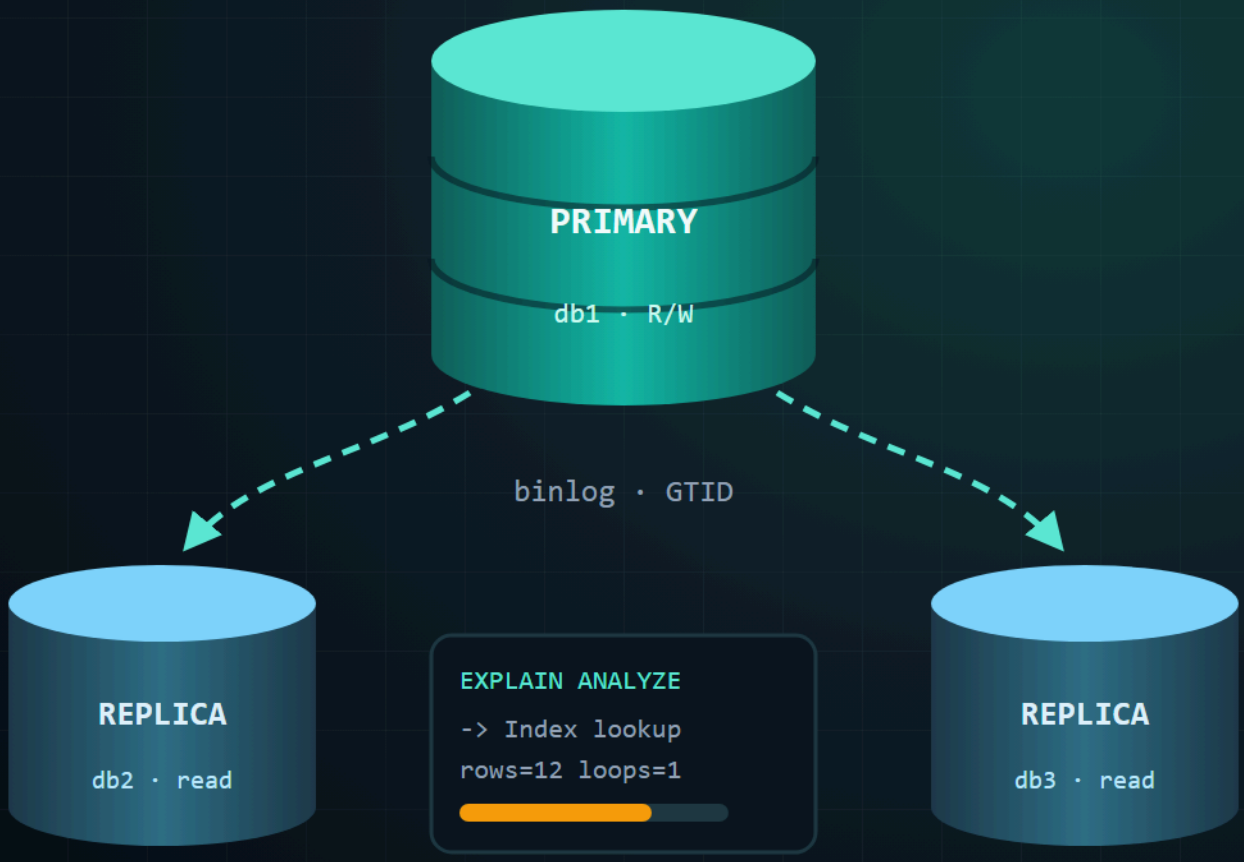


install · design · tune · secure · replicate · operate

FROM FIRST INSTALL TO PRODUCTION

MySQL from Setup to Production

Install, design, tune, secure, back up, and operate MySQL 8.4 with confidence

INDEXING · TUNING · SECURITY · BACKUPS · REPLICATION · HA

THENEXBOOKS

MySQL from Setup to Production

Install, Design, Tune, Secure, and Operate MySQL 8.4

First Edition

THENEXBOOKS

MySQL from Setup to Production
Install, Design, Tune, Secure, and Operate MySQL 8.4

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. Code examples and configuration templates 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. All traffic, data-volume, and capacity figures in the examples are illustrative assumptions, not measurements, and every configuration value is a starting point to be tested against your own workload. Always test backups, upgrades, and configuration changes outside production first, and consult the official MySQL documentation for the exact version you run.

MySQL is a registered trademark of Oracle and/or its affiliates. Percona, Percona XtraBackup, Percona Toolkit, ProxySQL, Vitess, gh-ost, Prometheus, Grafana, Flyway, Liquibase, Docker, Kubernetes, Linux, Ubuntu, Amazon Web Services, Microsoft Azure, Google Cloud, Java, and 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 independent and is not affiliated with, sponsored by, or endorsed by Oracle Corporation or any other company or project mentioned.

Kettleway, the running example in this book, is fictional, as are all clubs, members, instructors, vendors, incidents, and people mentioned in its examples. Any resemblance to real companies or persons is coincidental.

Preface

Getting MySQL running takes a few minutes. You install a package, start a service, and connect. Keeping it running — fast, correct, secure, and recoverable — while an application grows around it is a different job entirely, and it is the job this book is about.

Most teams learn that job the hard way. A query that took ten milliseconds on a laptop takes ten seconds once a table has a hundred million rows. An `ALTER TABLE` on a busy table blocks every write for an hour. A replica silently drifts. A backup that was "running every night" turns out never to have been restored, and cannot be. A default password or an open port becomes an incident report. None of these problems is exotic; they are the normal life of a production database, and each of them has a well-understood answer.

This book walks the whole path from a fresh installation to a production platform. We start with the basics — installing MySQL 8.4, configuring it deliberately, and understanding how InnoDB stores and protects your data. We then design a real schema, index it, and learn to read execution plans. We tune queries and the server, manage large tables, and change schemas without downtime. We secure the server and connect applications to it properly. Finally we back it up, replicate it, make it highly available, prepare for disasters, monitor it, and upgrade it. At every step we explain not only *what* to type, but *why* it works and *where* it breaks.

Who This Book Is For

- **Application developers** who use MySQL every day and want to understand what happens below their ORM — indexes, plans, transactions, and locks.
- **DevOps and platform engineers** who are responsible for running MySQL servers and need solid practices for configuration, backups, replication, monitoring, and upgrades.
- **Aspiring database administrators** who want a structured, practical path from first installation to production operations.
- **Tech leads and architects** deciding between self-managed MySQL, managed cloud services, and Kubernetes, or planning how a database should scale.

You should know basic SQL — `SELECT`, `INSERT`, `UPDATE`, and joins — and be comfortable on a Linux command line. No prior database administration experience is assumed.

What You Will Learn

- How to install MySQL 8.4 LTS on Linux, Windows, macOS, and Docker, and how to configure it deliberately instead of by accident.
- How the server and InnoDB work inside: the buffer pool, redo and undo logs, MVCC, and crash recovery.

- How to choose data types, character sets, and collations, and design a schema with keys and constraints that protect your data.
- How B-tree indexes really work, how to design composite and covering indexes, and how to read `EXPLAIN` and `EXPLAIN ANALYZE`.
- Transactions, isolation levels, locking, and deadlocks — and how to write code that survives them.
- Modern SQL in MySQL: JSON, common table expressions, window functions, and generated columns.
- Finding slow queries with the slow query log, Performance Schema, and the sys schema, and tuning both queries and the server.
- Partitioning and archiving large tables, and changing schemas online with instant DDL, gh-ost, and pt-online-schema-change.
- Accounts, roles, least privilege, TLS, encryption at rest, hardening, and auditing.
- Connecting applications with drivers, connection pools, and ORMs, and managing schema changes with Flyway or Liquibase.
- Backups and point-in-time recovery, replication with GTIDs, InnoDB Cluster, MySQL Router, scaling out, and disaster recovery.
- Monitoring and alerting, version upgrades, and running MySQL on managed cloud services and Kubernetes.

How This Book Is Organized

Part I — Getting Started (Chapters 1–5) introduces MySQL, installs it, gets you productive with the clients, configures the server, and explains how it works inside. **Part II — Designing Databases That Last** (Chapters 6–10) covers data types and character sets, schema design, indexing, transactions and locking, and modern SQL features. **Part III — Performance and Scale** (Chapters 11–15) teaches you to read execution plans, tune queries and the server, manage large tables, and change schemas safely. **Part IV — Security and Applications** (Chapters 16–20) secures accounts and data, hardens the server, and connects applications and deployment pipelines correctly. **Part V — Backup, Replication, and High Availability** (Chapters 21–25) protects your data and keeps it available, from backups and replication to InnoDB Cluster, scaling out, and disaster recovery. **Part VI — Running MySQL in Production** (Chapters 26–28) covers monitoring, upgrades, cloud and containers, and a complete end-to-end case study. The appendices provide a command cheat sheet, configuration templates, a production readiness checklist, and a glossary.

Each chapter ends with a **Key Takeaways** box for quick review.

The Running Example: Kettleway

Throughout the book we build and operate the database behind **Kettleway**, a fictional chain of fitness clubs. Members buy membership plans, book classes such as yoga, spin, and HIIT, check in at the door with a QR code, pay every month, and use a mobile app. Staff run the clubs from a back-office web application, and analysts produce monthly reports.

Kettleway starts small — three clubs and a single MySQL server — and grows to forty clubs in two regions. That growth is what makes it a useful teacher. The check-in table grows by the day and eventually needs partitioning and archiving. The last spot in a popular class invites race conditions. Payments demand correctness. Class timetables are read far more often than they are written, which makes read replicas attractive. And when the doors open at six in the morning, the database simply has to be up. Every concept in this book has a concrete home in Kettleway. All scale figures are assumptions we state explicitly.

Versions and Conventions

Examples target **MySQL 8.4 LTS** Community Edition, the long-term support release recommended for production at the time of writing, running on Linux. MySQL now publishes long-term support (LTS) releases alongside faster-moving Innovation releases; where a feature is version-dependent we say so and point you to the documentation rather than guessing. Replication examples use the current `SOURCE` / `REPLICA` terminology. Application examples use Java 21 with JDBC and HikariCP, with short Python and Node.js snippets where helpful, and local labs use Docker Compose.

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

> **NOTE**
> Background information, version details, or a useful aside.

> **WARNING**
> A common mistake or a behavior that can cause data loss, downtime, or security problems in production.

Code appears in `monospace`. Interactive sessions show their prompt: `mysql>` for the classic client, `MySQL JS >` for MySQL Shell, and `$` for a Linux shell. Money is stored as integer cents with a currency code, timestamps are stored in UTC, and long lines are wrapped to fit the page.

Contents

Preface

PART I — GETTING STARTED

1 Why MySQL: Architecture, Editions, Release Tracks, and Where It Fits

2 Installing MySQL 8.4 on Linux, Windows, macOS, and Docker

3 First Steps: The mysql Client, MySQL Shell, Accounts, and Your First Database

4 Configuration Fundamentals: Option Files, System Variables, and Sensible Defaults

5 Inside MySQL: Server Architecture, InnoDB, the Buffer Pool, Redo and Undo Logs

PART II — DESIGNING DATABASES THAT LAST

6 Data Types, Character Sets, and Collations: Getting utf8mb4 Right

7 Schema Design for Kettleway: Keys, Normalization, Constraints, and Naming

8 Indexing Deep Dive: B-Trees, Composite, Covering, Invisible, and Functional Indexes

9 Transactions, Isolation Levels, and Locking in InnoDB

10 Modern SQL in MySQL: JSON, CTEs, Window Functions, and Generated Columns

PART III — PERFORMANCE AND SCALE

11 Reading Execution Plans: EXPLAIN, EXPLAIN ANALYZE, and the Optimizer

12 Query Tuning in Practice: Slow Query Log, Performance Schema, and the sys Schema

13 Server Tuning: Memory, InnoDB, I/O, Threads, and Connections

14 Large Tables: Partitioning, Archiving, and Data Retention

15 Changing Schemas Safely: Instant DDL, Online DDL, gh-ost, and pt-online-schema-change

PART IV — SECURITY AND APPLICATIONS

16 Accounts, Authentication, Roles, and Least Privilege

17 Encryption and Secrets: TLS, Data-at-Rest Encryption, and Credential Handling

18 Hardening and Auditing: Securing the Server and Meeting Compliance Needs

19 Connecting Applications: Drivers, Connection Pools, ORMs, and Retries

20 Database Change Management: Migrations with Flyway and Liquibase in CI/CD

PART V — BACKUP, REPLICATION, AND HIGH AVAILABILITY

21 Backup and Recovery: mysqldump, MySQL Shell Dump Utilities, Physical Backups, and Point-in-Time Recovery

22 Binary Logs and Replication: GTIDs, Setting Up Replicas, and Monitoring Lag

23 High Availability: Group Replication, InnoDB Cluster, and MySQL Router

24 Scaling Out: Read Replicas, Proxies, and Sharding Strategies

25 Disaster Recovery: Multi-Region Designs, InnoDB ClusterSet, and Failover Drills

PART VI — RUNNING MYSQL IN PRODUCTION

26 Monitoring and Alerting: Key Metrics, Prometheus, Grafana, and Runbooks

27 Upgrades, Cloud, and Containers: Managed MySQL, Kubernetes Operators, and Version Upgrades

28 Case Study: Taking Kettleway from One Server to Production

APPENDICES

A MySQL Command Cheat Sheet

B Configuration Templates

C Production Readiness Checklist

D Glossary

PART I

Getting Started

What MySQL is and where it fits, how to install it on every major platform, how to work with its clients, how to configure it deliberately, and how the server and InnoDB work inside.

1 · Why MySQL
2 · Installing MySQL 8.4
3 · Clients, Shell, and First Database
4 · Configuration Fundamentals
5 · Inside MySQL and InnoDB

CHAPTER 1 · PART I — GETTING STARTED

Why MySQL: Architecture, Editions, Release Tracks, and Where It Fits

Before we install anything, we need a map: what MySQL actually is, how a query travels through the server, which edition and release track to run, and which problems MySQL solves well. We also meet Kettleway, the fitness-club chain whose database we will build, tune, secure, and operate from its first three clubs to forty clubs in two regions.

Meet Kettleway

Kettleway is a fictional chain of fitness clubs. When our story begins, it runs three clubs — Kettleway Downtown, Kettleway Riverside, and Kettleway North Quay — and everything the business knows lives in a single MySQL database called `kettleway` on one server. Members buy a membership plan, book classes such as yoga, spin, and HIIT, scan a QR code or a card at the door to check in, pay every month, and manage all of this from a mobile app. Front-desk staff and managers use a back-office web app. Once a month, analysts pull reports on attendance, churn, and revenue.

Behind those activities sit a handful of small services, and each one talks to MySQL in its own way:

- `members-api` — sign-ups, profile changes, plan changes, and freezes. Classic short transactions.
- `booking-api` — the class timetable and bookings. Very read-heavy (everyone checks the timetable), with sharp write spikes when popular classes open for booking.
- `checkin-gateway` — receives events from the door scanners. Small inserts, all day long, into a table that grows faster than anything else.
- `billing-worker` — charges members through the payment provider and records payments and invoices. Correctness matters more than speed here: a payment must never be recorded twice.
- `reporting` — monthly and ad-hoc reports. Long, scan-heavy queries that should never slow down the door scanners.
- The back-office app — staff search for members, fix bookings, and issue refunds.

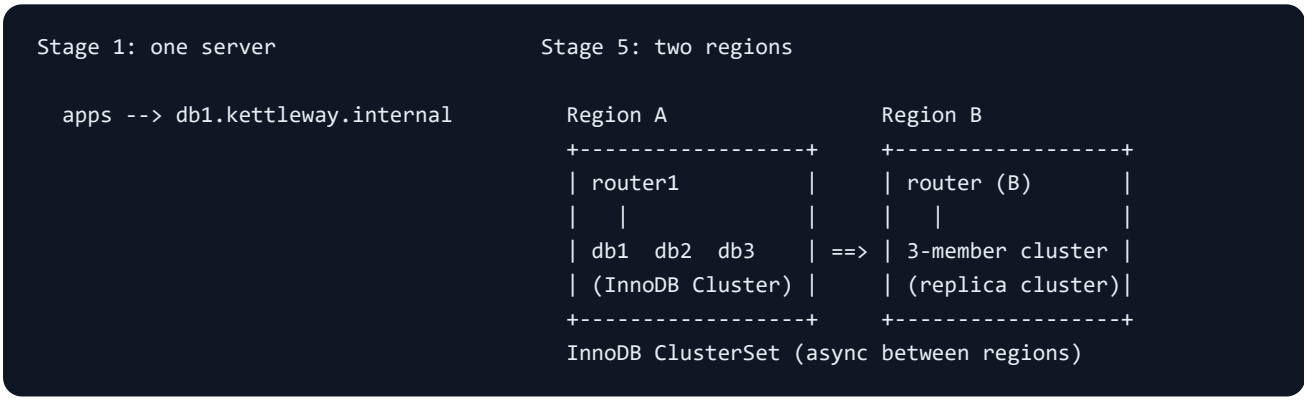
This mix is deliberately ordinary. Most production MySQL databases look like this: a core of relational data that must stay consistent, a time-series table that never stops growing, money that must be exact, a reporting workload that competes with the transactional one, and users who notice immediately when anything is slow at 6 p.m. on a Monday. Every chapter of this book uses Kettleway to make its topic concrete.

The journey from three clubs to forty

Kettleway grows as the book progresses. Let us assume the following milestones; the numbers are hypothetical and exist only to give each stage a realistic shape.

Stage	Kettleway	Database setup	Chapters
1. Launch	3 clubs, a few thousand members	One server, `db1`	2–10
2. Growth	12 clubs, slow reports, peak-hour latency	Tuned server, replica for reporting	11–15, 21–22
3. Scrutiny	Payment data, audits, many developers	Least privilege, TLS, migrations in CI	16–20
4. Always on	25 clubs, no downtime tolerated	InnoDB Cluster with MySQL Router	23–24
5. Two regions	40 clubs, second region	InnoDB ClusterSet, DR drills	25–28

At stage 1, a single server is the right answer. It is simple, cheap, and fast, and the most important work is getting the schema and the backups right. By stage 5, Kettleway runs a three-member cluster in each region with automatic failover inside a region and a rehearsed switchover between regions. The rest of the book is the path between those two pictures:



We will not jump to the complicated picture early. A recurring theme of this book is that every piece of infrastructure has a cost — in money, in operational effort, and in new ways to fail — and you should add each piece only when a concrete problem justifies it.

What MySQL Is

MySQL is a relational database management system: it stores data in tables with typed columns, lets you define keys and constraints between them, and answers queries written in SQL. It follows a client/server model. A long-running server process, `mysqld`, owns the data files on disk; clients such as the `mysql` command-line tool, MySQL Shell, or your Java application connect over TCP/IP or a local socket, send SQL, and receive result sets.

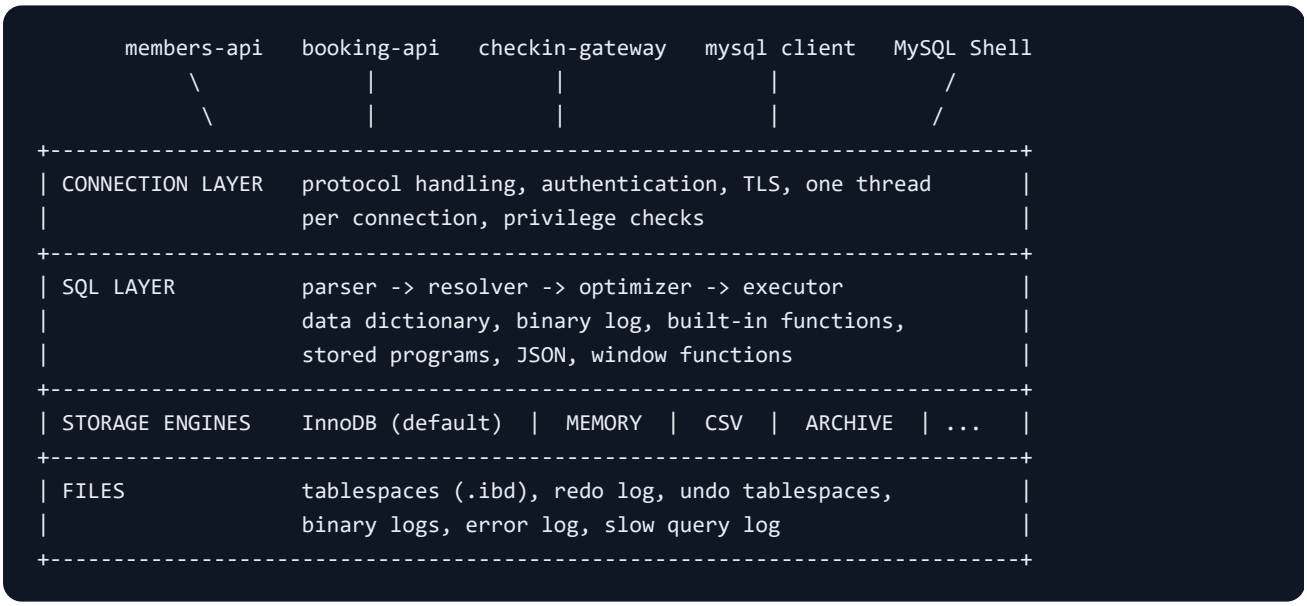
MySQL was first released in the mid-1990s by the Swedish company MySQL AB. Sun Microsystems acquired MySQL AB in 2008, and Oracle acquired Sun in 2010; Oracle has developed MySQL since then. The Community Edition is open source under the GNU General Public License version 2, and it is what this book uses throughout. MariaDB began as a fork of MySQL around the time of the Oracle acquisition. The two projects have diverged considerably since then, so MariaDB is not a drop-in replacement for every MySQL 8.x feature — system variables, replication details, JSON handling, and authentication differ. Everything in this book targets MySQL 8.4 itself.

Three properties explain why MySQL has remained so widely used for web and business applications:

- **It is transactional and durable by default.** The default storage engine, InnoDB, provides ACID transactions, row-level locking, crash recovery, and foreign keys. A committed payment stays committed after a power cut.
- **It is operationally approachable.** A single server can be installed in minutes, and the path to replicas, backups, and high availability uses tooling that ships with MySQL itself: binary logs, replication, the clone plugin, MySQL Shell, Group Replication, and MySQL Router.
- **The ecosystem is enormous.** Every mainstream language has mature drivers; every major cloud provider offers a managed MySQL service; and tools for backups, schema changes, monitoring, and proxying are widely available, many of them open source.

Architecture at a Glance

You do not need to know MySQL's internals to install it, but a rough mental model makes almost every later chapter easier. When the `booking-api` asks for tomorrow's spin classes at Kettleway Riverside, the request passes through three layers:



The connection layer

Each client connection is authenticated against an account defined as `'user'@'host'`. In MySQL 8.4, new accounts use the `caching_sha2_password` authentication plugin by default, and the older `mysql_native_password` plugin is disabled unless you explicitly enable it. Once connected, a session is served by a server thread, and every statement is checked against the privileges granted to that account. Connections are not free: each one uses memory and a thread, which is why the default `max_connections` of 151 is a deliberate limit rather than an oversight, and why applications use connection pools. Chapters 16 and 19 cover accounts and pooling in depth.

The SQL layer

The SQL layer turns text into results. The parser checks syntax and builds a parse tree; the resolver looks up tables and columns in the data dictionary; the optimizer chooses a plan — which index to use, which table to read first, whether to use a nested-loop or hash join — based on statistics; and the executor runs the plan, pulling rows from the storage engine. Chapter 11 teaches you to read the optimizer's decisions with `EXPLAIN` and `EXPLAIN ANALYZE`.

Two historical features are worth mentioning because older blog posts still recommend them. First, since MySQL 8.0 table definitions live in a **transactional data dictionary** stored in InnoDB, not in loose `.frm` files, which makes DDL far more robust after crashes. Second, the **query cache was removed** in 8.0. If you find advice to tune `query_cache_size`, it does not apply to MySQL 8.4 — caching of results now belongs in the application or a dedicated cache.

The SQL layer also writes the **binary log**, a record of every change made to the data. Binary logging is enabled by default, and the default format is row-based. The binary log is the foundation of replication and point-in-time recovery, which is why it lives above the storage engines: it records changes regardless of which engine made them.

The storage engine layer

MySQL's pluggable storage engine architecture means that the SQL layer talks to storage through an internal handler interface, and different engines can implement it differently. In practice, modern MySQL means InnoDB. It is the default engine, it is the only mainstream engine with transactions and crash recovery, and features such as Group Replication and the clone plugin require it. You can list the engines your server supports:

```
mysql> SELECT engine, support, transactions
    ->   FROM information_schema.engines
    ->  ORDER BY engine;
+--------------------+---------+--------------+
| ENGINE             | SUPPORT | TRANSACTIONS |
+--------------------+---------+--------------+
| ARCHIVE            | YES     | NO           |
| BLACKHOLE          | YES     | NO           |
| CSV                | YES     | NO           |
| InnoDB             | DEFAULT | YES          |
```

```
| MEMORY             | YES     | NO           |
| MRG_MYISAM         | YES     | NO           |
| MyISAM             | YES     | NO           |
| PERFORMANCE_SCHEMA | YES     | NO           |
| ...                | ...     | ...          |
+--------------------+---------+--------------+
```

The exact list depends on how your server was built. The important column is `TRANSACTIONS`: only InnoDB says `YES`. MyISAM, the default engine long ago, has no transactions, locks whole tables for writes, and can be damaged by a crash. Every Kettleway table uses InnoDB, and so should yours unless you have a very specific reason otherwise.

> **WARNING**
>
> If you inherit an older database, check for non-InnoDB tables before relying on transactions, replication, or backups. A single MyISAM table that receives writes can silently break the consistency guarantees you think you have. This query finds them: `SELECT table_schema, table_name, engine FROM information_schema.tables WHERE engine <> 'InnoDB' AND table_schema NOT IN ('mysql', 'information_schema', 'performance_schema', 'sys');`

Inside InnoDB, briefly

InnoDB keeps frequently used data and index pages in memory in the **buffer pool**. Every table is organized as a B-tree clustered on its primary key, and secondary indexes point back to that primary key. Changes are first recorded in the **redo log**, so that a commit can be made durable with a fast sequential write instead of rewriting data pages immediately; the modified pages are flushed to the tablespace files later. The **undo log** keeps older versions of rows so that readers see a consistent snapshot without blocking writers — this is multi-version concurrency control (MVCC), and it is why a long report in REPEATABLE READ, the default isolation level, can keep reading a stable view while check-ins keep arriving. Chapter 5 explains each of these structures, and Chapter 9 explains the locking that sits on top of them.

For now, remember one practical consequence: the buffer pool is the single most important memory setting, and its default size of 128MB is tiny for a real server. Chapters 4 and 13 show how to size it.

Editions and Products

"MySQL" is a family of products, and the names are easy to confuse. The following table sorts out the ones you will meet.

Product	What it is	License
MySQL Community Server	The database server this book uses	GPL v2
MySQL Enterprise Edition	Community server plus commercial tools and support	Commercial
MySQL Shell	Modern client, admin API, dump and load utilities	GPL v2 / commercial

MySQL Router	Lightweight routing layer for InnoDB Cluster	GPL v2 / commercial
MySQL Workbench	Graphical design and administration tool	GPL v2 / commercial
MySQL NDB Cluster	Separate distributed engine (NDB), different design	GPL v2 / commercial
HeatWave MySQL	Oracle's managed MySQL cloud service	Cloud service

Enterprise Edition adds commercial components on top of the same core server: MySQL Enterprise Backup for hot physical backups, Enterprise Audit, Enterprise Firewall, Enterprise Transparent Data Encryption with key-management integration, Enterprise Monitor, and Oracle support. Most of these needs can also be met with Community tools — Percona XtraBackup for physical backups, for example, or the Community Edition's own InnoDB data-at-rest encryption with a keyring component — and later chapters describe both routes neutrally. Whether a support contract is worth it is a business decision, not a technical one.

> **NOTE**
>
> Do not confuse **MySQL NDB Cluster** with **InnoDB Cluster**. NDB Cluster is a separate product built on the NDB storage engine, with its own data nodes and management nodes, designed for specific high-throughput, shared-nothing use cases. InnoDB Cluster, covered in Chapter 23, is ordinary MySQL servers running InnoDB and Group Replication, managed by MySQL Shell and fronted by MySQL Router. Kettleway uses InnoDB Cluster.

Release Tracks: LTS and Innovation

For most of MySQL's history, a major version such as 5.7 or 8.0 received both bug fixes and new features in its patch releases. That changed in 2023–2024. MySQL now publishes two kinds of releases:

- **Long-Term Support (LTS) releases**, such as **MySQL 8.4 LTS**, released in 2024. Patch releases within an LTS series (8.4.x) focus on bug and security fixes and avoid behavior changes, so you can apply them with low risk.
- **Innovation releases**, the 9.x series, which ship new features on a frequent cadence. Each Innovation release is supported only until the next one appears, so running them in production means upgrading often.

Features developed in Innovation releases are eventually collected into the next LTS release. The trade-off is familiar from other software: the LTS track gives stability and a long support window; the Innovation track gives earlier access to features at the cost of frequent upgrades.

```
          2024                    2025                    2026
LTS       8.4.0 ---- 8.4.x ---- 8.4.x ---- 8.4.x ---- 8.4.x ---- ...  (fixes only)
Innov.    9.0 --> 9.1 --> 9.2 --> ... (each supported until the next)
8.0       8.0.x ----------------------------------------X  end of life April 2026
```

MySQL 8.0, the long-running previous series, reached end of life in April 2026. If you still run 8.0 anywhere, planning an upgrade to 8.4 LTS is overdue; Chapter 27 covers upgrade procedures and the checks MySQL Shell can run beforehand.

This book targets **MySQL 8.4 LTS, Community Edition**, and we recommend an LTS release for production. A newer LTS release may exist by the time you read this, so check the MySQL release notes. Most of what you learn here — schema design, indexing, transactions, replication, operations — carries forward unchanged; the release notes list the defaults and features that differ.

Once a server is running, you can confirm what you are talking to:

```
mysql> SELECT VERSION(), @@version_comment, @@version_compile_os;
+-----------+-----------------------------+---------------------+
| VERSION() | @@version_comment           | @@version_compile_os |
+-----------+-----------------------------+---------------------+
| 8.4.x     | MySQL Community Server - GPL | Linux               |
+-----------+-----------------------------+---------------------+
```

We write `8.4.x` in sample output throughout the book; your server will show its exact patch release.

What 8.4 changed that you will notice

If you have used MySQL 5.7 or 8.0 before, a few 8.4 changes will affect you early:

- **Replication terminology.** The old MASTER/SLAVE statements are gone. You use `CHANGE REPLICATION SOURCE TO`, `START REPLICA`, `STOP REPLICA`, `SHOW REPLICA STATUS`, and `SHOW BINARY LOG STATUS`. Scripts written for older versions must be updated.
- **Authentication.** `mysql_native_password` is disabled by default. Old drivers that only support that plugin will fail to connect until they are upgraded — or until you deliberately enable the deprecated plugin, which should be a temporary measure.
- **Removed tools.** `mysqlpump` no longer exists. Use the MySQL Shell dump utilities or `mysqldump`, covered in Chapter 21.
- **Changed defaults.** For example, `innodb_adaptive_hash_index` is now `OFF` by default. Chapter 4 shows how to see every variable that differs from its default on your server.

Where MySQL Fits — and Where It Does Not

MySQL is a general-purpose relational database optimized for online transaction processing (OLTP): many concurrent, short transactions that read and write small numbers of rows by key. That describes the core of most web and mobile back ends, including Kettleway's. The following table maps each Kettleway workload to the MySQL features that serve it and the chapters where we build them.

Kettleway workload	Shape	MySQL features	Chapters
Members, plans, memberships	Relational OLTP	InnoDB, keys, constraints	7, 9

Class timetable	Read-heavy	Indexes, read replicas	8, 24
Bookings for popular classes	Contended writes	Unique keys, row locks	9
Door check-ins	Append-only time series	Partitioning, archiving	14
Payments and invoices	Money, exactly once	Integer cents, idempotency keys	7, 19
Device and app metadata	Semi-structured	JSON, generated columns	10
Monthly reports	Scans, aggregates	Window functions, replica	10, 22

Strengths

- **Primary-key and index lookups at high concurrency.** InnoDB's clustered primary key makes fetching a member, a booking, or a payment by key efficient, and row-level locking lets many sessions write different rows at once.
- **Replication that is simple to start.** Adding a read replica for reporting takes a backup and a few statements, and the same binary log gives you point-in-time recovery.
- **Built-in high availability.** Group Replication, InnoDB Cluster, and MySQL Router provide automatic failover without third-party software, and InnoDB ClusterSet extends that across regions.
- **Portability of skills and hosting.** The same server runs on a laptop, in a container, on a virtual machine, or as a managed cloud service such as Amazon RDS for MySQL, Amazon Aurora MySQL-compatible, Azure Database for MySQL – Flexible Server, Google Cloud SQL for MySQL, or Oracle HeatWave MySQL.

Limits and alternatives

No database is best at everything. Knowing where MySQL is a poor fit keeps you from forcing it into roles it handles badly.

- **Large-scale analytics.** Aggregating billions of rows across many columns is the job of column-oriented engines and data warehouses. MySQL handles Kettleway's monthly reports comfortably on a replica, but if analysts start asking for five years of check-in history sliced a dozen ways, export the data to an analytical system. (Oracle's HeatWave service adds an in-memory analytics engine to managed MySQL; that is a cloud offering, not part of Community Server.)
- **Full-text search as a product feature.** InnoDB has `FULLTEXT` indexes, which are fine for simple keyword search in the back-office app. Relevance tuning, typo tolerance, and faceting are better served by a dedicated search engine fed from MySQL.
- **Unbounded write scaling on one primary.** A MySQL replication topology has one writable primary per data set. When one server cannot absorb the writes, you shard — split data across independent primaries — which adds real complexity. Chapter 24 discusses when that becomes necessary and tools such as Vitess.

- **Extensibility-heavy workloads.** If your design depends on custom data types, rich geospatial analysis beyond MySQL's spatial support, or a large extension ecosystem, PostgreSQL may be the better choice. Both are excellent relational databases; the choice usually comes down to team experience, ecosystem, and specific features rather than general speed.

> **TIP**
> Choosing a database is a decision you will live with for years. Instead of relying on generic comparisons, write down your three most important queries and your most important write path, prototype them on each candidate with realistic data volumes, and measure. Published benchmarks rarely match your workload.

A First Look at the Kettleway Schema

Chapter 7 designs the schema properly, discussing keys, normalization, and constraints. Here is the map of where we are heading, so the table names in the next few chapters are familiar:

```
instructors 1---* classes *---1 clubs 1---* members (home_club_id)
                    1               |
                    |               +---* checkins *---1 members
                    *
                bookings *---1 members

members 1---* memberships *---1 membership_plans
members 1---* payments
members 1---* invoices

(1---* reads "one ... many")
```

A few conventions hold across the entire schema. Table names are plural and snake_case. Every table has a `BIGINT UNSIGNED` surrogate primary key named `id`. Members also carry a `public_id`, a 26-character ULID-style identifier that is safe to expose in URLs and the mobile app, so internal ids never leak. Money is stored as integer cents plus an ISO currency code — never as `FLOAT` or `DOUBLE`, which cannot represent most decimal amounts exactly. Timestamps are stored in UTC, and each club records its own time zone for display.

Here is the smallest table, to give you a feel for the style:

schema/001_clubs.sql

```
CREATE TABLE clubs (
  id         BIGINT UNSIGNED NOT NULL AUTO_INCREMENT,
  code       VARCHAR(16)     NOT NULL,          -- e.g. 'KW-DT01'
  name       VARCHAR(100)    NOT NULL,
  city       VARCHAR(100)    NOT NULL,
  timezone   VARCHAR(64)     NOT NULL,          -- IANA name, e.g. 'Europe/Stockholm'
  opened_on  DATE            NOT NULL,
  PRIMARY KEY (id),
```

```
  UNIQUE KEY uq_clubs_code (code)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

INSERT INTO clubs (code, name, city, timezone, opened_on) VALUES
  ('KW-DT01', 'Kettleway Downtown',    'Stockholm', 'Europe/Stockholm', '2024-03-01'),
  ('KW-RS01', 'Kettleway Riverside',   'Stockholm', 'Europe/Stockholm', '2024-09-15'),
  ('KW-NQ01', 'Kettleway North Quay',  'Stockholm', 'Europe/Stockholm', '2025-02-01');
```

The explicit `ENGINE`, character set, and collation merely spell out what MySQL 8.4 would choose anyway: InnoDB, `utf8mb4`, and `utf8mb4_0900_ai_ci`. Chapter 6 explains why `utf8mb4` — and not the older, misleadingly named `utf8` — is the right choice for member names and emails.

The table that will cause the most interesting problems is `checkins`. Every door scan adds a row. Let us assume Kettleway eventually records about 120,000 check-ins per day across 40 clubs; that is over 40 million rows a year, forever, unless we plan retention. That single table drives much of the discussion of indexing, partitioning, archiving, and backup time in later chapters.

```
CREATE TABLE checkins (
  id             BIGINT UNSIGNED NOT NULL AUTO_INCREMENT,
  member_id      BIGINT UNSIGNED NOT NULL,
  club_id        BIGINT UNSIGNED NOT NULL,
  checked_in_at  DATETIME(3)     NOT NULL,   -- UTC, millisecond precision
  gate           VARCHAR(32)     NOT NULL,
  device         JSON            NULL,       -- scanner model, firmware, app version
  PRIMARY KEY (id),
  KEY ix_checkins_club_time (club_id, checked_in_at)
) ENGINE=InnoDB;
```

What Goes Wrong in Production

Most MySQL incidents are not caused by exotic bugs. They come from a small set of predictable mistakes, and much of this book is about avoiding them. It helps to see them listed at the start:

- **Running with default memory settings.** A server with 64GB of RAM and a 128MB buffer pool reads from disk constantly. Chapter 4 fixes the basics and Chapter 13 goes deeper.
- **Backups that were never restored.** A backup job that "succeeds" every night is worthless if nobody has ever restored from it and measured how long it takes. Chapter 21 treats restore drills as part of the backup.
- **The application connecting as root.** A single over-privileged account turns any SQL injection or leaked configuration file into a total compromise. Chapter 16 builds `kw_app`, `kw_readonly`, and role-based `kw_admin` accounts with least privilege.
- **Missing or wrong indexes.** A query that was instant with 3 clubs of test data becomes a full table scan at 40 clubs. Chapters 8, 11, and 12 teach you to find and fix these before customers notice.

- **Schema changes that lock a busy table.** An `ALTER TABLE` on `checkins` at 6 p.m. can stall the door scanners. Chapter 15 covers instant, online, and tool-assisted schema changes.
- **Replicas nobody watches.** A replica that is hours behind serves stale timetables and would lose data in a failover. Chapters 22 and 26 cover lag and alerting.
- **Failover that was never tested.** High availability that has never been exercised usually fails at the worst moment. Chapter 25 makes failover drills a routine.

None of these is difficult to prevent once you know to look for it. The hard part is usually that the problem is invisible while the system is small, which is precisely why we follow Kettleway from a small start.

How to Use This Book

The book is organized in six parts that mirror the life of a production database. Part I (Chapters 1–5) installs MySQL, introduces the clients, and explains configuration and InnoDB internals. Part II (Chapters 6–10) designs the Kettleway schema: data types, character sets, keys, indexes, transactions, and modern SQL. Part III (Chapters 11–15) is about performance: execution plans, query tuning, server tuning, large tables, and safe schema changes. Part IV (Chapters 16–20) covers security and the application side: accounts, encryption, hardening, drivers and pools, and migrations in CI/CD. Part V (Chapters 21–25) covers backup, replication, high availability, scaling out, and disaster recovery. Part VI (Chapters 26–28) is about running the system: monitoring, upgrades and cloud options, and a closing case study that walks Kettleway from one server to its final production design.

You can read straight through, and the chapters build on each other in that order. If you already run MySQL, you can also jump to a specific part; each chapter repeats the `CREATE TABLE` statements it needs, so you are never forced to hunt for a definition.

Conventions

Interactive sessions show their prompt so you always know where a command runs:

```
$ sudo systemctl status mysql          # Linux shell (Ubuntu 24.04 unless stated)
mysql> SELECT COUNT(*) FROM members;   # classic mysql client
 MySQL  JS > dba.checkInstanceConfiguration()   # MySQL Shell, JavaScript mode
 MySQL  SQL > SHOW DATABASES;                   # MySQL Shell, SQL mode
```

SQL keywords are uppercase and identifiers are snake_case. Option-file snippets show the `[mysqld]` section and name the file they belong in: `/etc/mysql/mysql.conf.d/mysqld.cnf` on Debian and Ubuntu packages, `/etc/my.cnf` on RPM-based systems such as Oracle Linux and RHEL-compatible distributions. Hosts follow the pattern `db1.kettleway.internal` for the primary, `db2` and `db3` for replicas or cluster members, and `router1` for MySQL Router. Example people, such as Maya Lindqvist and Tomas Oyelaran, are invented, and all email addresses use `example.com`.