DataShelf Hub

DATABASES & DATA SYSTEMS

Scaling Relational Databases

Replication, partitioning and sharding without downtime

FREE SAMPLE CHAPTER

Omar El Alaoui
Data & systems engineer

PDF · CH. 01

ABOUT THIS SAMPLE

A free first chapter

What follows is Chapter 1 of *Scaling Relational Databases*, reproduced in full and provided at no cost by DataShelf Hub. It is unabridged — nothing has been trimmed or summarised. If it's useful to you, the remaining seven chapters continue exactly where this one leaves off.

This sample is for your personal use while you decide whether the book is right for you. Please don't republish or redistribute it — share a link to the book's page instead.

TITLE	Scaling Relational Databases
AUTHOR	Omar El Alaoui
SHELF	Databases & Data Systems
FORMAT	PDF, DRM-free, 292 pages
THIS SAMPLE	Chapter 1 of 8 — 15 pages
PUBLISHER	DataShelf Hub · datashelfhub.com

WHAT'S IN THE FULL BOOK

Table of contents

8 chapters · 24 sections

Continued on the next page →

TABLE OF CONTENTS, CONTINUED

Chapters 5–8

CHAPTER ONE

01

Scaling the Database You Have

Before you shard anything, the database you already have has more capacity left in it than most teams assume — and finding it is cheaper than any of the alternatives.

Somewhere around the second time a query times out under load, a familiar idea starts circulating: maybe it's time to shard. It's an understandable instinct — sharding sounds like the grown-up, web-scale answer, the thing companies do once they're serious. It's also, for the overwhelming majority of teams reaching for it, the wrong first move, undertaken months before it's actually necessary and at a cost that dwarfs whatever it was meant to solve.

This chapter argues for a less exciting sequence: exhaust what a single well-tuned database can do, understand precisely where relational databases actually stop scaling and why, and only then pick a strategy — usually something much simpler than sharding — matched to the specific limit you've actually hit. Most teams never reach the point where sharding is the right answer. This book still covers it, in Chapter 5, because some genuinely do. But it's the last chapter for a reason.

1.1 Vertical Headroom First

"Scaling" gets used almost exclusively to mean scaling out — adding more machines. It's worth remembering that scaling up still exists, and modern hardware has made it a far more generous option than the conventional wisdom admits. Cloud providers now offer database instances with terabytes of RAM and dozens of cores. A dataset that felt unmanageably large five years ago often fits comfortably in memory on a single modern instance today.

The appeal isn't just cost, though a bigger single instance is almost always cheaper in engineering time than any distributed alternative. It's that vertical scaling changes nothing about how your application talks to the database. No sharding logic, no routing layer, no new failure modes around cross-node consistency — the same queries, the same transactions, the same operational model you already understand, just with more headroom underneath them.

> *"The cheapest scaling win is the one that requires zero application changes. Vertical headroom is that win, and most teams abandon it before they've actually used it up."*

None of this means vertical scaling is infinite — it isn't, and this book doesn't pretend otherwise. It has a ceiling, set by the largest instance your provider offers and by the point where a single machine simply can't hold your working set or serve your write volume. The point is narrower: that ceiling is much higher than most teams test before reaching for something more complex, and the cost of finding out where it actually is is close to zero.

Before assuming you've hit that ceiling, it's worth ruling out the more common culprit: that the database isn't undersized, it's under-tuned. Missing indexes, connection pool exhaustion, and queries doing far more work than they need to are responsible for more "we need to scale" conversations than genuine capacity limits — and all three are cheaper to fix than any scaling strategy in this book.

BEFORE REACHING FOR A SCALING STRATEGY, RULE THESE OUT

Missing or unused indexes	the single most common cause of "the database can't keep up"
Connection exhaustion	covered in depth in Chapter 2 — often mistaken for a capacity problem
N+1 queries and unbounded scans	application-level waste that no amount of hardware fixes

None of this is an argument against ever scaling out. It's an argument for sequencing: confirm the database itself is well-tuned, confirm you've used the vertical headroom available to you, and only then treat scaling out as the answer to a specific, measured limit — not a reflex pulled from what larger companies did once they had genuinely outgrown a single machine, at a scale most systems never reach.

That sequencing matters because every strategy past this chapter — read replicas, partitioning, sharding — adds real operational complexity: new failure modes, new consistency questions, new things that can be subtly wrong in production. None of them are free, and all of them are easier to add later, deliberately, than to unwind after adopting them prematurely.

1.2 Where Relational Actually Breaks

When a single instance genuinely does run out of road, it breaks along a small number of specific, recognisable dimensions — not as a vague "it's slow now," but as one of a few distinct bottlenecks, each with a different fix. Naming which one you've hit is the first real diagnostic step, and it's usually possible to tell from your metrics alone.

Write throughput is the most common ceiling: a single primary can only accept so many writes per second before WAL flush, replication, and lock contention start queuing behind each other. Working-set size is the second: once your actively-queried data exceeds available memory, every query that misses cache pays real disk I/O, and performance falls off a cliff rather than degrading gracefully. Connection count is the third, and it's covered fully in Chapter 2, because it's frequently mistaken for the other two.

THREE DISTINCT CEILINGS, THREE DISTINCT FIXES

Write throughput	bottleneck on the primary — sharding or write batching, not read replicas
Working-set size	exceeds memory — bigger instance, or partition cold data out of the hot path
Read volume	the one case read replicas genuinely solve — covered in Chapter 3

The reason this distinction matters so much is that the wrong fix for the wrong ceiling doesn't just fail to help — it adds real complexity for no benefit. Read replicas do nothing for a write-throughput problem; every write still has to go through the same primary, replicas or not. Sharding does nothing for a working-set problem if your access pattern is skewed enough that most traffic still lands on one shard.

That's the pattern worth internalising before the rest of this book gets into specific techniques: a scaling strategy is a targeted answer to a specific, measured bottleneck, not a general-purpose upgrade you apply because growth demands it. Chapters 3 through 5 map directly onto the three ceilings above; picking the right chapter starts with correctly naming which ceiling you've actually hit.

1.3 A Scaling Roadmap

Put the last two sections together and a sane roadmap falls out almost on its own. First, confirm the database is well-tuned — indexed correctly, pooled correctly, running queries that do only the work they need to. Second, use the vertical headroom available before assuming you've outgrown a single instance; it's usually more generous than expected and costs nothing in application complexity.

Third, once you've genuinely hit a ceiling, name it precisely — write throughput, working-set size, or read volume — and reach for the specific tool matched to that ceiling, rather than the most impressive-sounding one. Read replicas for read volume. Partitioning for working-set and archival problems. Sharding, last and only when write throughput on a single primary is the actual, measured, unavoidable constraint.

This chapter's job was to earn that ordering, not just assert it. The rest of the book follows it chapter by chapter: connection management next, because it's cheap to fix and frequently mistaken for a real capacity problem, then read replicas, then partitioning, and only in Chapter 5 does sharding — the most expensive, most irreversible option — get its turn.

None of this is about avoiding complexity out of caution for its own sake. It's about spending complexity where it buys something, on the ceiling you've actually measured, instead of the one that sounded most likely from a distance. Every chapter ahead assumes you've done this diagnostic work first — they teach the techniques, but they don't tell you which one you need. This chapter does.

If you take one thing from it: the database you already have almost certainly has more room than you think, the ceiling you eventually hit is one of three specific things, and the fix that matches your actual ceiling is nearly always simpler, cheaper, and less risky than the one you'd reach for by reputation alone.

Chapter 2 starts with the ceiling most commonly mistaken for a capacity problem — connection exhaustion — and how to fix it in an afternoon instead of a quarter.

§ Chapter Summary

Seven things worth carrying into Chapter 2.

BEFORE YOU MOVE ON

01	Vertical scaling is a legitimate strategy, not a stopgap — it requires zero application changes and has more headroom than most teams test.
02	Before scaling anything, rule out missing indexes, connection exhaustion and wasteful queries — the most common causes of "we need to scale."
03	Relational databases break along three distinct ceilings — write throughput, working-set size, read volume — each with a different fix.
04	The wrong scaling strategy for your actual ceiling adds real complexity for no benefit — read replicas don't fix a write-throughput problem.
05	Sharding is the last, most expensive, least reversible option — reach for it only once you've named a write-throughput ceiling precisely.

BEFORE CHAPTER 2

For a database you operate, name which of the three ceilings — write throughput, working-set size, or read volume — you're actually closest to, using real metrics rather than instinct. If you're honestly not sure, that uncertainty is worth resolving before Chapter 2, because it determines which half of this book actually applies to you.

END OF SAMPLE

Seven chapters still to go.

Chapter 1 makes the case for sequencing — tune first, scale up next, scale out only against a named ceiling; the rest of the book covers each option in that order, from connection pooling through read replicas and partitioning to sharding, when you've genuinely earned it.