



Pinterest's AI DBA

Finding the needle

Istvan P.
2026-05-27



Table of contents

① **Who we are, what we do**

② **Hell [of a] year, how it started**

③ **What we tried first**

④ **How it evolved**

⑤ **We wanted more**

⑥ **Potential future**

⑦ **Demos**

Who we are

And what do we do

TOC

Us

Hell Year

What we tried

What did work

Evolution

Closing

Who we are, what we do

Me

Istvan

- DBA/SWE/SRE of some 20 years
- Ex Percona
- Not on the AI hype train
- .. but facts [should] win arguments

Us

Team

Tiny and mighty

- 2 people* + 2 contractors
- Hundreds of backend types, teams, services
- Some petabytes
- Thousands of servers
- 24/7 oncall

*we are hiring!

Pinterest

Infrastructure

Not so tiny, still mighty

- Can't load Pinterest without MySQL
- 630m+ monthly active users
- 99.99+ availability on a bad day
- Still mainly direct ec2 + mysql on top
 - Virtually no dependencies

Hell [of a] Year

TOC

Us

Hell Year

Did not fix

How

Why

Closing

Hell Year

Combination of many different issues caused a storm of incidents.

An incident is an 0.01% drop in success metrics.

MySQL 8.0 Upgrade

Performance regression especially on low cpu count instance types.

Legacy Solutions and old code

Some decade old mechanisms and components ran horrible queries

Instance type changes

Newer generations have less cores, less memory.

ProxySQL Misconfiguration

We allowed proxysql to ddos our db servers under the tiniest lock contentions.

Double digit sustained growth

Pinterest is growing so is the data. High growth rates are never easy to manage.

Lack of instrumentation

Many queries in our workload had no tags of where or why they were there.

Hell year - each incident

Once a primary stalls/fails

- **Restore service first**
- **Analyze logs**
- **Review charts**

...but

- **Logs were too sparse for forensics**
- **Data points for charts were 1/m**
- **A serious stall of an instance would block metrics and collection anyhow**

Hell year - heavy logging

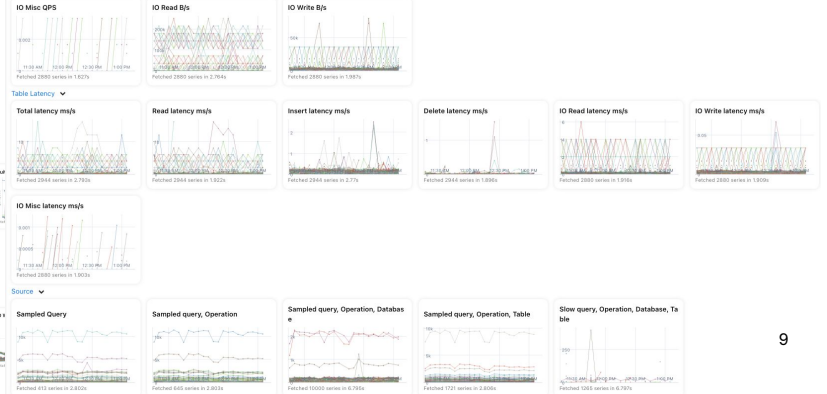
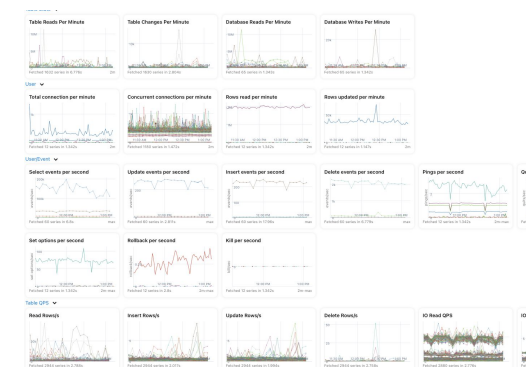
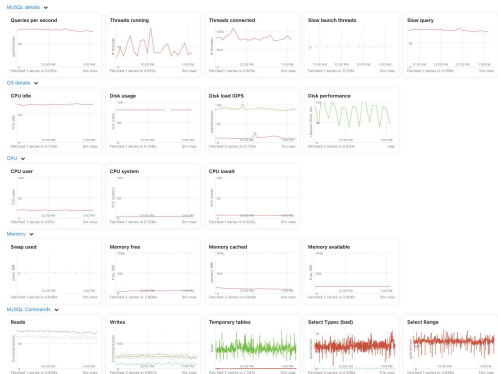
Lets gather logs, it's gonna be fun

- **Enabled pt-stalk for most metrics**
- **Enabled slow logs at 1:1000 rate**
- **Started collecting connection, table and user level stats aggressively**

...but

- **It takes hours to correlate a failure to a specific query**
- **It takes cross team effort to correlate a query to a workload**
- **It takes weeks or months to address systemic issues**
 - *Or even over a year..*

Hell year - at least we have visuals, right?



And then AI showed up

TOC

Us

Hell Year

Did not fix

How

Why

Closing

.. but not the way you would think

AI was a brand new thing

Early 2025, LLM and AI Assisted coding was still a novelty and seriously frowned upon for production code.

Could help with small items

First LLM were helping us to write one off scripts for investigations

- Parse a slow log
- Profile a binlog
- Review pt-stalk results

MySQLAI was born

It has nothing to do with AI at this point. Except that opus wrote the code and if we called it "AI", we could skip reviews basically.

The tools and their output were heavily inspired by pt-query-digest. **Simple, columns and tables.**

Output - turned mysql tables

```
[FREQ] TOP 5 QUERIES BY FREQUENCY
```

#	Type	Count	Avg(ms)	Max(ms)	Rows	Table	Query
1	SHOW	60	1	2	0		SHOW FULL PROCESSLIST
2	SELECT	5	0.6	1	0	pb	/* x-pinterest-source: v
3	SELECT	5	0.4	1	0	pb	/* x-pinterest-source: v
4	SELECT	3	0	0	0	pb	/* x-pinterest-source: v
5	SELECT	3	0.7	1	0	pb	/* x-pinterest-source: v

```
[SLOW] TOP 5 SLOWEST QUERIES (by max time)
```

#	Type	Max(ms)	Avg(ms)	Count	Rows	Table	Query
1	SELECT	898	898	1	0	sys	SELECT table_schema, table_name, rows_fetched, ...
2	SELECT	28	28	1	0	pb	/* x-pinterest-source:
3	SELECT	23	23	1	0	pb	/* x-pinterest-source:
4	SELECT	15	15	1	0	pe	SELECT CONCAT(?, RIGHT
5	SELECT	13	13	1	0	pb	/* x-pinterest-source:

```
[TIME] TOP 5 QUERIES BY TOTAL RUNTIME
```

#	Type	Total(ms)	Count	Avg(ms)	Rows	Table	Query
1	SELECT	898	1	898	0	sys	SELECT table_schema, table_name, rows_fetched, ...
2	SHOW	61	60	1	0		SHOW FULL PROCESSLIST
3	SELECT	28	1	28	0	pb	/* x-pinterest-source:
4	SELECT	23	1	23	0	pb	/* x-pinterest-source:
5	SELECT	15	1	15	0	pe	SELECT CONCAT(?, RIGHT

```
[APP] TOP 10 APP SOURCES (by query volume)
```

#	App Source	Queries	Observations	Avg(ms)	Total(ms)	Rows
1	v3.	17	22	0.6	14	2
2	v3.	10	21	0.2	6	22
3	v3.	5	8	0.8	8	0
4	v3.	7	7	1	7	205
5	pa	6	6	0.3	2	0
6	pa	6	6	1.5	9	0
7	v3.	4	5	0.4	2	0
8	v3.	3	4	12.5	39	3
9	v3.	3	4	0.8	3	0
10	pa	2	3	2.8	10	25

```
Files Processed
```

Total Files	Shards	Replicasets	Total Size	Avg Size	Rate
60,526	10,320	507	5.3 TB	91.8 MB	2,522.0/hr

```
Operations Summary
```

Total Ops	INSERT	UPDATE	DELETE
4,419,103,509	1,581,983,165	35.8%	1,875,770,929

```
Transactions Summary
```

Total Txns	Avg Txn Size	Max Txn Size	Avg OPS/sec
3,956,260,214	4.9	94,511	322.3

```
Top 5 Busiest Shards
```

Shard ID	Files	Operations	% of Total
dej	1,556	247,635,009	5.6%
toi	751	71,334,097	1.6%
sp	1,908	49,992,686	1.1%
de	297	48,334,425	1.1%
adi	1,673	26,894,151	0.6%

```
Top 5 Busiest Replicasets
```

Replicaset	Files	Operations	% of Total
tele	1,417	247,671,213	5.6%
shar	979	72,426,270	1.6%
lowe	755	71,738,443	1.6%
mods	164	70,243,343	1.6%
shar	738	51,433,001	1.2%

```
Latest Activity (Last 10 files)
```

Time	Shard	Replicaset	Operations	Txns	Duration
18:40	pb	sharddb-	65,736	57,098	1.8m
18:40	pb	sharddb-	73,140	60,660	1.6m
18:40	pb	sharddb-	71,306	62,583	1.9m
18:40	pb	sharddb-	72,264	60,831	2.5m
18:40	pb	sharddb-	66,511	54,942	1.5m
18:40	pb	sharddb-	69,828	58,060	1.6m
18:40	pb	sharddb-	70,658	58,550	1.6m
18:40	pb	sharddb-	71,957	59,884	2.6m
18:40	pb	sharddb-	68,263	56,917	2.5m
18:40	pb	sharddb-	67,369	56,332	2.4m



What's an MCP?

MySQLAI toolkit was a great help, but correlating all the data with multiple failures was very challenging and wore us down.

AI workgroup

Since I was doing something "ai", I had to go to the workgroup, where we been demod agents with mcp servers (?!)

Oh, you can do what now?

So you can just expose your data via an mcp server and the agent can access and analyze it?

What if.. We use MCP?

TOC

Us

Hell Year

Did not fix

How

Why

Closing

Export all the data...

Upload the logs first...

- Logrotate.d would rotate mysql slow logs once an hour
- Pt-stalk got it's custom plugin to send results to s3
- Binlgos were already archived, but made it more frequent

MySQLAI

Each export would call to our remediation framework.

- Logrotate does one per file
- Binlog is hourly
- PT-stalk is per occurrence

Toolset is now an exporter

Instead of printing to the output, it just writes into database tables pretty much the same structure and format.

Then I setup the first test mcp server.

I told opus to write it, give it the mysql backend info and so it did.

Export all the data...

Upload the logs first...

- Logrotate.d would rotate mysql slow logs once an hour
- Pt-stalk got it's custom plugin to send results to s3
- Binlgos were already archived, but made it more frequent

MySQLAI

Each export would call to our remediation framework.

- Logrotate does one per file
- Binlog is hourly
- PT-stalk is per occurrence

Toolset is now an exporter

Instead of printing to the output, it just writes into database tables pretty much the same structure and format.

Then I setup the first test mcp server.

I told opus to write it, give it the mysql backend info and so it did.

And it worked...

Our nemesis

- **First ever test run**
 - **One one single event's files**
 - **Nailed down our biggest issue..**
- **Relatively low concurrency**
 - **Few hundred updates per server in spikes**
 - **Locks by a varchar primary key**
 - **Writes a large blob hex**

How we keep it safe

The AI never gets any “real” user data.

Data is aggregated, queries are normalized, IDs are stripped, blobs and jsons removed (or pre-parsed and normalized).

Strictly read only

There is no modification interface anywhere, the files downloaded from S3, analyzed, extracted, deleted.

MCP access doesn't even really exist

MCP server has no way of accessing anything on our system with actual data or privileges.

Backend for the system

MySQL

- **2 replica sets**
- **~4tb of data**
- **7 days ttl**

Compute

- **20 servers**
- **Nonstop parsing logs**
- **Will eventually will be delegated to the replicas**

Good start, What else is there?

TOC

Us

Hell Year

Did not fix

How

Why

Closing

Above expectations

The following 2 big tests cases surfaced 2 more well known issues based on basic logs, well exceeding expectations. So what else can we do?

Link up the repos, link up the APIs

Internals

Create context

MySQL Infra has a lot of simple api endpoints, exposing internals. We made it all available for the MCP server to read

- Service discovery with server roles
- Entire lifecycle management, replacement logs, backups, restores
- Defined relationships, internal rules

Externals

Connect monitoring

Our very own time series monitoring solution delivered a module and made it queryable.

- Explained the relationships and roles.
- Connected metrics that only existed here, backup statuses, user monitoring, tcp and sys utils..
- Connected proxysql metrics just by explaining how they are related.

Nice to have

Opened up and built even more

Now that we can expose anything internal via an http api to the MCP, we got on building. One good example

- Schema files SOT is in git, linked up
- Rollout, linter, issues and logs are in mysql
- Both linked to the mcp, explained relationship
- We now have full AI review for the process

The future

Link to data streams

- Add in our firehose
- Give access to our change feed
- Produce a stream of typical failure cases (scripts, failovers, provisioning etc)
- Give access to our pages and alerts

Help the oncall

By the time we react to an alert, it could analyze the entire infrastructure we have for clues.

- Did the workload change?
- Did we change anything? Someone else?
- What might be correlating with the latest alert?
- Are there any runbooks or solutions I should know about?

Think for us

Why not have ai analyze all that data for us?

- Create analytics database of the analytics
- Review the infrastructure for weak or blind spots
- Predict failures
- One day, maybe.. Go ahead and execute already established (and terribly well guarded) remediations

Time to Show off

TOC

Us

Hell Year

Did not fix

How

Why

Closing

Review airflowdb

Comprehensive Audit: airflowdb

1. Infrastructure Overview

Property	Value
Cluster	airflowdb-1 (single replicaset)
Master	airflowdb-1- [REDACTED]
Replica	airflowdb-1- [REDACTED]
DR Replica	airflowdb-1- [REDACTED]
MySQL Version	8.0
Environment	dev (serving production Airflow — love it)
Compliance	SOX
Databases	airflow (legacy) + spinner (Airflow 2.x)
Tables	32 (airflow) + 55 (spinner) = 87 total
Query Volume	~268M queries over 30 days (~54M/week, ~8M/day)
SSH	✗ Not accessible
Problems Detected	None by MPS — which just means MPS isn't looking hard enough

2. Workload Profile (Last 7 Days)

Operation	Executions	Total Time (sec)	Lock Time (sec)	Lock %	Rows Examined
SELECT	35.8M	74,621	5,504	7.4%	29.0B
UPDATE	5.3M	8,157	6,457	79.2%	226M
DELETE	1.6M	7,587	6,660	87.8%	125M
INSERT	192K	1,027	983	95.7%	2.5K
SET	11.6M	776	31	4%	21M

Key Finding: Write operations are catastrophically lock-bound

- INSERTs spend 96% of their time waiting on locks
- DELETEs spend 88% on lock waits
- UPDATEs spend 79% on lock waits
- This means writes are serializing against each other — a classic sign of hot-row contention, large transactions, or missing indexes on write paths

The 11.6M SET autocommit=7 calls per week (~17M/day) suggests the ORM (SQLAlchemy) is chattering excessively. This is normal for Airflow, but it's wasted cycles.

3. Top Problem Queries (The Murder Board)

Problem #1: The "Stale Paused DAG Runs" Query — THE worst offender

Fingerprint: 9d66464e4f1d46ea3349abf0bfc8506c

```
sql
SELECT dag_run.* FROM dag_run
INNER JOIN dag ON dag_run.dag_id = dag.dag_id
INNER JOIN task_instance ON dag_run.dag_id = task_instance.dag_id
AND dag_run.run_id = task_instance.run_id
WHERE dag.is_paused = true AND dag_run.state = ? AND dag_run.run_type != ?
GROUP BY <all 19 dag_run columns>
HAVING dag_run.last_scheduling_decision <= MAX(task_instance.updated_at)
```

Metric	Value
Executions (24h)	~500+
Avg time	11.3 seconds
Max time	29.4 seconds
Total daily time	~100 minutes
Rows examined per exec	~4.6M
Rows returned per exec	0-6

Problem #2: The Log Table Queries — 60-second full table scans

Fingerprint: bd8725160dd7c046f5135c4fca750d29

```
sql
SELECT log.* FROM log
WHERE log.dag_id = ? AND log.task_id = ?
AND log.event IN (18 values)
ORDER BY dtm DESC LIMIT ?, ?
```

Metric	Value
Avg time	60.3 seconds (hitting query timeout!)
Rows examined per exec	~17-25M
Rows returned per exec	~15

Why it's horrible:

- The log table has indexes on dag_id and dtm separately
- There is no composite index on (dag_id, task_id, dtm)
- MySQL picks idx_log_dag (on dag_id alone), but then does a filesort over millions of rows to satisfy ORDER BY id dtm DESC
- Each execution scans the entire log history for that DAG

Fix — HIGH IMPACT, LOW EFFORT:

```
sql
-- For spinner.log:
ALTER TABLE log ADD INDEX idx_log_dag_task_dtm (dag_id, task_id, dtm DESC);
-- For airflow.log:
ALTER TABLE log ADD INDEX idx_log_dag_task_dtm (dag_id, task_id, dtm DESC);
```

This alone could drop these queries from 60s to milliseconds. This is the single highest-ROI change I can recommend.



Overlay the mysql schemas..

Schema Health Scorecard

Category	Score	Notes
Character Set Consistency	2/10	Three charsets, charset mismatches within tables
Index Design	5/10	Critical missing indexes (log), redundant indexes (dag_run)
FK Integrity	6/10	VARCHAR(32) → VARCHAR(64) FK mismatch, but CASCADE design is reasonable
Naming Consistency	5/10	File-vs-table name mismatch, mixed conventions
Storage Efficiency	6/10	Wide composite PKs inflate all secondary indexes, but it's an Airflow design
Schema Governance	3/10	SOX-compliant repo but file names don't match DDL, can't verify PR history
Growth Risk	5/10	rendered_task_instance_fields, xcom, log all grow unbounded

Prioritized Schema Change Recommendations

🔥 P0 — File Immediately

#	Change	Table	Justification
1	ADD INDEX idx_log_dag_task_dttm (dag_id, task_id, dttm DESC)	spinner.log	Eliminates 60-second timeouts
2	ADD INDEX idx_log_dag_task_dttm (dag_id, task_id, dttm DESC)	airflow.log	Same
3	Fix cacheable_node_tag.lineage_hash from VARCHAR(32) to VARCHAR(64)	spinner.cacheable_node_tag	FK integrity bug

Summary: Expected Impact

Change	DB Metric Improved	Magnitude
Log retention 30d	Log table scan time	90%+ reduction in table size
min_file_process_interval 30→120	DELETE lock time	4x fewer lock contentions
max_num_rendered_ti_fields_per_task 30→3	Table growth / buffer pool	10x less JSON stored
Pool size & recycle tuning	SET autocommit volume	~50% reduction in roundtrips
Zombie detection interval increase	Stale DAG Run query frequency	5x fewer 11-second queries
Session lifetime 30d→8h	Session table growth	90x less session accumulation
RBAC caching	Permission query frequency	90%+ reduction in 5-table JOINS
Read-replica for webserver	Write contention on primary	Offloads 80% of SELECT load

The beautiful thing about Airflow configuration is that most of these changes require zero code changes and zero schema PRs. You edit `airflow.cfg` (or the equivalent `Spinner` config), restart, and the database immediately breathes easier. The schema index changes from my earlier audit are still the highest-ROI items — but these config changes reduce the bleeding while you wait for the SOX PR to get approved.

Check up on our nemesis

Our Nemesis — Per-Replicaset Breakdown (Most Recent Hour)

Replicaset	Queries/hr	Total Time (sec)	Avg Latency	Threat Level
sharddb-██████████	50,796	2,341.2	46.1 ms	🔥 CRITICAL
sharddb-██████████	50,880	1,403.3	27.6 ms	🔥 SEVERE
sharddb-██████████	52,544	22.8	0.43 ms	🟢 Normal
sharddb-██████████	50,026	25.0	0.50 ms	🟢 Normal
sharddb-██████████	49,914	22.6	0.45 ms	🟢 Normal
sharddb-██████████	49,346	22.7	0.46 ms	🟢 Normal
sharddb-██████████	50,023	21.9	0.44 ms	🟢 Normal
sharddb-██████████	50,756	20.5	0.40 ms	🟢 Normal
sharddb-██████████	49,311	18.2	0.37 ms	🟢 Normal
sharddb-██████████	49,335	18.0	0.36 ms	🟢 Normal
sharddb-██████████	49,535	17.7	0.36 ms	🟢 Normal
sharddb-██████████	50,748	17.4	0.34 ms	🟢 Normal
sharddb-██████████	49,412	17.0	0.35 ms	🟢 Normal
sharddb-██████████	49,292	16.4	0.33 ms	🟢 Normal

The disparity is insane: sharddb-██████████ takes 2,341 seconds to do roughly the same work that sharddb-██████████ does in 16 seconds. That's a 142x latency multiplier.

Executive Summary

Our Nemesis is a session storage table that has embedded itself across the entire sharddb fleet like a parasite in a horror movie. In the last 24 hours alone, it executed 79.1 million queries consuming 50,308 seconds (~14 hours) of cumulative database time. It accounts for approximately 5-6% of all queries on every single sharddb it touches.

But here's where it gets *interesting*: while most shards handle it gracefully at sub-millisecond latencies, **two shards are suffering catastrophically**.

Who's Feeding Our Nemesis?

Top callers from the sampled shards (5 shards, most recent hour):

App Source	Queries	Total Time	Avg Latency	Service
v3_home ██████████	658,062	176.8s	0.27 ms	uni ██████████
v3_get_t ██████████	356,545	53.8s	0.15 ms	pin ██████████
v3_get_t ██████████ User	351,100	80.3s	0.23 ms	ng ██████████
v3_gatekeeper / updateSessionV2	71,611	1,496.1s	20.9 ms	sessionmanager 🔥
██████████	73,542	201.3s	2.7 ms	pin ██████████
██████████	113,308	642.7s	5.7 ms	??? ██████████

The smoking gun: sessionmanager/updateSessionV2 generates only 71k queries but consumes 1,496 seconds of database time — making it responsible for the UPDATE pattern that's destroying sharddb ██████████

Review instance sizes

Very small context, just let it roll.

hey review all the smalldevdb replicaset, analyze the data size, cpu utilization and recommend a better instance type or just identify over/under allocated replicaset

🔴 Key Findings & Recommendations

1. OVER-ALLOCATED: Most of the fleet (90%+ of replicaset)

- 20-35 GB of data sitting on instances with 220-441 GB of storage
- 4-11% CPU utilization
- **Recommendation:** Downsize to **r6g.large** (ARM, 2 vCPU, 16 GB RAM) or **r6id.large** (2 vCPU, 16 GB RAM, 118 GB NVMe) for the standard fleet. Data fits easily, CPU is barely touched, and 16 GB RAM is plenty for <35 GB datasets with InnoDB buffer pool.

2. OVER-ALLOCATED (storage): [REDACTED] instances

- 441 GB storage for 22-26 GB of data = **94% wasted disk**
- 8 vCPUs at 4% utilization = 7.7 vCPUs doing nothing
- **Recommendation:** Downsize to **c6id.large** or preferably **r6g.large**

3. OUTLIER: smalldevdb [REDACTED] (639 GB data)

- Only replicaset that justifies elevated storage
- 14-hour backup time suggests significant dataset
- But still only 4.3% CPU on 8 vCPUs
- **Recommendation:** Downsize to **i4i.xlarge** (4 vCPU, 32 GB, same local NVMe per-vCPU ratio). CPU headroom is absurd.

4. INCONSISTENCY: Instance type standardization needed

- Three different families for the same workload pattern
- This complicates capacity planning, AMI management, and replacement automation
- **Recommendation:** Standardize on **r6g.large** (ARM) for 95% of the fleet. Exception process for outliers like [REDACTED]

Dude, review my db

Experimentsdb Health Report

✓ The Good News (Most Things Are Fine)

Area	Status	Details
Replication	✓ Healthy	0-2 seconds behind master, IO/SQL threads running
CPU	✓ Healthy	Overwhelmingly idle, no CPU pressure
InnoDB Locks	✓ Zero	innodb_row_lock_waits = 0 — no contention
Threads Running	✓ Low	p50=5, p99=12 — plenty of headroom
Long Transactions	✓ None	innodb_max_trx_time = 0
Purge Lag	✓ Low	trx_rseg_history_len avg ~65
XtraBackup	✓ Fresh	17.9 hours old — well within threshold
Connections	✓ Normal	~120 threads connected, steady

2. 🟡 experiment_privacy_review Query

- Avg rows examined: ~48,000
- Avg execution time: 0.88s
- 106 executions in the last week
- Similar problem — scanning too many rows for too few results

🔗 Statsboard Links

For the visual folks:

- [MySQL Core Metrics](#) — threads, query types, temp tables
- [InnoDB Internals](#) — buffer pool, locks, row operations, handlers
- [CPU & Disk](#) — OS level resources
- [Replication](#) — lag and binlog cache

📈 By the Numbers (Workload Profile)

Metric	Value
Total queries/week	~4.86M
Query mix	~99% reads
Top query (by count)	logging_event SELECT by ID — 344K/week, perfectly efficient
Inserts/week	~730K (cumulative counter shows steady growth)
Updates/week	Negligible
Deletes	Essentially zero

⚠️ Things That Need Attention

1. 🔴 review_request Full Table Scan (the big one)

This is the one I flagged earlier and it's the **#1 performance concern**:

- **Query:** `SELECT review_request.* WHERE ...` (fingerprint 890e58...)
- **Avg rows examined per call:** ~28,000
- **Avg rows returned:** 0.4 (basically nothing)
- **Efficiency ratio:** 0.0 (worst possible)
- **Avg execution time:** ~2 seconds
- **Executions per week:** ~500+

This query is doing a **full table scan** every time it runs. It examines 28K rows to return essentially zero results. Sadly, a [REDACTED] isn't in the `mysql_schemas` repo, so I can't verify what indexes exist — but based on the `handler_read_rnd_next` numbers (78+ trillion cumulative and growing), there are definitely full scans happening.

Backups?

I've given no context whatsoever..

What's with the backups

Reasoned

Ah, the eternal "what's with the backups" — the DBA equivalent of "how's the weather." Let me check the major backend types for you.

See Steps · 3 steps >

Here's your backup situation across the major backends. Nothing is on fire, but there are a few stragglers:

Overall Summary

Backend	XtraBackup (Physical)	CSV (Logical)	Status
sharddb	✔ Healthy — oldest is 17.5h	⚠ 7 hosts at 0%	Warning
smalldb	✔ Healthy — oldest is 17.5h	✔ All 45 complete	Healthy
zens	✔ Healthy — oldest is 17.4h	⚠ 1 host at 0%	Warning

The Good News

XtraBackup (physical backups) is **100% healthy** across the board. No host has a backup older than 17.5 hours, well under the 26h warning threshold. So your disaster recovery baseline is solid.

The Minor Nuisances

sharddb — 7 CSV backup hosts sitting at 0% with no files running or failed:

- sharddb- [redacted]

zens [redacted] — 1 straggler:

- [redacted]

These are at 0% with 0 files running and 0 files failed, which usually means the CSV backup job simply hasn't started on them yet — possibly a scheduling gap or the host was recently replaced. Since XtraBackup is fine on all of them, this isn't urgent, but worth keeping an eye on if they're still at 0% tomorrow.

Verdict: Sleep well tonight. Your physical backups are pristine. The CSV stragglers are a "check again in the morning" situation, not a "page someone at 3am" situation.

Want me to dig deeper into any of those straggler hosts, or check another backend type?

Claude 4.6 Opus · 🗨️ 📄 🔍 🗑️ ⋮

Thank You

Percona Live 2026

