FOSS 2026 · CHERNIHIV POLYTECHNIC



# Teaching MySQL to quack

DuckDB inside MySQL: a storage engine, replication and honest benchmarks

**Evgeniy Patlan**

Engineering Manager, Build & Release · Percona

Oct 1–2, 2026 · online

TPC-H · ~500 GB · 3 BILLION LINEITEM ROWS

InnoDB

# 28+ hours

18 of 22 queries.
Four never finished.

MySQL + DuckDB

# ~3 min

All 22 queries.
Same server, same SQL.

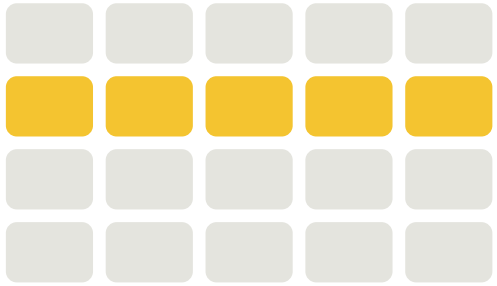
What changed? One word in `CREATE TABLE`.

THE PROBLEM

# Two worlds in one database

## OLTP — transactions

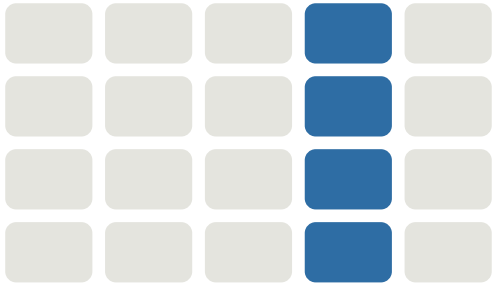
Find one order, update one row. Thousands of short queries per second.



Row store: read the whole row

## OLAP — analytics

Yearly sales by region. Few queries, but each one reads millions of rows.

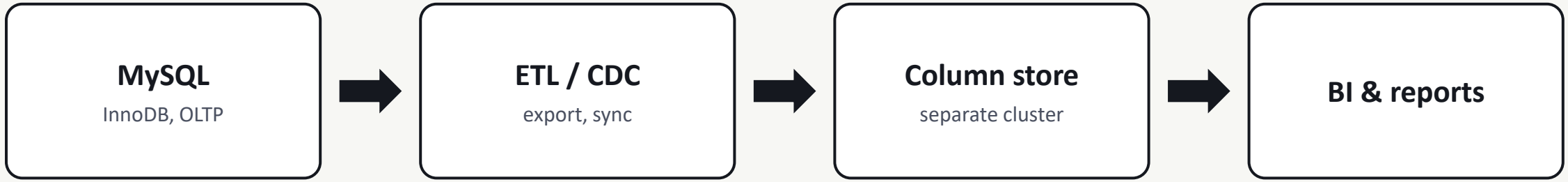


Column store: read only the column you need

**InnoDB stores rows. Analytics needs columns.**

THE CLASSIC FIX

# Two systems and a pipeline between them




**Two systems to run**

**Stale data**

**One more pipeline to break**

## What if the column store were just a table inside MySQL?

THE TOOL

# DuckDB in one minute

## “SQLite for analytics”

- In-process library, no separate server
- MIT license, no external dependencies
- First release in 2019, 1.0 in June 2024
- Current branch 1.5; DuckDB 2.0 due in fall 2026



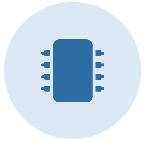
### Columnar storage

Reads only the columns it needs; data compresses well



### Vectorized execution

Processes batches that fit in CPU cache



### Parallelism

A single query spreads across all cores

THE LANDSCAPE

# Where DuckDB lives relative to MySQL

DuckDB stays separate

**Depth of integration →**

**1**

### DuckDB as a client

DuckDB's mysql extension: ATTACH to MySQL

**2**

### Outside the server, via binlog

dbtrail: binlog → Parquet → DuckDB

**3**

### Extension inside the server

VillageSQL vsql-duckdb

**4**

### Storage engine

AliSQL · MariaDB · Percona

**Transparency for users grows →**

**Risk in the commit path grows too →**

LEVELS 1–2

# Outside the server

## DuckDB as a client

```
INSTALL mysql;
ATTACH 'database=shop' AS m
  (TYPE mysql);
SELECT region, sum(amount)
  FROM m.sales GROUP BY 1;
```

Reads and writes MySQL, COPY to Parquet. Zero server changes, but data travels over the network.

## dbtrail: the binlog as a columnar feed




- Changes nothing inside mysqld
- Bonus: change history, undo, point-in-time
- Freshness depends on how often hours rotate to Parquet
- Apache 2.0 · by Daniel Guzman Burgos

LEVEL 3

# Extension: VillageSQL

vsql-duckdb embeds DuckDB in the VillageSQL Server process through its Extension Framework. SQL goes to DuckDB unchanged — DuckDB plans the query itself.

| | |
|---|---|
| `duckdb_scalar()` | a single value |
| `duckdb_query()` | JSON → JSON_TABLE |
| `duckdb_status()` | diagnostics |

No transparent routing, no WHERE pushdown. Results are capped at 1 MB.

```
INSTALL EXTENSION vsql_duckdb;

SELECT duckdb_scalar(
  'SELECT count(*)
   FROM read_parquet(
     ''s3://bucket/x.parquet'')');
```

Strength: Parquet, CSV and JSON in S3 over a plain MySQL connection, joinable with InnoDB tables.

LEVEL 4

# Storage engine: AliSQL and MariaDB

## AliSQL

- First to ship. Alibaba’s fork of MySQL 8.0.44
- Analytical read-only replica fed by the binlog
- Reworked commit and replay, since DuckDB has no 2PC
- DDL that DuckDB can’t do runs as copy DDL

**1,000+ nodes in production, per the AliSQL team**

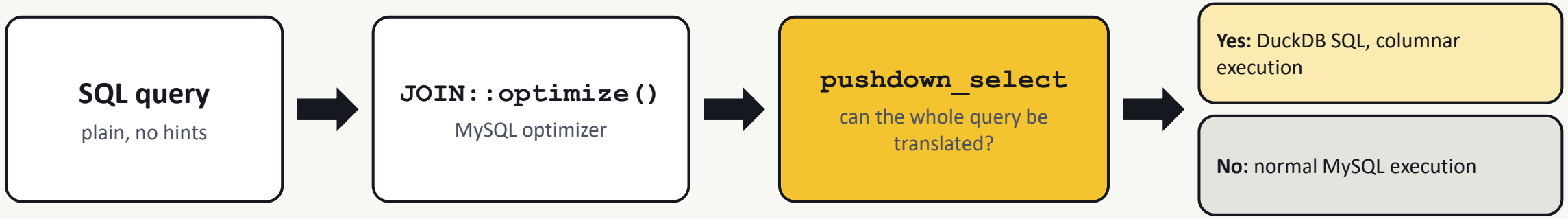
## MariaDB

- ha_duckdb plugin, alpha
- Design borrowed from AliSQL, upstream DuckDB 1.5.2
- Whole SELECT goes through select_handler
- JOINs with InnoDB: DuckDB pulls rows via a fiber callback

**Branches 11.4, 11.8, 12.3, 13.x**

AliSQL has also proposed a DuckDB engine for upstream MySQL (mysql-community, discussion #79).

# MySQL has no select_handler — so we added a hook




**CREATE TABLE … ENGINE=DuckDB**

**Engine compiled into mysqld**

**One schema = one DuckDB file**

PRINCIPLES

# All or nothing

### Whole query or nothing

A query either runs entirely in DuckDB or falls back transparently to normal MySQL execution. No half-queries.

### Identical results

Collation, NULL ordering and DECIMAL scale are matched on purpose. Unsure? It declines. All 22 TPC-H queries match InnoDB row for row.

### Point lookups stay put

A primary-key SELECT stays on the row path: for one row, an index seek beats spinning up DuckDB.

**You don’t pick the engine by hand — the server does.**

# Three generic patches

## 1 A hook after the optimizer

pushdown_select at the end of JOIN::optimize() sets JOIN::override_executor_func, which the executor already checks

## 2 LOAD DATA → DuckDB COPY

Instead of write_row one row at a time: 600 million rows in ~20 minutes instead of ~80

## 3 No semijoin rewrite

For single-engine statements, IN, EXISTS, NOT IN and NOT EXISTS stay subqueries the builder can render

**The patches are generic: they fire for any engine that implements the hook.**

DEMO

# One table, two paths

```
$ docker run -d -p 3306:3306 -e MYSQL_ROOT_PASSWORD=secret \
    perconalab/ducksdb-mysql-engine:9.7-duckdb-v0.2.2

CREATE TABLE sales (id INT PRIMARY KEY, region INT,
  amount DECIMAL(12,2)) ENGINE=DuckDB;
SELECT region, SUM(amount) FROM sales GROUP BY region;
SHOW STATUS LIKE 'Ducksdb_pushdown_count';   -- +1
SELECT * FROM sales WHERE id = 3;
SHOW STATUS LIKE 'Ducksdb_pushdown_count';   -- unchanged
```

BENCHMARK · SF10

# TPC-H: 60 million rows

**1317 s+**

InnoDB finished only 16 of 22; six hit the 180 s cap

**15.1 s**

Our engine: all 22 queries

| Query | InnoDB | Ours | MariaDB | DuckDB |
|---|---|---|---|---|
| Q1 | >180 | 1.77 | 0.84 | 0.77 |
| Q5 | 127.6 | 0.53 | 0.46 | 0.71 |
| Q9 | >180 | 1.52 | 2.30 | 1.78 |
| Q18 | 101.7 | 1.35 | 1.39 | 1.31 |
| Q19 | 120.4 | 0.15 | 0.67 | 0.83 |
| **All 22** | **16/22** | **15.1** | **13.3** | **16.3** |

Seconds, warm, best of several runs. Laptop: 20 cores, 62 GiB RAM, Docker.

BENCHMARK · SCALE

# SF100 and SF500

## SF100 · 600 million rows

| Query | Ours | MariaDB | DuckDB |
|---|---|---|---|
| Q1 | 15.25 | 6.50 | 5.50 |
| Q9 | 20.97 | 115.29 | 19.54 |
| Q18 | 14.88 | 29.62 | 11.08 |
| **Correct** | **22/22** | **20/22** | **22/22** |

Q15 is slow for us only because of a memory cap and spilling to disk; with MariaDB's memory it runs in ≈4 s. Loading: ~20 min vs ~80.

## SF500 · 3 billion rows

80 cores · 187 GB RAM

**186 s**

all 22 queries (InnoDB: 28+ h, 18/22)

**25×**

faster load

**5×**

less disk

REPLICATION

# An analytics replica from the replication you already run




## Create tables by hand

CREATE TABLE from the primary arrives with ENGINE=InnoDB and runs as written. No automatic mapping: pre-create DuckDB tables on the replica.

## A primary key is required

UPDATE and DELETE find rows by their before-image — the engine needs the key for that.

WHAT WENT WRONG

# The replica that silently lied

Every transaction was accepted. The GTID moved on. Zero errors. **Zero rows, too.**

The replica writes its position to InnoDB and the data to DuckDB. Two engines means two-phase commit.

Our prepare moved the transaction into the XA registry and cleared the connection state. Commit saw nothing — and committed nothing.

```
BEGIN;
INSERT INTO duck VALUES (1,10),(2,20),(3,30);
INSERT INTO inno VALUES (1,10),(2,20),(3,30);
COMMIT;

-- duck: 0 rows
-- inno: 3 rows
```

# The fix and the lesson

```
[8]  data integrity ......... PASS
[9]  DDL replication ........ PASS
[10] transactions ........... PASS
[11] bulk LOAD DATA ......... PASS
[12] SIGKILL recovery ....... PASS

VERDICT: PASS=24 FAIL=0
```

prepare now remembers its own transaction and commit finishes that one. Regression tests: MTR txn_mixed_engine and repro-2pc-dataloss.sh.

LESSON

**Counting rows is not enough. Compare md5 of the dumps.**

After every step: dump both sides ordered by primary key and compare md5. One byte off is a FAIL.

# An honest debate

### Where the CDC camp is right

- The commit path is the riskiest place — our bug proves it
- An external client fails with lag or gaps, not a corrupted commit
- Change history and undo come from the same Parquet files

### Where a storage engine wins

- Fresh data: a replica lags by seconds
- Native MySQL protocol — BI tools work unchanged
- Plain SQL, no separate pipeline

**The question isn’t “DuckDB or not”, but “where should the column store live?”**

BEING HONEST

# Limitations

- Experiment, not production: it patches mysqld
- Replica tables must be created by hand as ENGINE=DuckDB
- The row-by-row applier can’t keep up with bulk writes on the primary
- A prepared transaction lives only in memory: a small window between prepare and commit
- Not tested yet: multi-source, replication filters, real write rates

OPEN SOURCE · GPLV2

# Try it yourself

```
docker run -d -p 3306:3306 \
  -e MYSQL_ROOT_PASSWORD=secret \
  perconalab/ducksdb-mysql-engine:9.7-duckdb-v0.2.2
```

**github.com/Percona-Lab/ducksdb-mysql-engine**

Code, patches and the benchmark harness. Issues and PRs welcome — especially your queries where the engine backs off.



Repository

# Thank you!

## Questions?

**Evgeniy Patlan · Percona**

[contact: email / LinkedIn / Telegram]

Percona blog:
Running DuckDB as a MySQL 9.7 storage engine
Replicating from InnoDB into a DuckDB storage engine
The DuckDB MySQL engine at 500 GB