. The table below collects the ones you have met so far, so you can recognize them in other books, documentation, and conversations.

Everyday word	Also called	Meaning
Table	Relation, entity (in diagrams)	A collection of rows describing one kind of thing
Row	Record, tuple	One item in a table
Column	Field, attribute	One property that every row has
Schema	Structure, design	The set of tables, columns, types, and rules

Query	Statement (more generally)	A request to read or change data

We have used one new word in that table: **schema**. A schema is the overall structure of a database: which tables exist, what columns each has, what types those columns use, and what rules apply. The schema is the design; the rows are the data that fills it. (PostgreSQL also uses "schema" for a named folder of tables inside a database. We will point out that second meaning when it comes up in Chapter 2.)

Try It Yourself

1. Look again at Ovenly's `orders_sheet`. Find at least three different ways that a report of "total sales per shop" could come out wrong.
2. Think of a spreadsheet you have used, at work, school, or home. Write down which columns in it contain more than one value in a single cell, or repeat the same kind of information (like `phone1` and `phone2`).
3. List five kinds of "things" Ovenly needs to keep track of (for example, customers). For each one, write three properties you would want to record about it.
4. The order in which Maya bought two sourdough loaves and a croissant is order 1001. Using the `order_items` and `products` sample rows in this chapter, calculate the total in cents, and then check your answer against the spreadsheet.