Best Practices in MySQL · InnoDB

8 rules for lookups by ID at huge scale. The table file is the index. There are two logs, not one.

1 Two logs, two different jobs

Redo rebuilds pages after a crash. The binlog feeds replicas and point-in-time recovery. Losing either one breaks a different promise.

Deep dive ↓ Level 0 · Redo + binlog two-phase commit

ONE COMMIT, WRITTEN TWICE: ON PURPOSE

COMMIT → `#ib_redo0 · redo · physical page changes`

`mysql-bin.000042 · logical row changes`

Crash recovery replays **redo**. A replica replays the **binlog**. Two-phase commit is what keeps the two in agreement.

2 The table file IS the primary-key index

Rows live inside users.ibd in primary-key order. A lookup by ID is a walk down the file's own B+ tree, there is no separate heap.

Deep dive ↓ Level 2 · Change buffer & clustered index

SHOP/USERS.IBD: 16 KB PAGES, KEYS IN ORDER

`root page · keys 1…1M` → `internal page` →

`leaf page = the row itself ✓`

Consequence: a secondary index stores the **primary key**, not a row pointer, so a wide PK is paid for in every index you add.

3 The buffer pool is the database

Almost every read is served from the InnoDB buffer pool. Size it to the working set, typically 70-80% of RAM on a dedicated server.

Deep dive ↓ Level 1 · Buffer pool & checkpoint age

WHERE A READ ACTUALLY GOES

`buffer pool hit · ~0.1 ms`

`miss → read 16 KB page from .ibd · 10-100× slower`

Watch it: **Innodb_buffer_pool_reads ÷ read_requests**, physical reads should stay a small fraction of logical ones.

4 Checkpoint age near capacity = writes stall

Redo is a ring. When uncheckpointed changes fill it, InnoDB stops accepting writes and flushes pages until there is room.

Deep dive ↓ Level 1 · Redo capacity & checkpoint age

CHECKPOINT AGE, AS A SHARE OF REDO CAPACITY

`age < 50% · writes flow ✓`

`age → 100% · forced flush, write stalls ✗`

```
SHOW ENGINE INNODB STATUS\G  ← LOG section: "Log sequence number"
```

5 History list length is cleanup debt, in one number

Old versions live in undo until purge deletes them. A long-running transaction pins them all, and every read walks a longer version chain.

Deep dive ↓ Level 3 · Undo, purge & the history list

ONE ROW, THREE VERSIONS DEEP

`users.ibd · current row` → `undo_001 · previous version` →

`undo_002 · older still · purge blocked ✗`

History list length in SHOW ENGINE INNODB STATUS: hundreds is normal, **millions is an incident**, find the oldest open transaction.

6 The change buffer defers secondary-index writes

Inserts into a secondary index whose page is not in memory are recorded in the change buffer and merged later, writes get faster, cold reads pay the merge.

Deep dive ↓ Level 2 · Change buffer & adaptive hash

WRITE NOW, MERGE WHEN THE PAGE IS NEXT READ

`INSERT` → `change buffer · "merge later"` →

`page read → merge applied → index current`

Wins on **write-heavy, non-unique** secondary indexes. It never applies to the clustered index or unique indexes, uniqueness must be checked now, not later.

7 Binlog retention must beat your longest outage

A replica that is down longer than the binlog is kept cannot catch up, it must be rebuilt from a backup. Retention is a recovery decision.

Deep dive ↓ The log files, one by one

THE RACE EVERY REPLICA RUNS

`replica down · day 0` → `binlog kept 7 days` →

`back by day 6 → replays & catches up ✓`

`back day 9 → binlog gone · full rebuild ✗`

```
SHOW BINARY LOGS;  ← oldest file's age vs your longest maintenanc
```

8 GTIDs make failover position exact

Every transaction gets a global ID. A promoted replica knows precisely what it has applied, no guessing file names and byte positions at 3 AM.

Deep dive ↓ Level 4 · GTID & semi-sync replication

FINDING POSITION AFTER A FAILOVER

FILE / POSITION

× "mysql-bin.000042, position... 819203? 819230?", one wrong byte and the replica applies a transaction twice or skips it.

GTID

✓ Executed set **3E11FA47-...:1-10482**, the new primary asks for exactly the GTIDs it is missing. Nothing else is ambiguous.