MySQL / MariaDB Troubleshooting Runbook Pack

A senior engineer's incident runbook for the failures that page you at 2am — connections, locks, replication, disk, and data errors. DevOps AI ToolKit · devopsaitoolkit.com

When a MySQL or MariaDB instance misbehaves, the application errors are downstream symptoms — the truth is in `SHOW PROCESSLIST`, `SHOW ENGINE INNODB STATUS`, the global status counters, and the error log. This pack walks the ten failure modes that cause most database incidents. Work read-only first: look before you `KILL`, and never run a blocking `OPTIMIZE` or `ALTER` on a hot primary without a change record. Examples assume a host `db-1:3306` and an application schema `appdb`.

> MySQL 8.0 and MariaDB 10.x diverge on several commands. The biggest one: replication is `SHOW REPLICA STATUS` on MySQL 8.0.22+ but `SHOW SLAVE STATUS` on MariaDB (and older MySQL). Where they differ, both are shown. Run diagnostics as a read-only admin user; take a change record before any `KILL`, `OPTIMIZE`, or config change on production.

1. First-Response Triage

Start here for *any* database incident. These four reads tell you whether the server is up, saturated, blocked, or corrupt — before you touch anything.

```
-- Who is connected and what are they doing (right now)?
SHOW FULL PROCESSLIST;

-- The single most useful command: locks, deadlocks, I/O, buffer pool, pending work
SHOW ENGINE INNODB STATUS\G

-- Saturation and error counters
SHOW GLOBAL STATUS LIKE 'Threads_connected';
SHOW GLOBAL STATUS LIKE 'Threads_running';
SHOW GLOBAL STATUS LIKE 'Aborted_%';
SHOW GLOBAL STATUS LIKE 'Slow_queries';
SHOW GLOBAL VARIABLES LIKE 'max_connections';
SHOW GLOBAL VARIABLES LIKE 'read_only';   -- accidental read_only = write failures
```

```
# The error log names the real cause (crash, corruption, OOM, aborted startup)
sudo tail -n 100 /var/log/mysql/error.log        # Debian/Ubuntu MySQL
sudo tail -n 100 /var/log/mariadb/mariadb.log     # MariaDB RPM default
sudo journalctl -u mysql -u mariadb --no-pager | tail -n 60
```

- `Threads_running` climbing while `Threads_connected` is flat → queries are piling up behind a lock or a slow query, not a connection flood.
- Lots of `Aborted_connects` → auth failures or network resets (§5, §9).
- Look at `PROCESSLIST` **State** column: `Waiting for table metadata lock`, `Sending data`, `copy to tmp table`, and `statistics` each point at a different problem (§3, §4).

2. Too Many Connections (ER 1040)

`ERROR 1040 (HY000): Too many connections` means every slot up to `max_connections` is in use. Usually the app opened connections faster than it closed them, or long-running queries are holding slots. There is always one reserved slot for `SUPER`/`CONNECTION_ADMIN` — connect with an admin account.

```
-- Connect via the reserved super slot, then look
SHOW GLOBAL STATUS LIKE 'Threads_connected';
SHOW GLOBAL STATUS LIKE 'Max_used_connections';   -- high-water mark
SHOW GLOBAL VARIABLES LIKE 'max_connections';
SHOW GLOBAL VARIABLES LIKE 'wait_timeout';        -- idle conns held this long (default 28800s)

-- Who is hogging slots? Group by host/user to find the culprit app node
SELECT user, host, COUNT(*) AS conns
FROM information_schema.processlist
GROUP BY user, host ORDER BY conns DESC;

-- Idle 'Sleep' connections eating slots
SELECT COUNT(*) FROM information_schema.processlist WHERE command = 'Sleep';
```

- Many `Sleep` connections with a long `wait_timeout` → the app's connection pool is oversized or leaking. Lower `wait_timeout`, or fix the pool.
- `Max_used_connections` == `max_connections` → you genuinely hit the ceiling. Raising it is valid *if you have RAM* (each connection costs buffers), but first confirm it is not a leak.

```
-- Emergency headroom (dynamic, no restart). Set a considered value, not infinity.
SET GLOBAL max_connections = 500;
-- Kill a specific runaway (read the query FIRST):
-- KILL 12345;    -- terminates connection+query; KILL QUERY 12345 ends only the statement
```

> Raising `max_connections` without RAM headroom trades "too many connections" for an OOM kill. Fix the pool leak; treat the bump as a stopgap.

3. Locks & Deadlocks (1205 / 1213)

Two distinct errors: `ERROR 1205 (HY000): Lock wait timeout exceeded` (a transaction waited too long for a row lock held by another) and `ERROR 1213 (40001): Deadlock found` (two transactions each hold a lock the other needs — InnoDB auto-rolls back the cheaper one).

```
-- The authoritative deadlock report — read the LATEST DETECTED DEADLOCK section
SHOW ENGINE INNODB STATUS\G

-- Who is blocking whom (MySQL 8.0 / Performance Schema)
SELECT * FROM performance_schema.data_lock_waits\G
SELECT thread_id, object_name, lock_type, lock_mode, lock_status
FROM performance_schema.data_locks\G

-- MySQL 5.7 / older equivalent (needs innodb_lock_wait plugin data)
SELECT * FROM information_schema.innodb_lock_waits;
SELECT * FROM information_schema.innodb_trx ORDER BY trx_started;   -- oldest = suspect

SHOW GLOBAL VARIABLES LIKE 'innodb_lock_wait_timeout';   -- default 50s
```

- `LATEST DETECTED DEADLOCK` in `INNODB STATUS` shows both transactions, the exact SQL, and which one was rolled back — it is the ground truth. Note that it shows only the *most recent* deadlock.
- A long-running row in `innodb_trx` with `trx_state = LOCK WAIT` and a large `trx_rows_locked` is your blocker. Kill *the blocker*, not the victim.
- Deadlocks are usually an app bug: inconsistent lock ordering, or too-large transactions. The fix is to order statements consistently and keep transactions short — retry logic on 1213 is expected, not a workaround.

4. Slow Queries & Missing Indexes

Slowness shows up as high `Threads_running`, `Sending data` / `copy to tmp table` states, and CPU or I/O saturation. Confirm with the slow query log, then use `EXPLAIN` to find the missing index.

```
-- Turn on slow log at runtime (no restart); capture anything over 1s
SET GLOBAL slow_query_log = 'ON';
SET GLOBAL long_query_time = 1;
SHOW GLOBAL VARIABLES LIKE 'slow_query_log_file';

-- Queries doing full scans / no index (catches the worst offenders)
SET GLOBAL log_queries_not_using_indexes = 'ON';

-- Are we spilling to disk temp tables (a scan/sort smell)?
SHOW GLOBAL STATUS LIKE 'Created_tmp_disk_tables';
SHOW GLOBAL STATUS LIKE 'Created_tmp_tables';
```

```
-- Diagnose the actual plan. Look for type=ALL and a NULL possible_keys.
EXPLAIN SELECT * FROM appdb.orders WHERE customer_id = 42 AND status = 'open';

-- MySQL 8.0 / MariaDB 10.1+: real execution costs, not estimates
EXPLAIN ANALYZE SELECT * FROM appdb.orders WHERE customer_id = 42;
```

- `type: ALL` with `possible_keys: NULL` and a large `rows` → full table scan. Add a covering index on the filter/join columns: `ALTER TABLE appdb.orders ADD INDEX idx_cust_status (customer_id, status);`.
- `Using filesort` / `Using temporary` in Extra → the sort or group has no supporting index.
- Rising `Created_tmp_disk_tables` → queries exceed `tmp_table_size` / `max_heap_table_size` and spill to disk. Fix the query first; raise the buffers only if the query is already lean.

> `ALTER TABLE … ADD INDEX` on a large hot table is online in InnoDB (8.0 / MariaDB 10.x) but still consumes I/O and can block DDL behind metadata locks. Schedule it, or use a percona-online-schema-change / gh-ost tool on a busy primary.

5. "Server Has Gone Away" (2006 / 2013)

`ERROR 2006 (HY000): MySQL server has gone away` (connection lost mid-query) and `ERROR 2013: Lost connection during query`. Three usual causes: the server killed an idle/slow connection, the query exceeded `max_allowed_packet`, or the network/proxy dropped it. It is often a *client-side* symptom of a *server-side* limit.

```
-- Packet size: a single row/blob bigger than this kills the connection
SHOW GLOBAL VARIABLES LIKE 'max_allowed_packet';   -- default 64M (8.0), 16M (MariaDB)

-- Idle timeouts that silently drop pooled connections
SHOW GLOBAL VARIABLES LIKE 'wait_timeout';
SHOW GLOBAL VARIABLES LIKE 'interactive_timeout';
SHOW GLOBAL VARIABLES LIKE 'net_read_timeout';
SHOW GLOBAL VARIABLES LIKE 'net_write_timeout';

-- Did the server itself restart / OOM (vs. just the connection)?
SHOW GLOBAL STATUS LIKE 'Uptime';                  -- small = recent crash/restart
SHOW GLOBAL STATUS LIKE 'Aborted_clients';         -- clients dropped without closing
```

- `Uptime` is low and you did not restart it → the server crashed or was OOM-killed. Check `error.log` and `dmesg | grep -i oom` (§7).
- Big `INSERT` / `LOAD` failing on 2006 → raise `max_allowed_packet` (dynamic on 8.0: `SET GLOBAL max_allowed_packet = 268435456;`) but the client must reconnect to pick up the session value.
- Failures only on long-idle pooled connections → `wait_timeout` is shorter than the pool's idle time. Align the pool's max-idle to below `wait_timeout`, or enable pool keepalive/validation.

---

6. Replication Errors & Lag

A replica can be *broken* (a thread stopped with an error) or *lagging* (running but behind). Check both threads and the lag counter. Command name differs by fork.

```
-- MySQL 8.0.22+
SHOW REPLICA STATUS\G
-- MariaDB and older MySQL
SHOW SLAVE STATUS\G
```

Read these fields from the output:

- `Replica_IO_Running` / `Slave_IO_Running` and `Replica_SQL_Running` / `Slave_SQL_Running` — both must be `Yes`. IO=No is a connection/auth issue to the primary; SQL=No is an applied-statement error.
- `Seconds_Behind_Master` (`Seconds_Behind_Source` on 8.0) — replication lag in seconds. `NULL` means a thread is stopped, not "caught up".
- `Last_SQL_Error` / `Last_IO_Error` — the exact failure (e.g. `1062` duplicate key applied on the replica, a missing row on `UPDATE`).

```
-- GTID health (both forks, syntax differs)
SHOW GLOBAL VARIABLES LIKE 'gtid_mode';            -- MySQL
SHOW GLOBAL VARIABLES LIKE 'gtid_slave_pos';       -- MariaDB

-- Restart a stopped SQL thread AFTER you understand the error
-- MySQL 8.0:  START REPLICA;      STOP REPLICA;
-- MariaDB:    START SLAVE;        STOP SLAVE;
```

- Lag with both threads `Yes` → the SQL apply is single-threaded or I/O-bound. Check `Created_tmp_disk_tables`, enable parallel replication (`replica_parallel_workers` / `slave_parallel_threads`), or find the big transaction blocking apply.
- SQL thread stopped on `1062` / `1032` → data drift. **Do not** blindly `SET GLOBAL sql_slave_skip_counter = 1;` — that hides drift. Investigate why the row differs first; skipping is a last resort with a change record.

---

7. InnoDB Issues (Buffer Pool, Log Waits, Corruption)

InnoDB internals surface as stalls, log-flush waits, or — worst case — page corruption that crashes the server on read. `SHOW ENGINE INNODB STATUS` is again the primary lens.

```
SHOW ENGINE INNODB STATUS\G     -- read BUFFER POOL, LOG, and SEMAPHORES sections

-- Buffer pool efficiency (hit ratio) and pressure
SHOW GLOBAL STATUS LIKE 'Innodb_buffer_pool_reads';         -- misses that hit disk
SHOW GLOBAL STATUS LIKE 'Innodb_buffer_pool_read_requests'; -- logical reads
SHOW GLOBAL VARIABLES LIKE 'innodb_buffer_pool_size';

-- Redo log pressure: high = checkpointing can't keep up
SHOW GLOBAL STATUS LIKE 'Innodb_log_waits';
SHOW GLOBAL VARIABLES LIKE 'innodb_log_file_size';    -- <=8.0.29; innodb_redo_log_capacity on 8.0.30+
```

```
# Corruption / assertion failures show in the error log as page checksum errors
sudo grep -Ei "corrupt|checksum|assertion|InnoDB: Database page" /var/log/mysql/error.log
```

- Low buffer-pool hit ratio (`reads` / `read_requests` climbing) → the working set doesn't fit in RAM; raise `innodb_buffer_pool_size` (rule of thumb ~70% of a dedicated host's RAM).
- `Innodb_log_waits > 0` → redo log too small for the write rate; increase `innodb_redo_log_capacity` (8.0.30+) or `innodb_log_file_size`.
- `InnoDB: Database page corruption` / checksum mismatch → **stop making it worse.** Take a cold backup of the datadir now. Recover with `innodb_force_recovery` (start at 1, only raise as needed) *for dumping data out*, then rebuild from backup — do not run production on a forced-recovery instance.

---

8. Disk Full & Table Full

`ERROR 1114 (HY000): The table is full` or writes failing with `Disk is full`. Either the filesystem is out of space, `tmpdir` filled during a big sort, or the binary logs grew unbounded.

```
# Where did the space go?
df -h /var/lib/mysql /tmp
sudo du -sh /var/lib/mysql/* 2>/dev/null | sort -rh | head
sudo du -sh /var/lib/mysql/binlog.* /var/lib/mysql/mysql-bin.* 2>/dev/null | tail
```

```
SHOW GLOBAL VARIABLES LIKE 'tmpdir';              -- fills during big filesorts/joins
SHOW GLOBAL VARIABLES LIKE 'binlog_expire_logs_seconds';   -- MySQL 8.0 retention
SHOW GLOBAL VARIABLES LIKE 'expire_logs_days';             -- MariaDB / older MySQL

-- Binlogs are often the culprit; list and purge SAFELY
SHOW BINARY LOGS;
-- Purge only logs no replica still needs (check SHOW REPLICA STATUS positions first!)
-- PURGE BINARY LOGS BEFORE '2026-07-01 00:00:00';
```

- `The table is full` on an in-memory temp table → the query exceeded `tmp_table_size`; it spilled and `tmpdir` filled. Free space or point `tmpdir` at a larger volume (needs restart).
- Binlogs eating the volume → confirm no replica still reads them (`SHOW REPLICA STATUS` / relay positions), then `PURGE BINARY LOGS BEFORE ...`. **Never** `rm` binlog files by hand — it corrupts the `binlog.index` and can break replication.
- A full datadir can leave the server read-only or crashed. Reclaim space, then confirm writes resume (`SHOW GLOBAL VARIABLES LIKE 'read_only'`).

---

9. Authentication & Access (1045)

`ERROR 1045 (28000): Access denied for user 'app'@'10.0.0.5' (using password: YES)`. MySQL grants are matched by *user + host pattern*, so the same password can work from one host and fail from another. In 8.0 the default auth plugin also changed, which breaks older clients.

```
-- What does the server actually think this account can do, and from where?
SELECT user, host, plugin FROM mysql.user WHERE user = 'app';
SHOW GRANTS FOR 'app'@'10.0.0.%';

-- Confirm which account a live session authenticated as
SELECT CURRENT_USER(), USER();     -- CURRENT_USER = matched grant row; USER = requested
```

- `Access denied ... using password: YES` but the password is right → the connecting **host** doesn't match any grant row. A grant for `'app'@'localhost'` does **not** cover `'app'@'10.0.0.5'`. Add the right host pattern.
- `plugin` is `caching_sha2_password` (MySQL 8.0 default) and an old client/driver fails to authenticate → either upgrade the driver, use TLS, or `ALTER USER 'app'@'%' IDENTIFIED WITH mysql_native_password BY '...';` (MariaDB uses `mysql_native_password` / `ed25519`).
- After any grant change: `FLUSH PRIVILEGES;` is only needed if you edited `mysql.*` tables directly — `GRANT` / `CREATE USER` apply immediately.

---

10. Data Errors (1062 / 1452 / 1366)

Application-visible integrity errors. They are *correct* database behavior rejecting bad writes — the fix is usually in the data or the app, not the server.

```
-- 1062 Duplicate entry: a value collides with a UNIQUE/PRIMARY key
-- ERROR 1062 (23000): Duplicate entry 'user@example.com' for key 'orders.uniq_email'
SELECT customer_email, COUNT(*) FROM appdb.orders
GROUP BY customer_email HAVING COUNT(*) > 1;      -- find the collisions

-- 1452 Foreign key: child row references a missing parent
-- ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint fails
SELECT o.* FROM appdb.orders o
LEFT JOIN appdb.customers c ON c.id = o.customer_id
WHERE c.id IS NULL;                               -- orphans that violate the FK

-- 1366 Incorrect string value: charset/collation mismatch (e.g. emoji into latin1)
-- ERROR 1366 (HY000): Incorrect string value: '\xF0\x9F...' for column 'note'
SHOW FULL COLUMNS FROM appdb.orders;              -- check per-column Collation
SHOW VARIABLES LIKE 'character_set_%';
```

- `1062` → decide intent: is the duplicate a retry (use `INSERT ... ON DUPLICATE KEY UPDATE`), or genuinely bad data to dedupe? Do not just drop the unique key.
- `1452` → the parent row is missing or was deleted out of order. Fix the insert order / cascade rules; check for a `SET FOREIGN_KEY_CHECKS=0` bulk load that left orphans.
- `1366` → a `utf8mb4` string is hitting a `latin1` / `utf8` (3-byte) column. Convert the column: `ALTER TABLE appdb.orders CONVERT TO CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci;` (MariaDB: `utf8mb4_general_ci`). Ensure the client connection is also `utf8mb4`.

---

11. Decision Tree

```
Database incident?
 ├─ Can't connect at all?
 │   ├─ ERROR 1040 too many connections ── §2: pool leak vs. real ceiling; connect on super slot
 │   ├─ ERROR 1045 access denied ──────── §9: check user@host grant + auth plugin
 │   └─ server down / Uptime tiny ─────── §1/§7: error.log, dmesg OOM, corruption?
 ├─ Connected but slow / hanging?
 │   ├─ Threads_running high, State=Sending data/copy to tmp ── §4: EXPLAIN, add index
 │   ├─ State=Waiting for lock / 1205 / 1213 ──────────────── §3: INNODB STATUS, kill blocker
 │   └─ Innodb_log_waits / low buffer hit ratio ──────────── §7: buffer pool / redo sizing
 ├─ Writes failing?
 │   ├─ 1114 table is full / disk full ── §8: df -h, tmpdir, purge binlogs safely
 │   ├─ read_only = ON ──────────────── §1: replica promoted? disk-full lockout?
 │   └─ 1062 / 1452 / 1366 ──────────── §10: data/constraint/charset — fix data, not server
 ├─ Connection drops mid-query (2006/2013)? ── §5: max_allowed_packet, wait_timeout, OOM
 └─ Replica broken or behind?
     ├─ IO/SQL thread = No ── §6: Last_*_Error; understand BEFORE skip/restart
     └─ Seconds_Behind high — §6: parallel apply, big txn, I/O
```

12. Copy/Paste One-Shot Triage

Read-only. Run as an admin user against the ailing instance.

```
DB="mysql -h db-1 -uadmin -p --connect-timeout=5"   # add -P3306 if non-default port
echo "== version =="        ; $DB -N -e "SELECT VERSION();"
echo "== saturation =="      ; $DB -e "SHOW GLOBAL STATUS WHERE Variable_name IN
('Threads_connected','Threads_running','Max_used_connections','Aborted_connects','Slow_queries');"
echo "== limits =="          ; $DB -e "SHOW GLOBAL VARIABLES WHERE Variable_name IN
('max_connections','wait_timeout','max_allowed_packet','read_only','innodb_buffer_pool_size');"
echo "== top by host =="     ; $DB -e "SELECT user,host,COUNT(*) c FROM information_schema.processlist GROUP BY user,host ORDER BY c DESC LIMIT
10;"
echo "== active (not idle) ="; $DB -e "SELECT id,user,time,state,LEFT(info,60) info FROM information_schema.processlist WHERE command<>'Sleep'
ORDER BY time DESC LIMIT 15;"
echo "== innodb waits ==="     ; $DB -e "SHOW GLOBAL STATUS WHERE Variable_name IN
('Innodb_log_waits','Innodb_row_lock_waits','Created_tmp_disk_tables');"
echo "== replica (8.0) ==="    ; $DB -e "SHOW REPLICA STATUS\G" 2>/dev/null | grep -E "Running|Behind|Last_.*Error" || $DB -e "SHOW SLAVE
STATUS\G" | grep -E "Running|Behind|Last_.*Error"
echo "== disk =="             ; df -h /var/lib/mysql
```

13. Incident Notes Template

```
INCIDENT: MySQL / MariaDB
Started (UTC):        ____    Detected by: ____    Sev: ____
Server:               db-1:3306   Flavor/version: ____ (MySQL 8.0 / MariaDB 10.x)
Symptom / error code: [ ] 1040 conns [ ] 1205/1213 lock [ ] slow [ ] 2006 [ ] repl [ ] 1114 disk [ ] 1045 auth [ ] 1062/1452/1366 data
Saturation:           Threads_connected __/__ max   Threads_running ____
Blocking evidence:    (INNODB STATUS: blocker trx id ____ / query ____)
Replica:              IO ____ SQL ____ Seconds_Behind ____ Last_SQL_Error ____
Disk:                 /var/lib/mysql __% used   binlogs ____
Action taken:         (KILL id / index added / config change / purge / restart + who + when)
Change record:        ____
Result:               [ ] recovered @ ____   [ ] escalated to ____
Root cause:           ____
Follow-ups:           (pool sizing, index, alert, retention, monitoring) ____
```