MySQL 8.4 for Oracle DBAs — Free Sample

About This Free Sample

Preface

Lab Environment Guide — Getting Started on macOS

Part I: MySQL 8.4 Architecture

Chapter 1. MySQL for Oracle DBAs — Shifting Perspective

Chapter 2. MySQL 8.4 Overview and the LTS Release Model

Chapter 3. MySQL Server Architecture — Compared with an Oracle Instance

Through a DBA's Eyes

Series 01

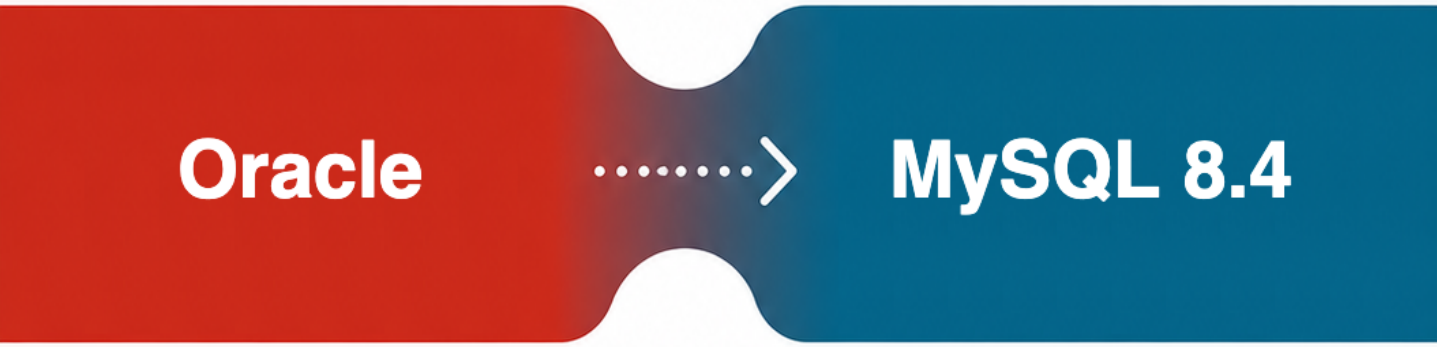
For the Oracle DBA

MySQL 8.4

in Practice

Leverage what you already know —
Every result came from a real run

✓ Recommended for both DBAs & Developers



About This Free Sample

This sample contains the front matter and the first three chapters of **MySQL 8.4 for Oracle DBAs** — enough to see how the whole book works: every chapter opens with *Through an Oracle DBA's Eyes*, mapping the Oracle concept you already know onto its MySQL counterpart, and every measured result comes from a real run.

The full edition (continuously updated — buy once, all future updates free):

- **Part I — MySQL 8.4 Architecture** (ch. 1–10): server architecture, InnoDB, memory, transactions/MVCC, redo + binlog, the data dictionary, physical files
- **Part II — Query Fundamentals** (ch. 11–20): the SQL dialect, data types, joins, NULL, DML/DDL differences from Oracle
- **Part III — Advanced Queries and Tuning** (ch. 21–32): EXPLAIN, the optimizer, index design, plan stability without SPM, tuning methodology
- **Part IV — Operations Fundamentals** (ch. 33–44): accounts, backup/recovery and PITR, monitoring and automation
- **Part V — Advanced Operations** (ch. 45–54): performance analysis, locks and deadlocks, connection pools, incident response, upgrades
- **Part VI — High Availability and DR** (ch. 55–62): replication, GTID, InnoDB Cluster, Router, failover procedures
- **Part VII — Security and Compliance** (ch. 63–67): encryption, audit logging, masking, certification audits

- **Part VIII — Operating at Scale** (ch. 68–73): big-table design, an ~500-million-row cleanup, online schema changes, partitioning
- **Part IX — Case Studies** (ch. 74–79): real incidents, symptom to lessons
- **Appendices A1–A9**: AWS RDS for MySQL, Aurora, and the 8.4/9.x version map

Preface

The author built his career as an Oracle DBA — operating and delivering projects on large Oracle Exadata estates at financial companies, and working on projects for a semiconductor company. At one company, a single large table was 40 TB and the whole database was 800 TB — and that was *after* HCC compression; converted to MySQL terms, it would be at least 1.5 PB. That history left one side effect. When MySQL landed on my desk one day, the instinct trained on that scale said: “at this size, this should be nothing.”

That judgment was only half right. The volumes MySQL handles are indeed far smaller than the Exadata world. But operational difficulty does not scale with volume. Things Oracle did for you had to be done by hand, there were kinds of traps Oracle simply does not have, and above all the premise itself — “a small Oracle” — was wrong. This book was written so that the time the author paid to correct that premise can be saved by the next person.

The premise collapsed in concrete moments. Statistics, a managed asset in Oracle, turned out to be a wobbling sample here; Flashback, which I had thought of as a safety net, did not exist; execution plans changed all the time, with no way to manage them as precisely as Oracle allows. What was truly disorienting in front of an incident was not the absence of knowledge but the fact that **the knowledge I had did not apply** — the hand still reached for a V$ view, and the view was not there. What was needed was not to throw away the Oracle experience, but to set it down again on new terrain.

So this book was written, from beginning to end, in the form of an Oracle → MySQL translation. Every chapter is structured as “in Oracle it was like this; in MySQL it is like that,” and everything that could be executed was executed and confirmed while writing. Note that measured values can come out differently depending on version and environment — that boundary is spelled out up front, in the first section of the “Lab Environment Guide.”

One thing should be said in advance: this book does not rank the two DBMSs. Oracle and MySQL are tools with different origins and different design philosophies, and each choice has its reasons. The practical question is never “which is better” but “which fits this situation, now.” The reader’s Oracle knowledge and experience are the lever this whole book stands on.

This book hopes for one thing: that the time the author paid in trial and error, the reader can skip by table of contents. And that someday, sitting alone in front of an unfamiliar database incident, some chapter of this book is there with you. If it does that much, this book has done its job.

Kei Kim

Who This Book Is For

- **DBAs with Oracle experience** — every chapter of this book is written in the structure “in Oracle it was X; in MySQL it is Y.” Building on the knowledge you already have is this book’s method.
- Operators who have inherited — or are about to inherit — a MySQL server; each chapter carries checklists for “what to look at first on a server you inherited.”
- Operators in managed environments (RDS) — the entire appendix and RDS notes throughout assume that environment.
- Developers with a serious interest in databases — those who want to go beyond writing SQL, understand the database architecture precisely, and develop and tune SQL on top of that understanding.

This is not a SQL primer. It assumes relational database fundamentals and operational experience.

How This Book Is Organized

- **Part 1, Architecture (ch. 1–10)**: server structure, InnoDB, memory, transactions and MVCC, the two logs, the dictionary, and physical files, mapped against their Oracle counterparts.
- **Part 2, Query Basics (ch. 11–20)**: SQL dialect differences, data types, NULL, joins, and the absence of sequences and its replacement.
- **Part 3, Advanced Queries and Tuning (ch. 21–32)**: the heart of the book — execution plans, the optimizer, indexes, and the craft of “doing by hand what Oracle did for you.”

- **Part 4, Operations Basics (ch. 33–44)**: configuration, accounts, backup, and an in-database automation suite.
- **Part 5, Advanced Operations (ch. 45–54)**: a performance analysis methodology; type-by-type diagnosis of locks, connections, memory, I/O, and replication lag; incident response; upgrades.
- **Part 6, High Availability and DR (ch. 55–62)**: replication architecture, GTID, semi-synchronous replication, InnoDB Cluster and Router, DR design, and failover.
- **Part 7, Security and Compliance (ch. 63–67)**: encryption, audit logging, masking, access control, and compliance audits.
- **Part 8, Operating at Scale (ch. 68–73)**: big-table design, chunked deletes, online schema change, partitioning, bulk loading, and data lifecycle.
- **Part 9, Case Studies (ch. 74–79)**: the record of the incidents that gave birth to the preceding chapters. The final chapter reads public external cases in the same language.
- **Appendix (A1–A9)**: RDS for MySQL and Aurora, plus a map of MySQL 8.4/9.x new features — parameter groups, Multi-AZ, snapshots and PITR, CloudWatch, and everything else that changes in a managed environment.

The book is designed to be read in order, but a reader in a hurry can read by part — the symptom-classification table in chapter 45 is the entrance to Part 5, and the "Summary" at the end of each chapter serves as an index.

Conventions

- **Test cases**: each chapter carries executable verification procedures. Code blocks verified by actual execution carry a `<!-- verified: mysql-8.4.x, date -->` comment; anything not yet verified, or unverifiable in the environment, is marked `<!-- REVIEW -->`. Every number and output presented as an execution result came from a real run — a rule this book set for itself. Those values are, however, a record of the writing-time version and environment; a different version or environment may not reproduce them — the first section of the "Lab Environment Guide" covers this in detail.

- **Lab environment**: a single Docker `mysql:8.4` image covers most test cases — the setup procedure (macOS-based) is in the "Lab Environment Guide" right after this preface. SQL code blocks in the body assume you are inside a `mysql>` prompt opened with `docker exec -it book-mysql84 mysql -uroot -pbook --default-character-set=utf8mb4` (bash blocks run in the host terminal). RDS-only content is marked `RDS-only`.
- **Diagrams**: Oracle-side components are drawn in red tones, MySQL-side in blue tones, and neutral elements in gray, consistently.
- **Terminology**: standard MySQL documentation terms are used throughout; names that are de facto proper nouns — GTID, binlog, and the like — are left as-is.
- **Cases**: the author's cases (chapters 74–78 and anecdotes in the body) all come from real operations. Information that could identify a company or service has been anonymized and generalized, and details for which no records remain (some settings and timelines) were reconstructed with typical values — the key figures stated in the body (3,614 ms, 96%, and so on) are real values. As an exception, the external cases in chapter 79 are cited by name from each company's own official public sources, with reference footnotes.
- **Footnotes = source references**: statements of fact that can drift with time — versions, dates, support windows, defaults — carry footnote numbers pointing to the official documents used to verify them. They are collected after the body as the notes list.

Lab Environment Guide — Getting Started on macOS

The test cases in this book are designed to run **inside Docker containers only**. You do not install MySQL on the host OS, and the same commands work the same way on macOS, Linux, or Windows (WSL2). This page walks through the setup, macOS-based.

About Reproducibility — Read This First

The test cases and measured values in this book are a record of actual runs in a specific environment at writing time — the Docker `mysql:8.4` image, at whatever minor version was current. Following the same commands may produce different results, and the cause is almost always one of three things.

- **Version**: even a minor update can change optimizer decisions, parameter defaults, or output formats. Later 8.4 patches, or the next major version, may change some behaviors outright.
- **Environment**: CPU and disk performance, buffer pool state (cold or warm), and container resource limits all shift timings and ratios.
- **Data**: a different load order or a different statistics sample can produce a different execution plan for the same query.

So the measured values in this book are not a promise that "you will get this number anywhere" — they are **reference records that exist to explain the principles**. If you follow along and your numbers differ, nothing is wrong; what matters is not the decimals but the order of magnitude and the direction (what is roughly how many times faster, which way the plan leans). If the *direction* itself differs from the book, that is in fact a moment worth learning from — going back to the version release notes and the chapter's principles to find out why is exactly the diagnostic training this book aims to build.

One more note — the test cases in this book are **an explanatory device, not a lab workbook.** Their purpose is to show, by execution, that the principles in the body really behave as described; the results are printed alongside so that reading with your eyes is enough. For readers who do want to run things, the environment and commands are provided, but some are abbreviated for the page — outputs are excerpted to the relevant part, repeated commands are shown once, and setup blocks sometimes refer back to an earlier chapter as "ch. N, TC N-M." If you get stuck while following along, read the surrounding body text and the referenced chapter together.

Prerequisites — Docker Desktop, and Nothing Else

Install Docker Desktop (OrbStack works identically), then confirm in a terminal:

```
docker version    # Client/Server info means you are ready
```

The Standard Lab Container

Most chapters can be followed with this one container.

```
docker run -d --name book-mysql84 -e MYSQL_ROOT_PASSWORD=book mysql:8.4

# Connect (mind the character set — see the trap below)
docker exec -it book-mysql84 mysql -uroot -pbook --default-character-set=utf8mb4
```

This connection is an **interactive session that stays open**, like sqlplus — you keep typing SQL at the `mysql>` prompt and only `exit` when you are done. There is no need to prefix docker commands every time (the `docker exec ... -e "SQL"` form is for one-line checks and script automation). The book's code blocks follow the same rule: **`sql blocks run inside the connected `mysql>` prompt;` bash blocks run in the host terminal.**

Port publishing (`-p`) is deliberately omitted — every test case in this book runs inside the container via `docker exec`, so no host port is needed. Open one only when you also want a GUI tool such as DBeaver to connect, e.g. `-p 13306:3306`.

The first connection takes a moment (measured). On first startup the container spends tens of seconds initializing data — connecting during that window can fail with `ERROR 1045 (Access denied)` or a refused connection (a temporary bootstrap server is briefly running). Watch `docker logs book-mysql84` and connect after **`ready for connections` appears a second time** (after the temporary server).

The port-conflict trap (measured on macOS). Starting with `-p 3306:3306` can fail with `Bind for 0.0.0.0:3306 failed: port is already allocated` — a local MySQL or another container already owns 3306 (check with `lsof -nP -i :3306`). The follow-up trap is that **the failed container still occupies the name** — remove it with `docker rm book-mysql84`, then start again without the port, or on another port (`-p 13306:3306`).

Why the password is short. The lab password `book` is accepted because the official image ships without the validate_password component installed. Measured — after `INSTALL COMPONENT 'file://component_validate_password';` the same password is rejected with ERROR 1819 (policy requirements) (revert with `UNINSTALL COMPONENT ...` after checking). It is deliberately simple for an isolated lab container; production password policy is covered in chapter 35.

One character-set trap (measured). Depending on your terminal locale, the mysql client may connect as latin1, and non-ASCII data — accented characters, emoji, CJK strings — gets corrupted on display. Make `--default-character-set=utf8mb4` a habit when connecting.

Bundled tools (measured). The mysql:8.4 image ships mysql, mysqladmin, mysqldump, and **mysqlsh (8.4)**. It does **not** ship mysqlbinlog — when you need it (as in the chapter 41 PITR lab), borrow it from the percona-server container below (no host install needed).

```
        # Example: process a binlog file extracted from the container
        # with the percona-server image's mysqlbinlog
        docker run --rm -v "$PWD":/work percona/percona-server:8.4 \
          mysqlbinlog -v /work/binlog.000003
```

Resetting Between Chapters — No Leftover Worries

Each chapter's test-case setup block starts with `CREATE DATABASE IF NOT EXISTS labs;` and `DROP TABLE IF EXISTS ...` — you can run it as-is even if a same-named table from an earlier chapter is still around. When you want to reset the lab wholesale, either of these is enough:

```
        DROP DATABASE IF EXISTS labs;   -- reset just the lab schema
```

```
        docker rm -f book-mysql84
# reset the whole container (restart with the command above)
```

Additional Images by Chapter

Image	Used in	Purpose
`mysql:8.4`	whole book	base labs + mysqlsh (incl. AdminAPI)
`mysql:8.0`	ch. 54	starting version for the upgrade lab
`gvenzl/oracle-free`	ch. 47 etc.	verifying Oracle-side behavior for comparison
`percona/percona-server:8.4`	ch. 64, toolbox	audit component, mysqlbinlog, etc.
`percona/percona-xtrabackup:8.4`	ch. 40–41	physical backup lab
`percona/percona-mysql-router:8.4`	ch. 60	Router bootstrap lab

A note for Apple Silicon (M-series). `mysql` and `gvenzl/oracle-free` provide arm64 images. If a Percona-family image fails to start due to architecture, run it under emulation with `--platform linux/amd64` (accept the slowdown — it does not affect the labs functionally).

Chapters That Need Multiple Containers (Replication and Cluster)

Chapters 51 and 55–60 use two or three containers. Create one dedicated network and the rest is exactly the test-case commands in each chapter — use the network name each chapter specifies (`repl-net` for the replication labs, `grnet` for group replication).

```
# Example: network and first node for the replication labs (ch. 51, 55–58)
docker network create repl-net
docker run -d --name src --network repl-net -e MYSQL_ROOT_PASSWORD=book \
  mysql:8.4 --server-id=1
```

Cleanup

Remove finished containers and networks like this. Chapters that created explicit volumes (ch. 40–41) need the volumes removed too before the disk space comes back.

```
docker rm -f book-mysql84 src rep
docker network rm repl-net
docker volume prune   # run after reviewing what it lists
```

Part I: MySQL 8.4 Architecture

Chapter 1. MySQL for Oracle DBAs — Shifting Perspective

Learning Objectives

- Explain where MySQL came from, why Oracle came to own it, and what traces that history left on today's ecosystem (MariaDB, Percona)
- Explain the design-philosophy difference between Oracle and MySQL — "everything built into a single engine" vs. "a light core plus plugins and an ecosystem" — with concrete feature contrasts
- Know what MySQL 8.4 means as an LTS (Long-Term Support) release, and its support window
- Distinguish which parts of your Oracle experience carry over as-is and which must be unlearned
- Set up this book's lab environment (Docker MySQL 8.4)

Through an Oracle DBA's Eyes

The paths by which an Oracle DBA first ends up with MySQL are mostly alike. The company decides to build a new system on an open-source database for cost reasons; a cloud migration brings a managed service (AWS RDS and the like); or a merger suddenly drops a MySQL-based service into your care. More often than not, the situation chose you rather than the reverse — it did for this book's author.

The first impression usually comes in two stages. At first: “there is less here than I expected.” SQL runs, indexes exist, backups get taken. It looks like a scaled-down version of the job you did on Oracle. Then you start operating it, and the second stage arrives. Things you assumed would exist are not there (materialized views, synonyms, DB links); things you assumed would be the same behave differently (optimizer statistics, locking, NULL versus empty string); even the `ps` output looks wrong (chapter 6). What you need at that moment is not a memorized feature list but a **shift of perspective**. MySQL is not a “small Oracle” — it is a database that grew toward different goals in a different way. This chapter starts at the root of that difference: history and philosophy.

How MySQL Got Here

MySQL was born in 1995 at MySQL AB in Sweden. Co-founder Michael “Monty” Widenius named it after his daughter, My. The goal differed from Oracle’s from day one: not a complete enterprise engine, but **a database fast enough, simple enough, and free enough for web applications**. Through the 2000s it became the ‘M’ in the LAMP stack alongside Linux, Apache, and PHP — the default for web startups. Early Facebook, YouTube, and Wikipedia all started on MySQL.

The ownership story comes down to two acquisitions.

- In 2008, Sun Microsystems acquired MySQL AB for roughly one billion dollars.
- In 2010, Oracle acquired Sun, and **MySQL became Oracle’s.** The MySQL 8.4 we work with today is developed and shipped by Oracle.
- Right after the acquisition, Monty Widenius — worried MySQL would languish under Oracle — forked the source code and created **MariaDB**, named after his other daughter, Maria.

The terrain that history left looks like this. MySQL is distributed under a dual license — GPLv2 open source and a commercial license. The Community Edition we use is free; the Enterprise Edition, which adds features such as audit and firewall, is paid. Around it sits a compatibility ecosystem: MariaDB has since become a genuinely separate product on its own path (even the version numbers diverge), while Percona Server tracks MySQL compatibility and ships operational tooling (Percona Toolkit, XtraBackup). In the field, “the MySQL family” often means these three together — but **this book covers only Oracle’s MySQL Community Edition 8.4.**

It is worth seeing the market position plainly, too. MySQL has held second place behind Oracle in the DB-Engines ranking for years.[1] Developer preference, however, has shifted — in the Stack Overflow 2025 survey, PostgreSQL led at 55.6% usage with MySQL second at 40.5%.[2] PostgreSQL's growth in new projects is real. Yet in the installed base already in production, and in managed cloud services (AWS RDS and Aurora, GCP Cloud SQL, and others), MySQL's share remains large — and the demand for "a DBA who can properly run the MySQL that is already running" is as large as that installed base. That is precisely where this book aims.

> **TMI — where the two names come from.** The My in MySQL is not a possessive — it is the name of co-founder Monty Widenius's elder daughter, My. MariaDB, the fork he later created, carries the name of his younger daughter, Maria; and Sakila, the dolphin in the logo, was named through a public contest — the official sample database (sakila) inherited it. Oracle, meanwhile, took its name from the code name of a CIA project Larry Ellison had worked on before founding the company — at founding in 1977 the company was called Software Development Laboratories, and the product's name became the company's.

The Philosophical Difference — What Is Missing, and What Differs

Oracle has grown for over thirty years in one direction: "put everything a database needs inside the engine." Partitioning, materialized views, parallel query, RAC, Data Guard, Flashback, automated optimizer statistics management — the features live in the engine, and the DBA turns them on and tunes them.

MySQL's design points the other way. **Keep the core light, and solve the rest with architecture and the ecosystem.** The difference shows up not as a feature matrix but as a difference in how you operate.

- **Storage engines are separate**: MySQL splits the SQL-processing layer from the storage layer, and the storage layer is replaceable as a plugin (a storage engine). The de facto standard today is InnoDB, and this book covers only InnoDB — but the structure itself is the starting point for understanding MySQL (chapter 4).
- **What is missing is really missing**: there is no equivalent of materialized views, synonyms, or DB links. There are no sequences either — `AUTO_INCREMENT` stands in (chapter 20). Most queries execute on a single thread; there is no general-purpose parallel execution like Oracle's parallel query. When you need it, you solve it in the application and the design.

- **High availability is assembled**: not a finished, built-in HA product like RAC or Data Guard, but a construction — replication as the base part, with GTID, semi-synchronous replication, MySQL Router, and orchestration tools assembled on top (Part 6). Managed services (RDS Multi-AZ) are the cloud doing that assembly for you (Appendix).
- **Operational tooling lives in the ecosystem too**: instead of a diagnostic framework like Oracle's AWR/ASH, there is raw material — performance_schema — and the things that refine it are the sys schema, Percona Toolkit, and external monitoring (Part 5).

This is not an argument about which is better. Oracle's integration costs license fees and weight; MySQL's simplicity hands the responsibility for assembly and verification to the operator. The "emptiness" an Oracle DBA first feels in MySQL is not a defect but a design — and the remaining chapters of this book are about building Oracle-grade operational quality on top of that design.

Why 8.4 — the LTS Promise

A short word on versions. The release model is covered in detail in chapter 2; here is only why this book standardizes on 8.4.

In the MySQL 8.0 era, new features kept landing in 8.0.x minor versions. Behavior could change when all you did was apply a patch — a burden for any operation that values stability. In 2023 Oracle changed the release model, splitting **Innovation releases and LTS releases**. An LTS release takes no new features — only bug and security fixes — and is supported long-term.

- **MySQL 8.4**: the first LTS, released April 2024. Under the Oracle Lifetime Support policy, Premier support runs to April 2029 and Extended support to April 2032.[3]
- **MySQL 8.0**: support ended (EOL) on April 30, 2026. If you still run 8.0 in production, the 8.4 upgrade is a current task — chapter 54 covers it.[4]
- **MySQL 9.7**: the next LTS, released April 21, 2026.[5] Given how new it is and how conservative production tends to be, 8.4 will remain the mainstream in operations for years — and it is the default choice in managed services including AWS RDS.

In short: 8.4 is "the version you learn now and use the longest," the version the author actually operates, and the baseline against which every example in this book is verified.

How Far Does Oracle Experience Carry?

Sorting assets from liabilities at the start makes the learning faster.

What carries over as-is — SQL and relational modeling skill, the habit of thinking in indexes and execution plans, the conceptual framework of transactions, isolation levels, and backup/recovery, and above all the operational habit of driving from symptom to diagnosis to cause in an incident. These are why an Oracle DBA learns MySQL far faster than a newcomer.

What must be unlearned or rewritten — the following instincts actively get in the way in MySQL.

- "Look at the processes and you see the sessions." MySQL is a single process with many threads (chapter 6).
- "Optimizer statistics can be left to automatic management." MySQL's statistics are simpler than Oracle's and fragile under churn; neglected, your execution plans wobble (chapter 29). The author's 3,614 ms incident came from exactly this (chapter 74).
- "Instance and database are separate things." In MySQL, database is a synonym for schema, and the counterpart of Oracle's instance is the one mysqld server (chapter 3).
- "The engine will take care of it." As seen above, in MySQL the assembly and the verification are the DBA's job.

This book is designed along that list. Part 1 rebuilds the architectural instincts; Parts 2–3 rebuild SQL and tuning against the Oracle contrast; Parts 4–5 cover operations; Part 6 and the Appendix cover high availability and the cloud; Parts 7–9 cover security, operating at scale, and field cases. Every chapter's test cases were actually executed and verified on Docker MySQL 8.4.

Test Cases — Setting Up the Lab

We build the lab environment used throughout this book. Docker is all you need.

TC 1-1. Start the MySQL 8.4 container and confirm the connection.

```
        # If you already followed the Lab Environment Guide and book-mysql84 is up,
        # skip this step and just connect (on a name conflict: docker rm -f book-mysql84)
```

```
docker run -d --name book-mysql84 \
  -e MYSQL_ROOT_PASSWORD=book mysql:8.4

# Wait a few tens of seconds for startup, then
docker exec -it book-mysql84 mysql -uroot -pbook -e "SELECT VERSION();"
```

Measured output:

```
+-----------+
| VERSION() |
+-----------+
| 8.4.10    |
+-----------+
```

Now connect interactively — like sqlplus, the session **stays open once you are in**, and every ``` `sql block in this book runs inside this` mysql> ` prompt (see the Lab Environment Guide).

```
docker exec -it book-mysql84 mysql -uroot -pbook --default-character-set=utf8mb4
```

TC 1-2. First contact — what an Oracle DBA checks first.

```
-- The counterpart of Oracle's V$VERSION
SELECT VERSION();

-- The storage engine list — a concept Oracle does not have (chapter 4)
SHOW ENGINES;

-- The "database" list — closer to Oracle's schemas (chapter 3)
SHOW DATABASES;
```

In the measured `SHOW ENGINES` output, InnoDB appears as `DEFAULT` (Transactions/XA/Savepoints all YES), listed alongside MyISAM, MEMORY, CSV, ARCHIVE, and others. The initial `SHOW DATABASES` list is four entries — `information_schema`, `mysql`, `performance_schema`, `sys` — whose identities are covered in chapter 9.

Checkpoint

- Which company owns MySQL, and why did MariaDB come into being?
- Does MySQL have counterparts to Oracle's materialized views, DB links, and sequences? If not, what stands in for each?
- Can you explain why 8.4 is this book's baseline version, in terms of support windows and the release model?

Summary

- MySQL grew up in the web era aiming for "fast, simple, free," and has been owned and developed by Oracle since 2010
- Oracle's philosophy builds every feature into the engine; MySQL's assembles storage engines, replication, and ecosystem tools around a light core
- Materialized views, synonyms, DB links, and sequences do not exist — the "emptiness" is a design, not a defect
- 8.4 is the first LTS, supported to April 2032 (Extended); 8.0 support ended in April 2026
- SQL, modeling, and incident experience carry over as assets; process instincts, trust in automatic statistics management, and the instance vocabulary must be rewritten

Chapter 2. MySQL 8.4 Overview and the LTS Release Model

Learning Objectives

- Explain the difference between MySQL's Innovation and LTS releases, and each one's support window
- Place your own system on the version timeline running MySQL 8.0 → 8.4 → 9.7
- Identify the 8.4 changes that hit operations directly (authentication default, removed replication syntax, InnoDB defaults)

- Understand what a quarterly patch release contains, and build a judgment rule for applying patches
- Check the exact version and key defaults of a running server

Through an Oracle DBA's Eyes

This chapter's material is, in fact, a structure an Oracle DBA already knows. Oracle Database moved to annual releases in 2018 and split its versions in two — long-supported **Long Term Releases** (19c, 23ai) and **Innovation Releases** (21c) that ship features first but are supported briefly. Common sense was to keep production on 19c and use 21c only to evaluate new features.

The release model MySQL adopted in 2023 is exactly that structure. **MySQL's LTS corresponds to Oracle's Long Term Release, and MySQL's Innovation release to Oracle's 21c.** In other words, "8.4 is MySQL's 19c" — and with that, half this chapter is done. The other half is what, concretely, 8.4 changed relative to 8.0; that becomes the foundation of the upgrade checklist (chapter 54).

The Problem in the 8.0 Era — Patches That Were Not Patches

MySQL 8.0 went GA in April 2018 and lived as one version for nearly eight years. The problem was the delivery style: new features kept landing in 8.0.x **minor patches** (continuous delivery). Function-based indexes arrived in 8.0.13, CHECK constraints in 8.0.16,[6] and as features landed, default behavior could shift — or existing features could be deprecated — inside a minor patch.

In an Oracle DBA's terms: an environment where **applying what looked like a PSU/RU could change optimizer behavior.** The operating principle "security patches promptly, feature changes deliberately" simply could not be applied, and that became the background for the release-model overhaul.[7]

Innovation and LTS — the New Release Model

In July 2023, starting with MySQL 8.1, releases split into two tracks.[8]

- **Innovation releases**: quarterly (8.1, 8.2, 8.3, then 9.0–9.6), carrying new features. Production-grade in quality, but **supported only until the next release (about three months)** — hard to run in production unless your organization tracks the latest continuously.

- **LTS releases**: roughly every two years. No new features afterward — bug and security fixes only — with **Premier 5 years + Extended 3 years = 8 years** of support under the Oracle Lifetime Support policy.

As a timeline:

Version	Character	GA	End of support
8.0	old model (continuous delivery)	2018-04	**2026-04-30 (EOL reached)**[9]
8.1 – 8.3	Innovation	2023-07 – 2024-01	each until the next release
8.4	**first LTS**	**2024-04-30**	Premier 2029-04 / Extended 2032-04[10]
9.0 – 9.6	Innovation	2024-07 –	each until the next release
9.7	second LTS	2026-04-21[11]	GA + 8 years per policy

Set alongside Oracle Database, the mapping is crisp. If 19c (whose Premier support was extended to the end of 2029[12]) is 8.4, then 21c is the 9.0–9.6 Innovation line, and 23ai is 9.7. If your production database is on 8.0 as you read this, that is the Oracle equivalent of running out-of-support 11.2 — chapter 54's upgrade procedure is your top priority.

> **TMI — why 8.0 follows 5.7.** Version 6.0 was abandoned in the late 2000s and became a skipped number, and the 7.x range was already taken by MySQL Cluster (NDB). So the major after 5.7 became 8.0 — a scar of the vendor's history left in the version numbers.

> **TMI — Oracle bought MySQL's heart first.** Oracle got MySQL through the Sun acquisition in 2010 — but five years earlier, in 2005, it had already acquired Innobase, the maker of InnoDB. In 2008 Sun bought MySQL AB for about a billion dollars; when Oracle's acquisition of Sun was announced in 2009, Monty forked MariaDB the same year. The deal closed in 2010 after EU competition review — today's terrain (Oracle's MySQL, the community's MariaDB) is the product of those three years.

Patches ship quarterly on both tracks — the same quarterly rhythm as Oracle's CPU (Critical Patch Update), so it will feel familiar. The 8.4 line, for example, has run 8.4.8 (2026-01-20), 8.4.9 (2026-04-21), 8.4.10 (2026-06-16).[13] LTS patches carry no feature changes, so "patch = security and bug fixes" holds as an equation. Patch decisions are simpler than in the 8.0 era — which does not mean the verification step can be skipped.

What Changed in 8.4 — the Operator's Core

The full list of changes against 8.0 is in the official document "What Is New in MySQL 8.4."[14] Here we pick only what operations actually runs into.

Authentication: mysql_native_password disabled by default

From 8.4 the legacy authentication plugin `mysql_native_password` is **disabled by default**, and in 9.0 it was removed entirely. The default authentication has been `caching_sha2_password` since 8.0. If old application drivers (legacy JDBC, PHP clients, and the like) were connecting with native password, logins fail right after an 8.4 upgrade. The server option `--mysql-native-password=ON` buys temporary life, but the feature's removal is announced — migrating the accounts is the real answer (chapter 35).

Replication: MASTER/SLAVE syntax removed

The old replication syntax that coexisted as deprecated in 8.0 is **removed** in 8.4. Running it is now a syntax error.

Removed syntax	Use in 8.4
`CHANGE MASTER TO ...`	`CHANGE REPLICATION SOURCE TO ...`
`START SLAVE` / `STOP SLAVE`	`START REPLICA` / `STOP REPLICA`
`SHOW SLAVE STATUS`	`SHOW REPLICA STATUS`

Not just the statements — the related status variables and field names moved to the SOURCE/REPLICA family too. The danger is less the database than **the periphery**: years-old operational runbooks, monitoring

scripts, and failover automation that scrape `SHOW SLAVE STATUS` break — quietly or loudly — the moment you upgrade. When this book reaches replication in Part 6, it uses the new syntax exclusively from the start.

InnoDB defaults modernized

8.4 changed the defaults of twenty InnoDB system variables. The gist: "conservative defaults sized for 2000s hardware, raised for the SSD and multi-core era."[15] The ones with real operational impact:

Variable	8.0 default	8.4 default
`innodb_io_capacity`	200	10000
`innodb_flush_method`	fsync (Linux)	O_DIRECT (Linux)
`innodb_log_buffer_size`	16MB	64MB
`innodb_adaptive_hash_index`	ON	OFF
`innodb_purge_threads`	4	1 (with ≤16 logical CPUs)

Note that this cuts both ways. A system that relied on defaults — nothing in my.cnf — gets **different I/O and flush behavior** after the upgrade. Conversely, if you tuned these explicitly on 8.0, my.cnf wins and nothing changes. Knowing which side you are on is where upgrade preparation starts (chapter 54). The variables themselves are covered in chapters 4–5 and 53.

New in replication: tagged GTIDs

The GTID format extends from `UUID:NUMBER` to `UUID:TAG:NUMBER`, letting you name a group of transactions (`SET gtid_next = 'AUTOMATIC:tag'` ; the tag is letters, digits, and underscores, up to 32 characters; requires the dedicated `TRANSACTION_GTID_TAG` privilege).[16] For example, the transactions of a bulk batch or a maintenance job can be distinguished from regular traffic by GTID alone. The existing `UUID:NUMBER` format remains compatible. If GTIDs are new to you, chapter 56 starts from the basics.

This Book's Baseline Version

Put together, the reason this book standardizes on 8.4 falls out: 8.0 is out of support, Innovation releases have support windows unfit for production, and 9.7 has only just shipped. 8.4 is the LTS with support locked in to 2032 and the widest installed base today. Every example in the chapters ahead is verified on Docker's `mysql:8.4` image (always the latest 8.4.x patch), with the exact patch version recorded in each verification comment.

Test Cases

TC 2-1. Confirm the exact version and license.

```
SELECT VERSION();
SHOW GLOBAL VARIABLES LIKE 'version%';
-- Measured: version = 8.4.10, version_comment = 'MySQL Community Server - GPL'
```

TC 2-2. Authentication plugin state — is mysql_native_password disabled?

```
SELECT plugin_name, plugin_status
  FROM information_schema.plugins
 WHERE plugin_name LIKE '%password%';
```

Measured output:

```
+-----------------------+---------------+
| plugin_name           | plugin_status |
+-----------------------+---------------+
| sha256_password       | ACTIVE        |
| caching_sha2_password | ACTIVE        |
| mysql_native_password | DISABLED      |
+-----------------------+---------------+
```

TC 2-3. Confirm the old replication syntax really errors out.

```
        SHOW SLAVE STATUS;
        -- Measured: ERROR 1064 (42000): You have an error in your SQL syntax; ... near
'SLAVE STATUS'
        SHOW REPLICA STATUS;
        -- Measured: runs normally (Empty set — replication not configured)
```

TC 2-4. Check the 8.4 defaults — the starting point of the default-reliance audit.

```
        SELECT @@innodb_io_capacity, @@innodb_flush_method,
               @@innodb_log_buffer_size / 1024 / 1024 AS log_buffer_mb,
               @@innodb_adaptive_hash_index, @@innodb_purge_threads;
```

Measured output — the 8.4 default changes discussed in the body, confirmed as-is:

```
+----------------------+-----------------------+---------------
+------------------------------+-----------------------+
| @@innodb_io_capacity | @@innodb_flush_method | log_buffer_mb |
@@innodb_adaptive_hash_index | @@innodb_purge_threads |
+----------------------+-----------------------+---------------
+------------------------------+-----------------------+
|                10000 | O_DIRECT              |   64.00000000
|                             0 |                       1 |
+----------------------+-----------------------+---------------
+------------------------------+-----------------------+
```

Checkpoint

- Can you map MySQL's Innovation and LTS releases onto Oracle Database's release scheme?
- What is the classic reason application connections break on an 8.0 → 8.4 upgrade, and how do you audit for it in advance?
- For a system that never set InnoDB parameters in my.cnf, which areas of behavior change on 8.4?

Summary

- Since 2023 MySQL splits releases into Innovation (quarterly, short support) and LTS (about every two years, 5+3-year support) — the same structure as Oracle's Long Term/Innovation releases
- 8.4 is the first LTS (GA 2024-04, Extended support to 2032-04); 8.0 support ended 2026-04-30; 9.7 is the next LTS
- 8.4's three big operational impacts: mysql_native_password disabled by default, MASTER/SLAVE replication syntax removed, twenty InnoDB defaults modernized
- The removed replication syntax breaks the periphery first — runbooks, scripts, monitoring — before it breaks the database
- LTS patches carry only bug and security fixes, so patch decisions are simple — keep the verification step anyway

Chapter 3. MySQL Server Architecture — Compared with an Oracle Instance

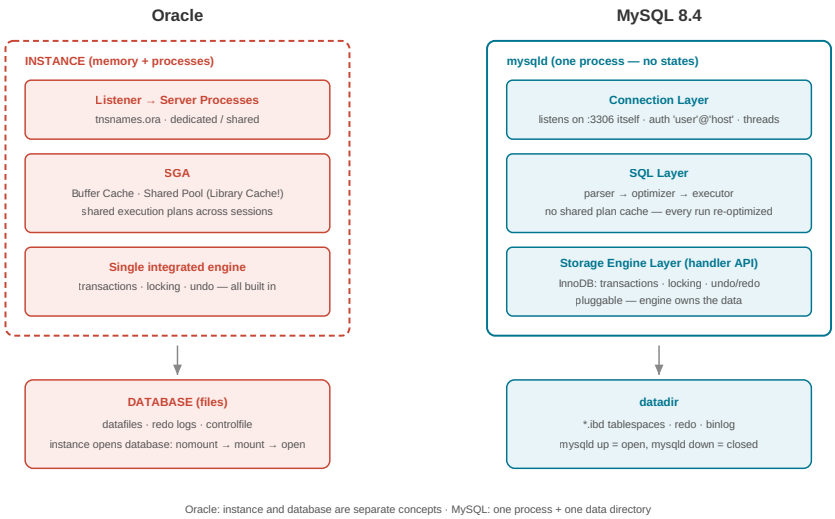
Learning Objectives

- Explain how Oracle's "instance + database" distinction is simplified in MySQL
- Explain the three layers of the MySQL server (connection layer, SQL layer, storage engine layer) and the role of each
- Understand the decisive difference between Oracle and MySQL from the standpoint of execution-plan sharing (the library cache)
- Explain the difference between Oracle's user-as-schema-owner model and MySQL's separation of accounts and databases
- Distinguish the purposes of the four system databases (mysql, information_schema, performance_schema, sys)

Through an Oracle DBA's Eyes

The first sentence you memorize when learning Oracle is "the instance and the database are different things." The instance is the SGA plus the set of background processes; the database is the files on disk. The instance walks through nomount → mount → open to open the database, and RAC is explained on top of that distinction as "multiple instances opening one database."

MySQL has no such distinction. **One mysqld process and one data directory (datadir)** are the whole story, with no state stages in between. When mysqld is up, it is open; when it dies, it is closed. At first glance there seems to be less to learn, but in practice the confusion is worse, because familiar terms are reused with subtly different meanings — MySQL's "database" is not an Oracle database but a schema, and MySQL's "user" is not a schema owner. This chapter redraws that map of terms.



The MySQL server's three-layer structure contrasted with the Oracle instance structure

The Disappearance of the Instance Concept

Start by sorting out where the components an Oracle-trained mind expects have gone in MySQL.

Oracle	MySQL 8.4	Notes
Instance (SGA + processes)	one mysqld process	no state stages (nomount/mount/open)
Database (set of files)	data directory (datadir)	detailed in ch. 5 and 10
spfile / pfile	my.cnf (+ SET PERSIST)	detailed in ch. 33
alert log	error log	location in the `log_error` variable
SID / service name	none — host:port is the identifier	default 3306
tnsnames.ora / listener	none — clients connect directly	there is no separate listener process

The absence of a listener feels strange at first. In Oracle, the listener accepts a connection and hands it to a server process; in MySQL, mysqld itself opens port 3306 directly and creates a thread per connection (ch. 6). The connection string is nothing more than `host + port + account`, so there is no name-resolution layer like tnsnames.

Startup and shutdown are simple too. When mysqld starts, InnoDB examines the redo log and performs crash recovery automatically if needed (ch. 8), then begins accepting connections. There is a process corresponding to Oracle's "instance recovery," but the DBA has no stages to manipulate. In an RDS environment, startup and shutdown themselves disappear behind the console (appendix).

Inside the Server — the Three-Layer Structure

The inside of mysqld is best understood as three broad layers.

1) Connection layer — accepts connections, authenticates them (at the `'user'@'host'` granularity, ch. 35), and assigns a dedicated thread to each connection. Everything Oracle does with the listener plus dedicated server process creation happens inside this one layer.

2) SQL layer — the parser turns the statement into a parse tree, the optimizer builds an execution plan, and the executor carries out the plan by calling into the storage engine. This corresponds to what happened in Oracle's shared pool — with one decisive difference, covered in the next section.

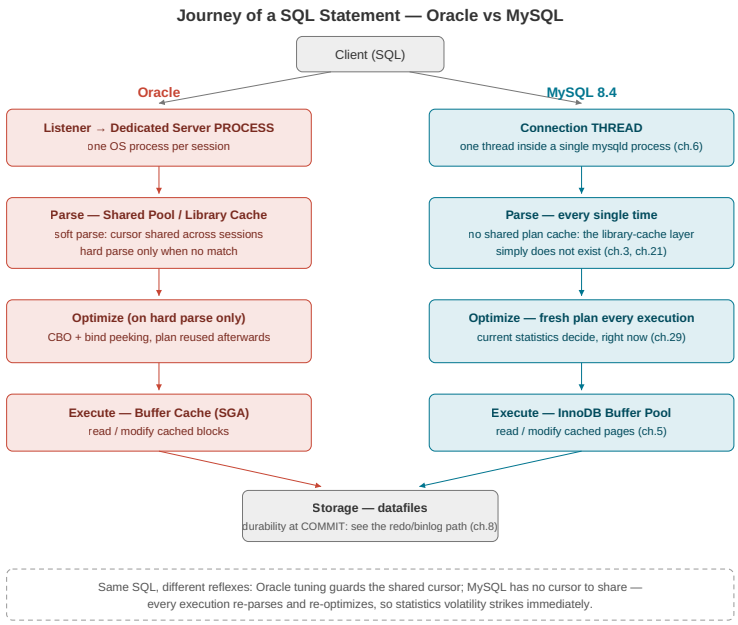
3) Storage engine layer — responsible for storing and retrieving data, transactions, and locking. It is separated from the SQL layer by an interface called the handler API, so engines can be swapped in like plugins. The de facto standard is InnoDB (ch. 4). Oracle has no such separation at all — the buffer cache, undo, and locking are all parts of a single engine. The fact that in MySQL "transactions and locking are features of the **engine**, not the server" is a premise for every chapter that follows.

For reference, the query cache that older material mentions so often was removed entirely in 8.0 because of concurrency bottlenecks. It does not exist in 8.4.

There Is No Library Cache — a Database That Does Not Share Plans

A large part of Oracle tuning revolves around the shared pool. Identical SQL text shares a cursor in the library cache; bind variables are enforced to reduce hard parsing; the cursor cache and plan baselines pin plans down.

MySQL **does not have this layer at all.** Ordinary SQL is **parsed anew and optimized anew on every execution** — across sessions, and even for the same statement repeated in the same session. There is no shared plan cache.[17] Prepared statements let the parse result be reused, but their scope is the session — other sessions cannot access them.[18]



The journey of one SQL statement — Oracle and MySQL diverge at each stage: connection, parsing, optimization, execution. MySQL is missing the shared-plan-cache layer entirely

This difference carries gains and losses together.

- **The good side**: no hard-parse storms, no cursor contention (library cache latches), no bind-variable obsession. MySQL's parsing and optimization are designed to be correspondingly light, and literal SQL is not as deadly as it is in Oracle.
- **The bad side**: there is also no mechanism to store and pin a plan. Nothing corresponds to plan baselines or SQL profiles, so **when the statistics wobble, the plan changes on the spot** — because every execution is an optimization. In Oracle, the problem was a bad plan getting "pinned"; in MySQL, the problem is that plans are "not pinned" at all. This is one of the central themes of this book (ch. 29–30), and the root of the 3,614 ms incident the author lived through (ch. 74).

Users and Schemas — There Is No Concept of Ownership

In Oracle, creating a user creates a schema of the same name; tables that user creates belong to that schema, and ownership follows. The very name `SCOTT.EMP` says "whose what."

In MySQL the two axes are completely separate.

- **Database (= schema)**: nothing more than a namespace that holds tables. `CREATE DATABASE` and `CREATE SCHEMA` are exact synonyms. It is owned by no one.
- **Account**: the combination of **user name + connecting host**, as in `'appuser'@'10.0.%'`, is one account. The same user name with a different host is a different account, with its own separate privileges (ch. 34).

In other words, MySQL has no "owner of this table." What exists is only "the privileges this account holds on this database/table" (the DEFINER of a stored program plays an exceptional owner-like role, covered in ch. 37). The "schema per account" design an Oracle DBA reaches for by habit translates in MySQL to "separate databases + granted privileges."

One more consequence: since a database is just a namespace, services that would have been split into separate DB instances in Oracle commonly cohabit as multiple databases on one MySQL server. Cross-database joins (`db1.t JOIN db2.u`) are everyday usage — not something that would need an Oracle DB link, just a namespace reference within the same server.

When a genuinely **cross-server** reference is needed, be honest about it — there is no standard equivalent of a DB link's remote join. The FEDERATED engine formally exists, but it is disabled by default and not recommended in practice (ch. 4). The realistic responses are three: bring the data over with replication and join locally (ch. 55), move it with ETL, or join at the application layer. Letting go of the very idea of "transparently reaching a remote database from inside the database" is where porting begins.

The Four System Databases

The four entries you see when you run `SHOW DATABASES` on a new server correspond to Oracle's dictionary and dynamic performance view system. Details come in chapter 9; here we only draw the map.

MySQL	Contents	Oracle counterpart, roughly
`mysql`	the actual server operating data — accounts, privileges, time zones	the base tables under the SYS schema
`information_schema`	standards-based metadata views	DBA_* / ALL_* views
`performance_schema`	runtime instrumentation data (waits, threads, statement statistics)	V$ views + the raw material of ASH/AWR
`sys`	views that reshape performance_schema for human reading	summary views close to an AWR report

Since 8.0, the data dictionary itself is stored transactionally in InnoDB tables (the old .frm files are gone). Atomic DDL is a product of this same change — chapter 10 covers it together with the file structure.

Test Cases

TC 3-1. Confirm that CREATE DATABASE and CREATE SCHEMA are synonyms.

```
CREATE DATABASE d1;
CREATE SCHEMA d2;
SHOW CREATE DATABASE d1;
SHOW CREATE DATABASE d2;
DROP DATABASE d1; DROP DATABASE d2;
```

Measured: the two results have exactly the same form — d2, created with SCHEMA, is also displayed as `CREATE DATABASE `d2` /*!40100 DEFAULT CHARACTER SET utf8mb4 ... */`. The two are literally synonyms.

TC 3-2. Confirm that an account is a user-name + host combination.

```
CREATE USER 'app'@'localhost' IDENTIFIED BY 'Pass_1234';
CREATE USER 'app'@'10.0.0.%'  IDENTIFIED BY 'Pass_5678';
SELECT user, host FROM mysql.user WHERE user = 'app';
-- Measured: (app, 10.0.0.%) (app, localhost) — same user, two independent accounts
```

```
        DROP USER 'app'@'localhost'; DROP USER 'app'@'10.0.0.%';
```

TC 3-3. Confirm that a cross-database join is ordinary syntax.

```
        CREATE DATABASE sales; CREATE DATABASE ref;
        CREATE TABLE sales.orders (id INT PRIMARY KEY, region_id INT);
        CREATE TABLE ref.regions  (id INT PRIMARY KEY, name VARCHAR(20));
        INSERT INTO sales.orders VALUES (1, 10);
        INSERT INTO ref.regions  VALUES (10, 'Seoul');
        SELECT o.id, r.name
          FROM sales.orders o JOIN ref.regions r ON r.id = o.region_id;
        -- Measured: (1, Seoul) — joined immediately, no DB link required
        DROP DATABASE sales; DROP DATABASE ref;
```

TC 3-4. Check the system databases and basic server information.

```
        SHOW DATABASES;
        -- Measured: information_schema, mysql, performance_schema, sys (+ user DBs)
        SELECT @@datadir, @@log_error, @@port;
        -- Measured: /var/lib/mysql/ | stderr | 3306  (container context — log_error being

        --       stderr is the Docker log-collection convention. A normal install shows a
file path)
```

Checkpoint

- Can you explain what Oracle's instance/database distinction simplifies down to in MySQL?
- Can you state two implications — one good, one bad — that "MySQL does not share execution plans" has for tuning strategy?
- Are `'batch'@'10.1.%'` and `'batch'@'localhost'` the same account? How are their privileges managed?

Summary

- MySQL has no instance/database distinction — the mysqld process + datadir are everything, and there is no listener
- The server has three layers — connection layer / SQL layer / storage engine layer — and transactions and locking are features of the engine (InnoDB), not the server
- There is no shared plan cache (library cache), so every execution is an optimization — no parse burden, but no plan-pinning mechanism either
- A MySQL database is a namespace synonymous with a schema, an account is a `'user'@'host'` combination, and there is no object-owner concept
- Four system DBs: mysql (the substance), information_schema (metadata), performance_schema (instrumentation), sys (summaries)

---

1. DB-Engines Ranking, https://db-engines.com/en/ranking (as of 2026: Oracle #1, MySQL #2).↩
2. Stack Overflow Developer Survey 2025 — Technology, https://survey.stackoverflow.co/2025/technology↩
3. MySQL 8.4 Release Notes (GA 2024-04-30), https://dev.mysql.com/doc/relnotes/mysql/8.4/en/ and the Oracle Lifetime Support Policy. For a support-window summary see https://endoflife.date/mysql.↩
4. MySQL Product Support EOL Announcements, https://www.mysql.com/support/eol-notice.html↩
5. MySQL 9.7 Release Notes — Changes in MySQL 9.7.0 (2026-04-21), https://dev.mysql.com/doc/relnotes/mysql/9.7/en/news-9-7-0.html↩
6. MySQL 8.0 Release Notes — Changes in MySQL 8.0.16 (CHECK constraints enforced), https://dev.mysql.com/doc/relnotes/mysql/8.0/en/news-8-0-16.html↩
7. Introducing MySQL Innovation and Long-Term Support (LTS) versions, https://dev.mysql.com/blog-archive/introducing-mysql-innovation-and-long-term-support-lts-versions/↩
8. Introducing MySQL Innovation and Long-Term Support (LTS) versions, https://dev.mysql.com/blog-archive/introducing-mysql-innovation-and-long-term-support-lts-versions/↩

9. MySQL Product Support EOL Announcements, https://www.mysql.com/support/eol-notice.html↩

10. MySQL 8.4 Release Notes, https://dev.mysql.com/doc/relnotes/mysql/8.4/en/ and https://endoflife.date/mysql↩

11. MySQL 9.7 Release Notes — Changes in MySQL 9.7.0 (2026-04-21), https://dev.mysql.com/doc/relnotes/mysql/9.7/en/news-9-7-0.html↩

12. Oracle Database 19c Premier support extended to 2029-12-31, Extended to 2032-12-31 (announced 2025-02), https://www.theregister.com/2025/02/18/oracle_extends_19c_support/↩

13. MySQL 8.4 Release Notes, per-version entries, https://dev.mysql.com/doc/relnotes/mysql/8.4/en/↩

14. What Is New in MySQL 8.4 since MySQL 8.0, https://dev.mysql.com/doc/refman/8.4/en/mysql-nutshell.html↩

15. MySQL 8.4 LTS — new production-ready defaults for InnoDB, https://lefred.be/content/mysql-8-4-lts-new-production-ready-defaults-for-innodb/↩

16. MySQL 8.4 Reference Manual — GTID Format and Storage, https://dev.mysql.com/doc/refman/8.4/en/replication-gtids-concepts.html↩

17. MySQL has no execution-plan cache shared across statements. Markus Winand, “Planning For Reuse”, https://use-the-index-luke.com/blog/2011-07-16/planning-for-reuse (a comparison of plan caches across databases)↩

18. MySQL 8.4 Reference Manual — Caching of Prepared Statements and Stored Programs: “statements cached for one session are not accessible to other sessions”, https://dev.mysql.com/doc/refman/8.4/en/statement-caching.html↩