MYSQL · PARQUET · DUCKDB

Poor man’s lakehouse

DuckDB + Parquet

Historical analytics and MySQL recovery with DuckDB + Parquet.

Daniel Guzman Burgos dbtrail

~~MySQL: 22 min.~~

Parquet: 4.6 s.

MYSQL PARQUET DUCKDB

```
$ duckdb
D CREATE VIEW order_line AS
    SELECT * FROM 's3://your-bucket/…/order_line1.parquet';
D SELECT ol_w_id, sum(ol_amount) AS revenue
    FROM order_line
    GROUP BY 1 ORDER BY 2 DESC LIMIT 5;

-- no server to run, no pipeline, no warehouse.
-- the primary never saw this query.
```

Your app asks about now. The business asks about the past.

THE BUSINESS ASKS ABOUT THE PAST



“Revenue by month, 3 years?”

finance



“Churn after the price change?”

product



“This account, as of 3 March?”

support · compliance

on your primary

YOUR OLTP PAYS FOR IT



Hot pages evicted

a full scan pushes out your working set



Purge and vacuum stall

a long read keeps old row versions alive



Temp tables on disk

GROUP BY over millions of rows

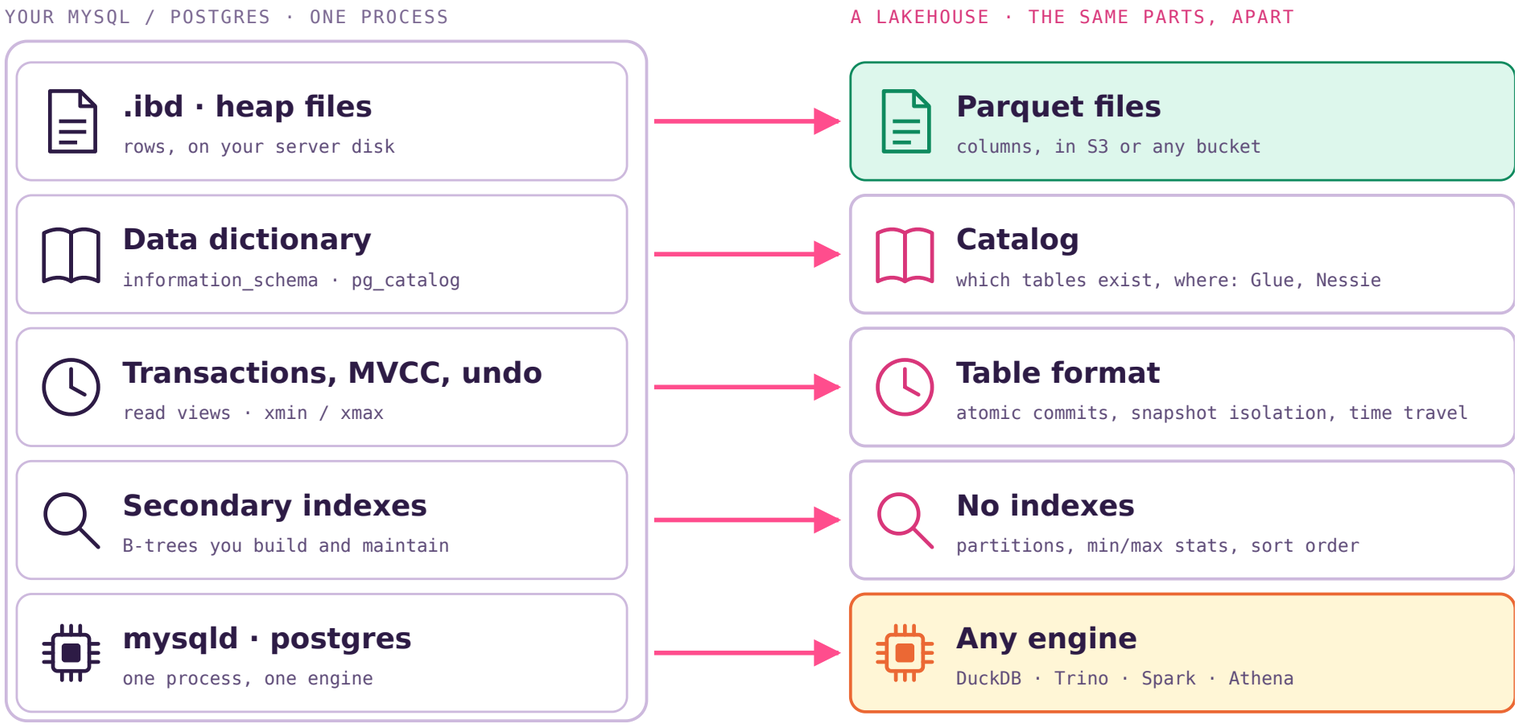
Row stores answer “now” fast; “over time” is a different shape of question. Give them a copy they can’t hurt.

A DATABASE, TAKEN APART

Lakehouse, for DBAs.

Your tables as open files in cheap object storage, plus a thin layer that makes those files behave like tables, queried by **whatever engine you bring**.

Every part of your OLTP database is still there. It just stopped living in one process.



An analogy, not an equivalence: snapshot isolation is per table, not per row, and snapshots live until a retention policy expires them, closer to your binlog than to undo.

BEFORE THE ARCHITECTURE

What is a Parquet file?

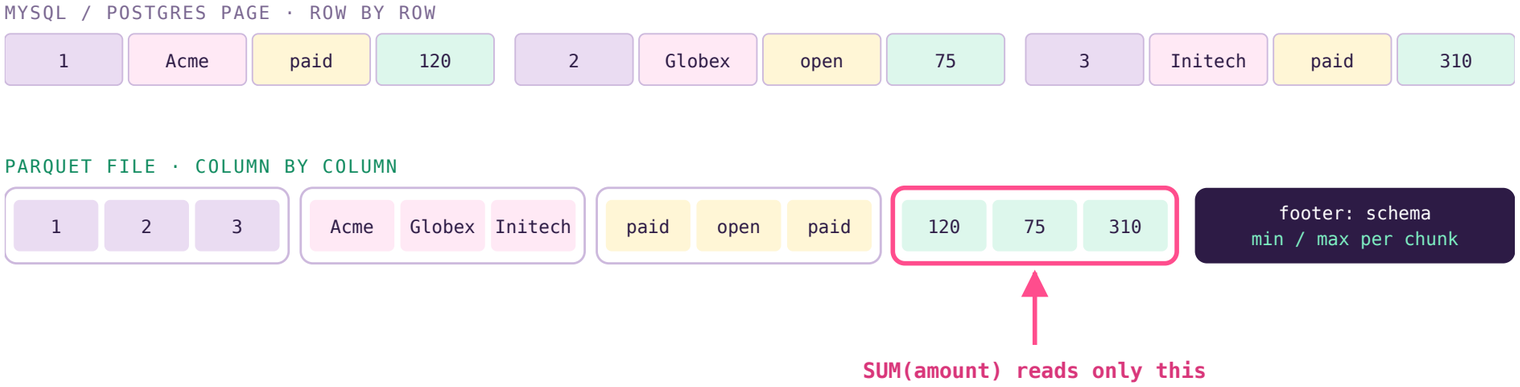
A file format for tables, stored **column by column** instead of row by row.

- **Reads less.** A SUM touches only its column.
- **Smaller.** Similar values sit together and compress: 10.3 GB in MySQL, 0.7 GB per snapshot in our test.
- **Skips.** The footer keeps min/max per chunk, a very coarse index.
- **Open.** DuckDB, Spark, pandas, Athena, ClickHouse read it.

YOUR TABLE

id	customer	status	amount
1	Acme	paid	120
2	Globex	open	75
3	Initech	paid	310

Same three rows.
Two ways to lay them on disk.



OPEN SPEC (APACHE) · COMPRESSED PER COLUMN · WRITTEN ONCE, READ MANY

You don’t UPDATE a Parquet file; you write a new one. That is why dbtrail folds changes into a fresh snapshot every five minutes.

Same promise, fewer moving parts.

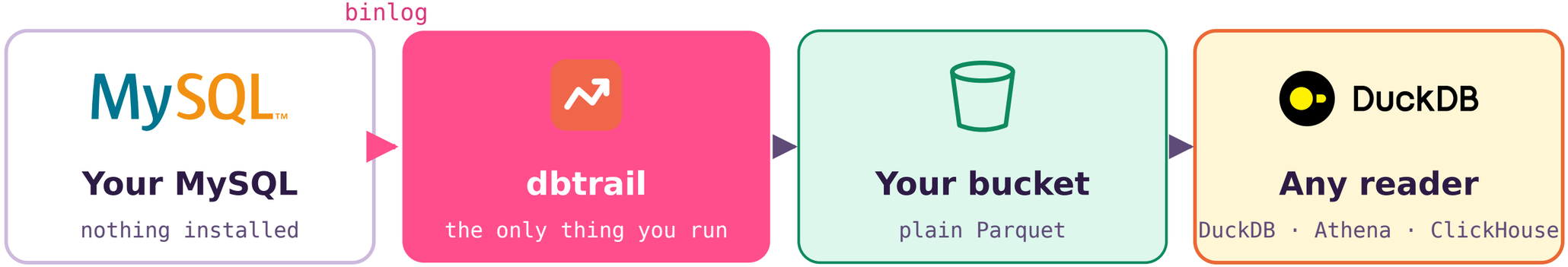
A LAKEHOUSE, THE USUAL BUILD

3-5



THE POOR MAN’S VERSION

1



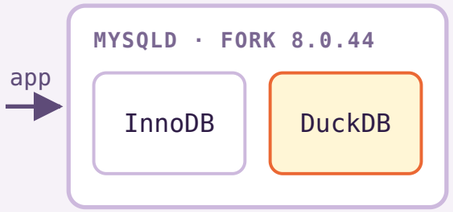
Stream and catalog depend on the build: Debezium Server, Airbyte and PeerDB run without Kafka, and DuckDB reads files by path with no catalog at all. What the poor man gives up is on “Where it stops”.

THE IDEA IS EVERYWHERE

Everyone wants DuckDB next to MySQL.

OPEN SOURCE

AliSQL

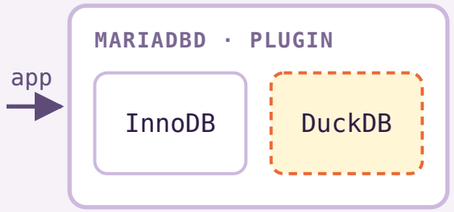


Replays its own ROW binlog into columns and folds to one final row per key.

runs where you can run AliSQL

ALPHA, NOT IN RELEASES

MariaDB

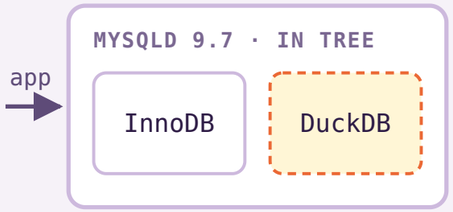


Tables you create as ENGINE=DuckDB. InnoDB and DuckDB in one query.

runs where you can load an alpha plugin

EXPERIMENTAL

PERCONA

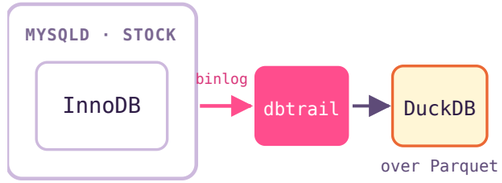


A second mysqld fed by row-based replication. You create its tables as ENGINE=DuckDB by hand; row changes then land in DuckDB, and mysqld hands each query to it.

experimental, v0.2.3 · runs where you can build MySQL 9.7 from their tree

APACHE 2.0, SHIPPING

dbtrail



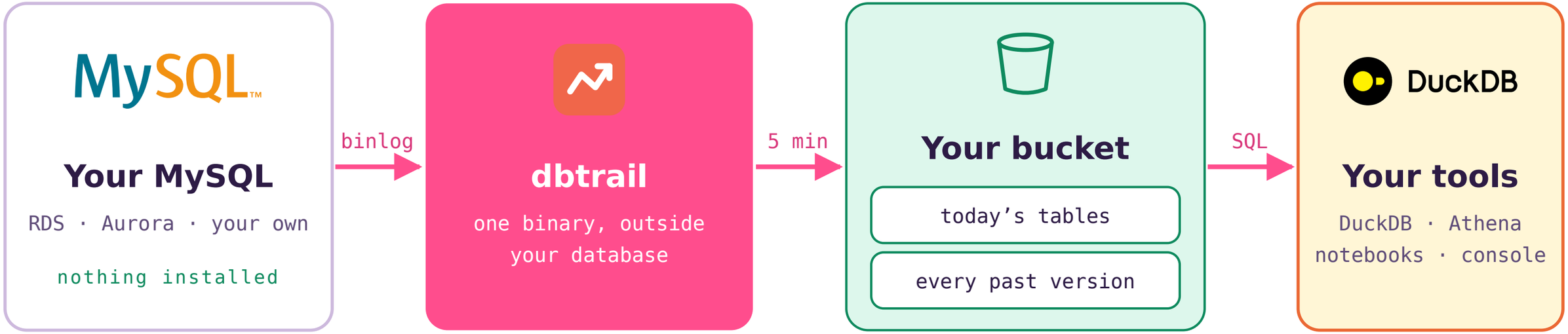
Nothing inside the server. The Parquet written for recovery is the Parquet DuckDB reads.

runs on RDS and Aurora, where you cannot load an engine

Three of the four change mysqld. Same idea, different place to put it.

WHAT IT IS

Your MySQL, as Parquet, in your bucket.



5 minutes fresh

a new snapshot, all day

Every version kept

any row, any second

Primary untouched

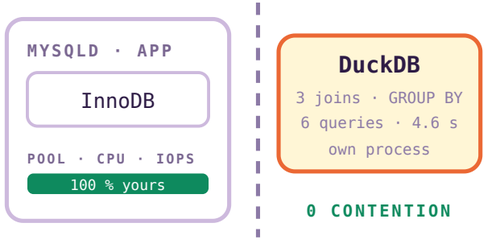
reports read the copy

Day one: one copy with mydumper. After that, only the binlog. The files are plain Parquet and open with nothing of ours running.

WHAT YOU DO WITH IT

Four things you do with the copy.

REPORTS

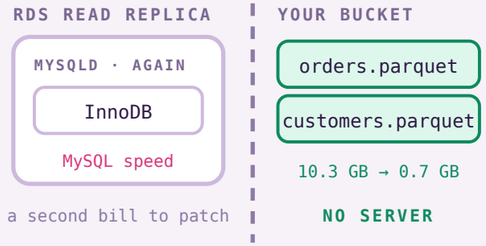


The ugly query runs somewhere else

Not inside mysqld. A separate process, over the copy. It never competes with your transactions for buffer pool, CPU or IOPS.

22 min on RDS · 4.6 s on the copy · primary did 0 work

REPLICA

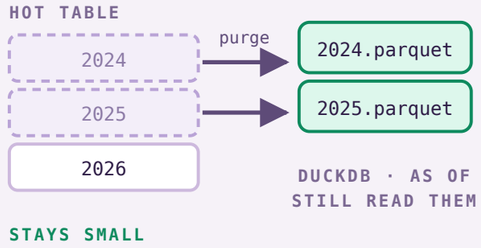


Retire the reporting replica

A read replica is another MySQL to patch, answering reports at MySQL speed, with no history. This is files in a bucket.

not a server to patch

TABLES

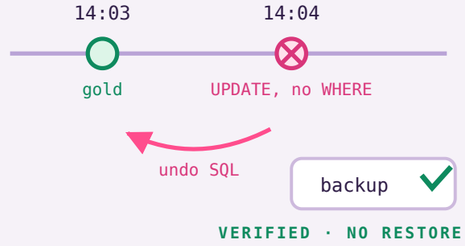


Shrink InnoDB

Purge last year from the hot table. The copy and AS 0F still have every row, at every date.

unbounded tables stay small

HISTORY



Undo and verify

The UPDATE without WHERE, the DELETE that cascaded: the SQL that reverses it. And the backup checks itself, no restore.

every version of every row

The first two are why you install it. The last two come with the copy.

HISTORICAL ANALYTICS

22 minutes on MySQL. Seconds on any copy.

ONE OF THE SIX

PLACEHOLDER

```
SELECT o.o_d_id,
       DATE_FORMAT(ol.ol_delivery_d, '%Y-%m') AS month,
       SUM(ol.ol_amount) AS revenue,
       COUNT(*)          AS lines
FROM   order_line1 ol
JOIN   orders1 o
  ON o.o_w_id = ol.ol_w_id AND o.o_d_id = ol.ol_d_id
 AND o.o_id   = ol.ol_o_id
GROUP BY 1, 2 ORDER BY 1, 2;
```

MYSQL

3 min 52 s

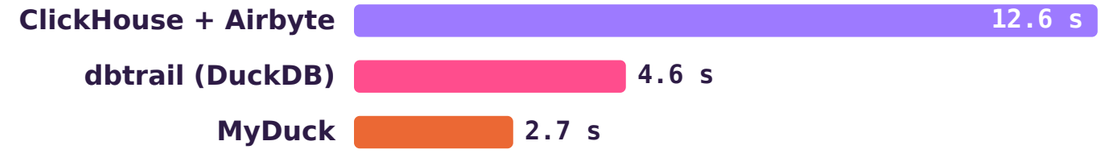
DBTRAIL · DUCKDB

0.9 s

ALL FOUR, SAME SCALE · TOTAL FOR SIX QUERIES



THE THREE COPIES, ZOOMED IN



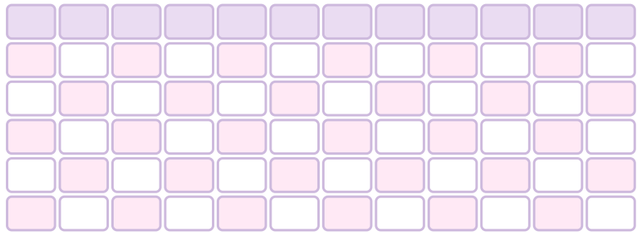
Six queries, median of three runs, source quiet.

BEFORE THE NUMBERS

What is ClickBench?

sysbench · TPC-C is to OLTP what **ClickBench** is to analytics.

THE DATA



1 table: hits

~100 M rows × 105 columns

real web-analytics traffic, anonymized

THE WORK

```
COUNT(*)
COUNT(DISTINCT UserID)
GROUP BY RegionID … LIMIT 10
WHERE URL LIKE '%google%'
```

43 queries

scans, filters, GROUP BY, top-N

THE CLOCK

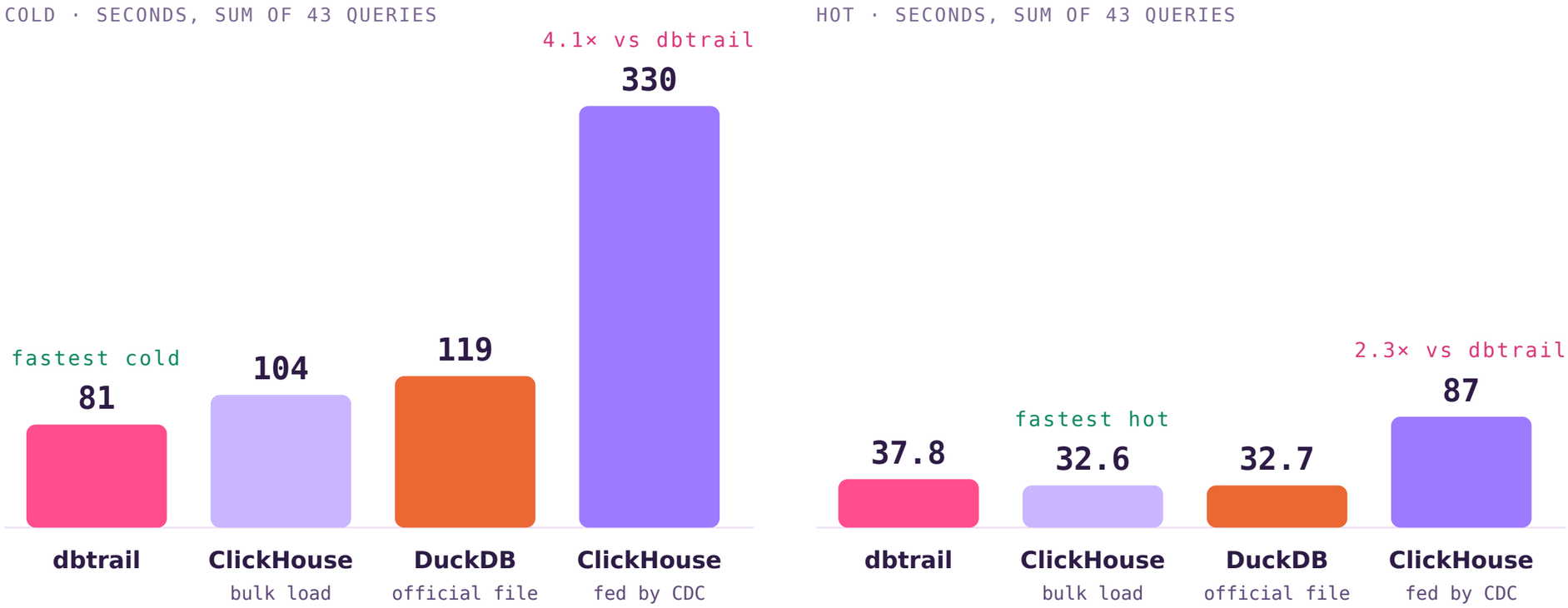
run 1	caches dropped	COLD
run 2	caches warm	
run 3	caches warm	

Cold = first run.

Hot = best of runs 2 and 3.

Made by the ClickHouse team, open to any engine, results public. By its own README it covers only this shape of workload, so it says nothing about OLTP.

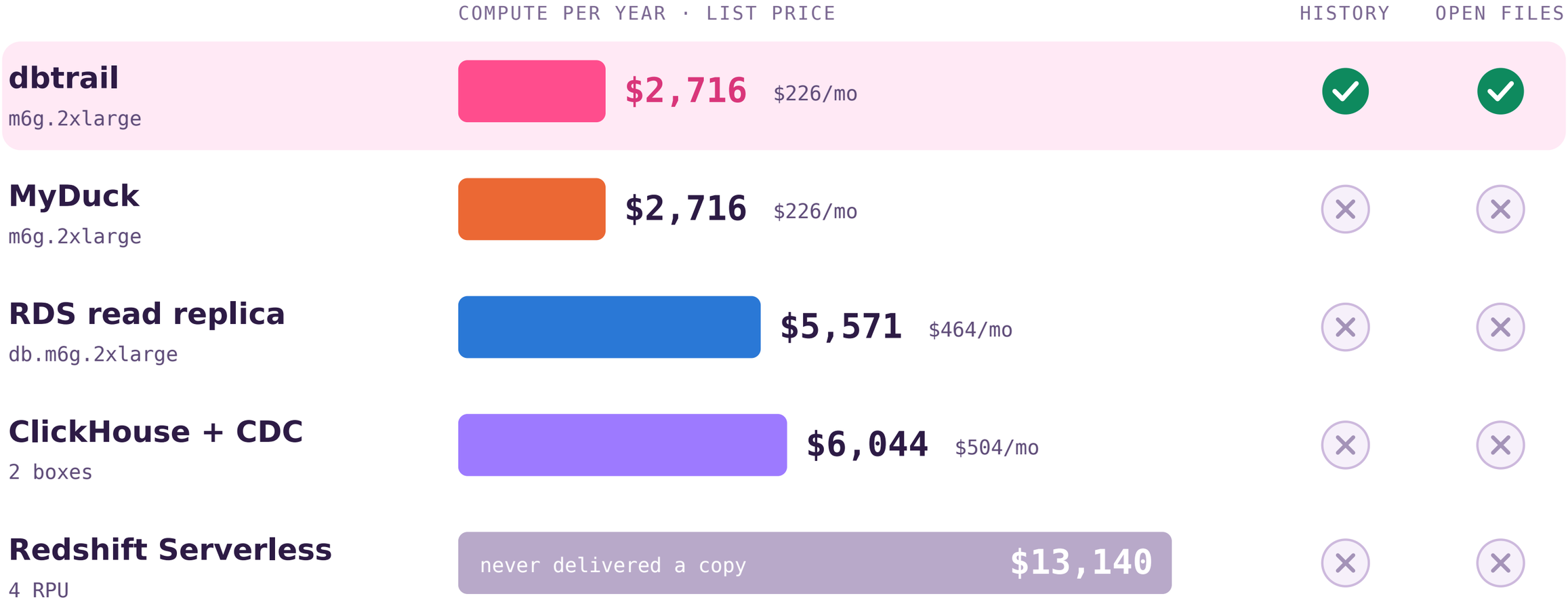
Our Parquet against ClickHouse.



Both ClickHouse rows hold the same rows. One took the published file **in one shot**; the other received them **in small batches, the way CDC delivers them**. That costs **4.2 GB on disk and 2.8× cold**: ordering, part count and the column types the connector picked. We did not isolate which.

One c6a.4xlarge · ClickHouse 25.8 · 20 Sep 2026 · our DuckDB control: 32.7 s vs 32.5 s published. The bulk-loaded table is a ceiling, not a copy: it does not follow MySQL.

What does the copy cost to run?



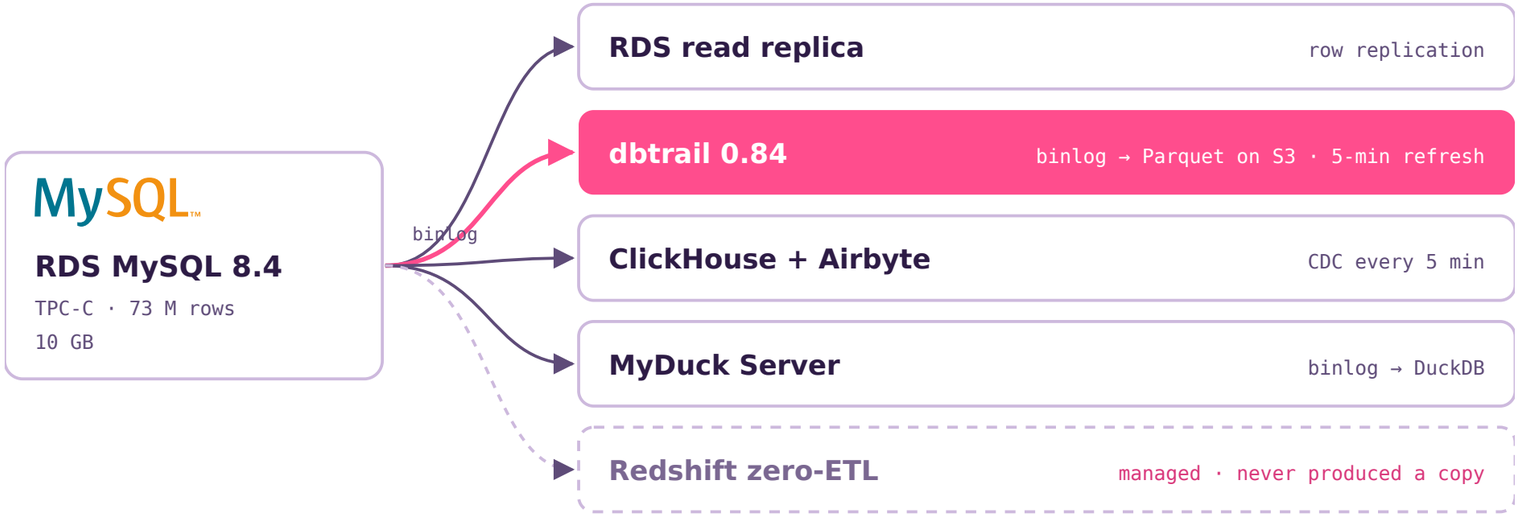
On-demand us-west-2, compute only; the source RDS is the same in every case. The replica is priced at the box that matches the others: the one we measured was a db.m6g.large, $117 a month, and that is the machine behind its lag on the freshness chart.

THE DUEL · HOW IT WAS MEASURED

One source. Five copies. One clock.

Same load, same queries, same day. One system on the binlog at a time.

One consumer per 30–60 min window; initial copies timed apart; every default kept, every fix counted as hands. Measured 18–20 Sep 2026.



Heartbeat clock

1 row/s · freshness = seen − committed

Load generator

sysbench-tpcc · 180 tps · p95 on source

Six analytical queries

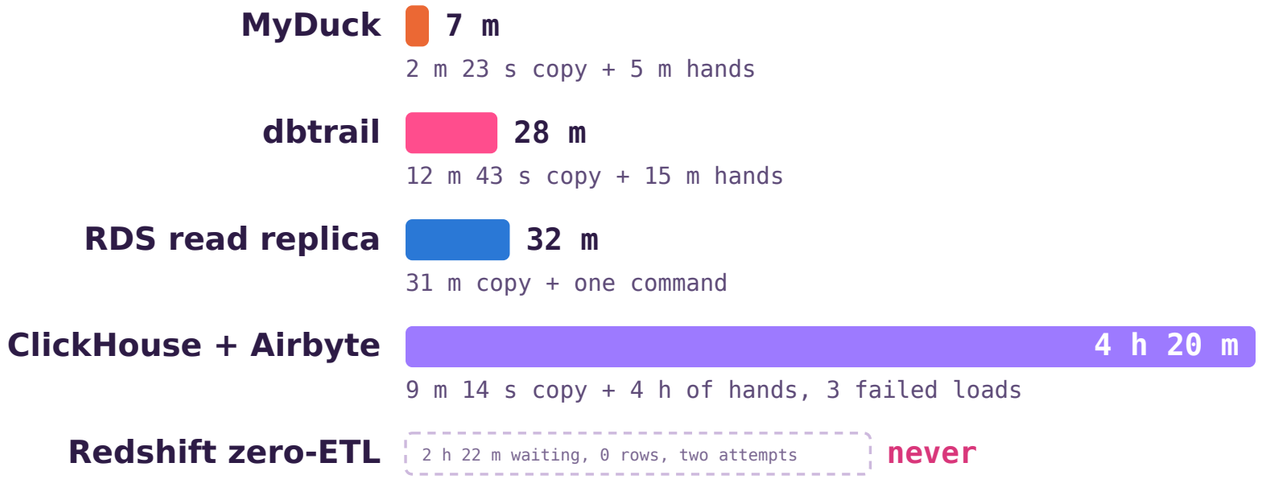
source quiet · identical SQL

THE DUEL · FIRST COPY

How long until there is a copy?

Wall clock until the first usable copy: **hands plus copy time**. Source quiet during every copy.

Redshift’s integration went active twice and never accepted CREATE DATABASE (“Unknown integration ARN”): 41 min to create, plus an unrequested backup of the source, each attempt.



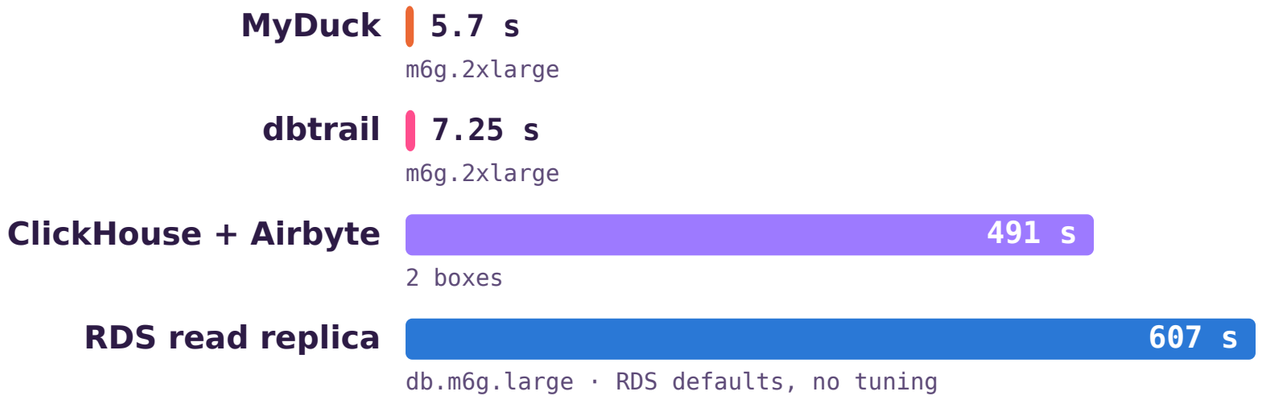
THE DUEL · FRESHNESS UNDER LOAD

How far behind is the copy?

Two copies stay **within seconds**. Two drift **8 to 10 minutes** behind. p95 of the heartbeat.

Each ran on the box its own path suggests, **not on matched hardware**.

180 tps · 30–60 min windows · probe every 5 s. Read the replica’s bar as what an untuned db.m6g.large did here, not as a claim about MySQL replication: more CPU, more IOPS and writeset dependency tracking would all move it.



THE DUEL · LOAD ON THE SOURCE

Does copying hurt the source?

Reading the binlog: **no**. p95 of the TPC-C transactions on the source, with each copy attached.

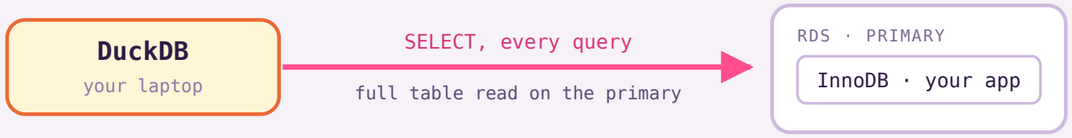
sysbench-tpcc, 16 threads, 180 tps, same instance and hour of day. One exception measured: 111 ms while a lock-all full backup ran, triggered by a refresh rule of ours, since fixed.

Nothing connected	**38.3 ms**
RDS read replica	**37.6 ms**
MyDuck	**38.3 ms**
ClickHouse + Airbyte	**38.3 ms**
dbtrail	**39.7 ms**

THE FIRST QUESTION AN ENGINEER ASKS

Why not just ATTACH?

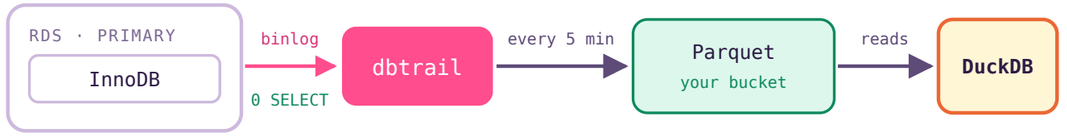
ATTACH is a DuckDB feature: it mounts your MySQL as a schema, so prod.orders is one more DuckDB table, read from MySQL on every query.




```
D ATTACH 'host=rds…' AS prod (TYPE mysql);
D SELECT … FROM prod.orders;  -- reads RDS, now
```

- ✕ **Every query competes with your app**
  "read directly from MySQL at query time": same buffer pool, same CPU.
- ✕ **The copy the docs suggest is by hand**
  A full scan, stale when it finishes, redone from production.
- ✕ **No yesterday**
  One state, the one you copied.

dbtrail: copy once, then only the binlog. DuckDB reads Parquet in your bucket and never talks to MySQL.




```
15:40  fold   0 SELECT on your tables
D SELECT … FROM state_demo_orders;  0.5 s
```

- ✓ **The primary only ships its binlog**
  What it already does for any replica. No contention with your app.
- ✓ **Fresh every 5 minutes, by folding**
  Never a dump. Untouched tables carry forward.
- ✓ **Every version of every row**
  The state at 14:03, the SQL to undo 14:04.

Small tables, occasional queries, no history needed: use the extension. It is free and good at that.

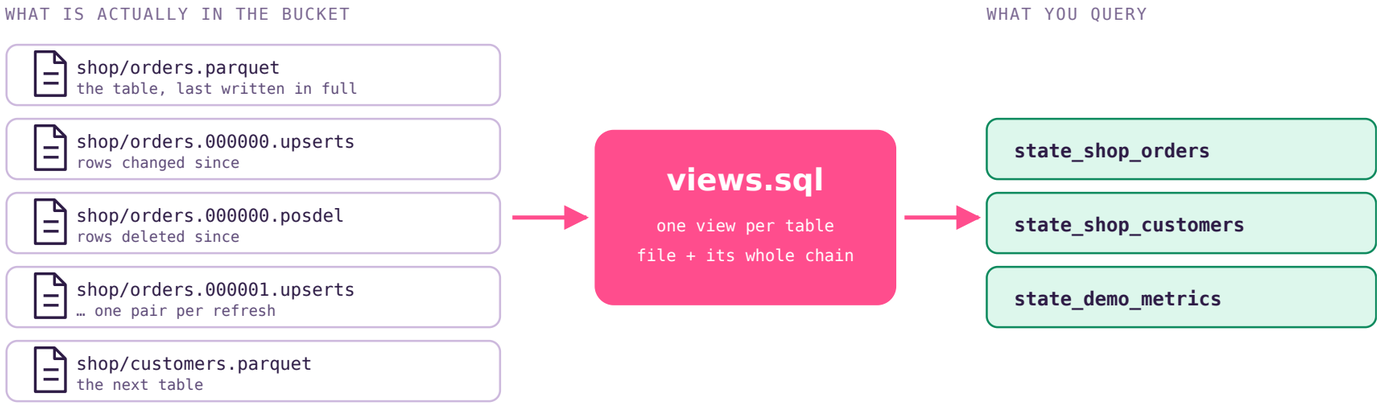
ONE STEP IN BETWEEN

Download a DuckDB schema.

A table is not one file. It is the table file, plus what changed and what was deleted since.

The views do the merge, so your SQL is just FROM state_shop_orders.

Download it again when a table or a column changes.



CONSOLE → CONNECT AI → DOWNLOAD A DUCKDB SCHEMA

```
$ duckdb -init views.sql lake.db
D SELECT count(*) FROM state_shop_orders;
```

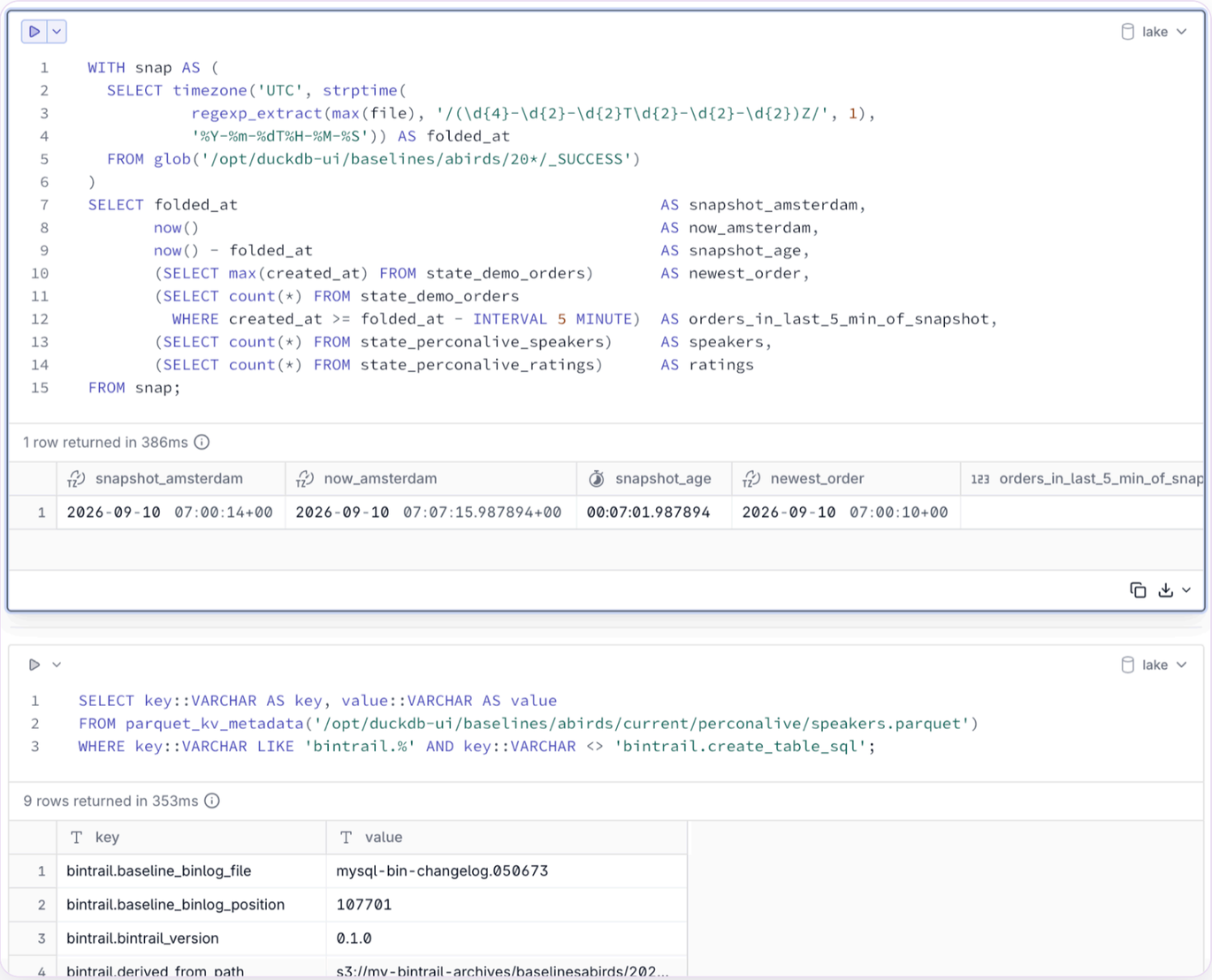
THE SAME PARQUET, IN A NOTEBOOK

DuckDB

Open it with DuckDB.

The snapshot the fold just wrote is a directory of Parquet files. **duckdb -ui** over it: one view per table, and the footer of every file names the binlog position it stands on.

Every five minutes the views move to the newest snapshot. Nothing on this page talks to MySQL.



THE QUESTION EVERYONE ASKS

What is Iceberg?

A table made of a list. Parquet files are boxes, and nobody opens a box again. A metadata file is the packing list that says which boxes are the table right now.

Every write adds boxes and publishes **a new list**, then moves **one pointer** to it. Old lists stay: those are the snapshots.

Why a new list instead of editing the old one: an object store has no append, a half-written list would be read by somebody, and the old lists are the history.



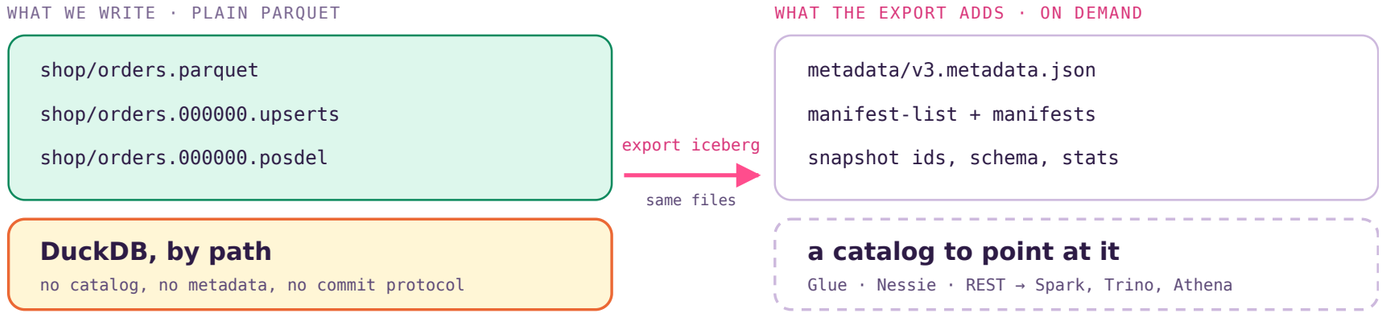
What that buys: **many writers at once**, schema evolution, and snapshots you can go back to.

Iceberg itself reads nothing. It is a format, not a process. Something has to write it, and that something reads your binlog.

… AND WHY WE SKIP IT

Not here. One command away.

- **One writer here.** The commit protocol settles a fight we do not have.
- **It wants a catalog.** A service to run and back up, the piece this talk removes.
- **Its time travel is per commit, per table.** Ours is per second, per row.
- **It cannot undo twelve rows.** It rolls the whole table back to a snapshot.

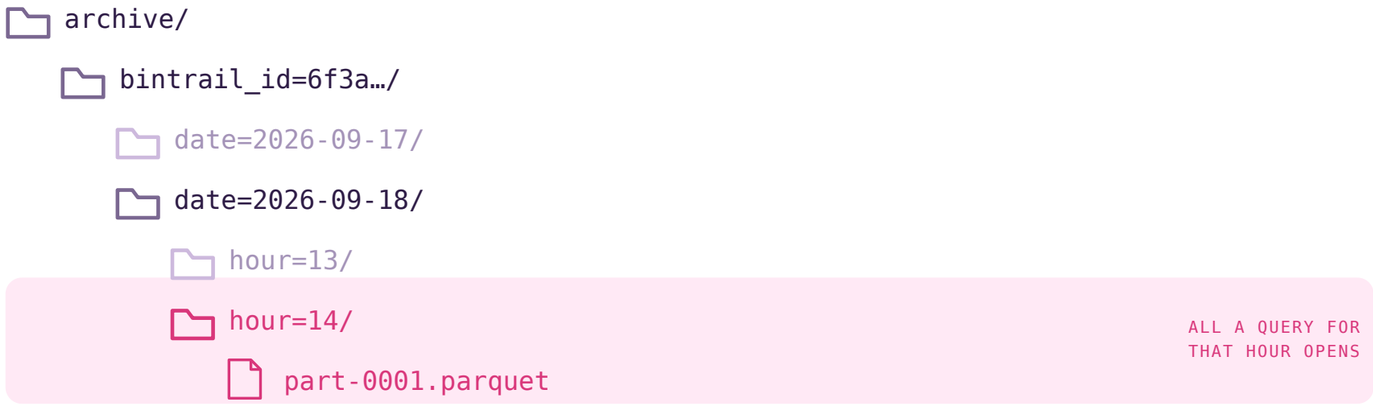


Export it the day someone else’s stack needs a table instead of a path: several writers, a platform team that requires a catalog, an engine that only speaks Iceberg. The files underneath do not change.

THE LAYOUT

Why the folders are named key=value.

- **The path is the index.** A filter on the keys skips whole folders without opening a file.
- **The keys come back as columns.** One flag, no catalog, no DDL.
- **Every engine prunes the same way.** DuckDB, Athena, Spark, Trino.



HIVE PARTITIONING, IN ONE FLAG

```
SELECT count(*) FROM read_parquet(
    's3://your-bucket/archive/**/*.parquet', hive_partitioning = true)
WHERE date = '2026-09-18' AND hour = 14;
```

One prefix per server, so two sources share a bucket and never collide. Snapshots go the other way: one folder per moment, because a snapshot is read whole.

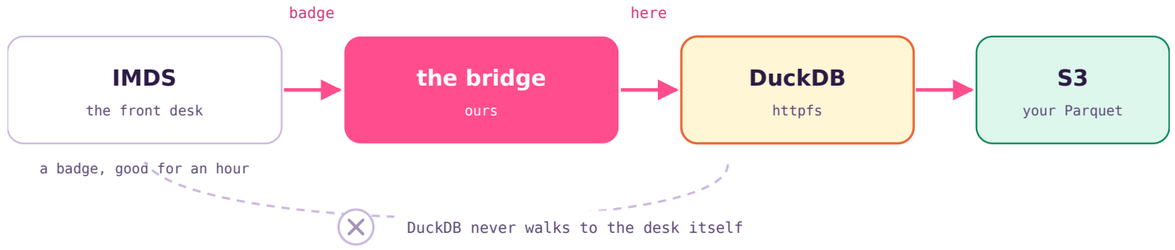
GOTCHA · CREDENTIALS

Nobody types an AWS key on a server.

A password on a box is a password you lose with the box. So AWS does not give the machine one. It runs a front desk, and the machine walks up and asks for **a badge that expires in about an hour**. That desk is IMDS.

DuckDB takes a badge. It does not go and get one. Its S3 reader wants a key handed to it; fetching the badge lives in a second extension we did not want to depend on.

So a small program of ours does the walking, hands DuckDB the badge, and goes back for a new one before the old one dies.



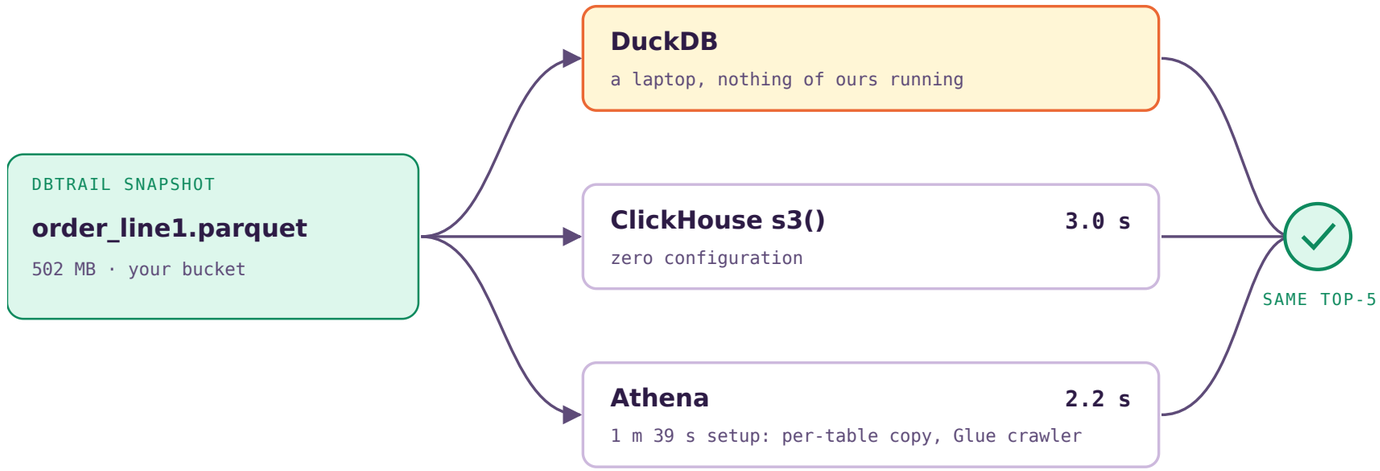
In AWS words: httpfs takes a key and a secret; instance metadata comes from the aws extension’s credential chain. Our bridge fetches and refreshes the instance role’s temporary credentials instead.

THE LAKEHOUSE PART

Other engines read the same bytes.

Same files, same answer, no conversion: **the same top-5 items over 55.6 M rows** from every engine.

Nothing else in the duel is readable by another engine: MergeTree, Redshift and MyDuck’s .db are one-engine formats.



Snapshot 2026-09-18T12-15-20Z. Caveat: table deltas add .posdel/.upserts files next to the snapshot, and an outside reader must apply them.

READ THE PAST

AS OF,
in your client.

It speaks the MySQL wire protocol. Use your own client, pointed at the time travel port.

Timestamps are UTC. Run date -u before you pick one.

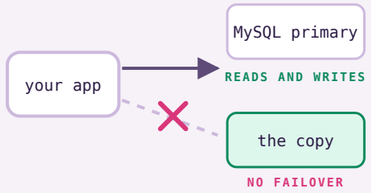
```
mysql> SELECT * FROM speakers WHERE id = 67;
Empty set (0.00 sec)

mysql> SELECT * FROM speakers WHERE id = 67
    -> AS OF '2026-08-19 09:12:00';
+----+---------------+---------+---------+
| id | name          | role    | company |
+----+---------------+---------+---------+
| 67 | Peter Zaitsev | Founder | Percona |
+----+---------------+---------+---------+
```

WHAT IT IS NOT

Where it stops.

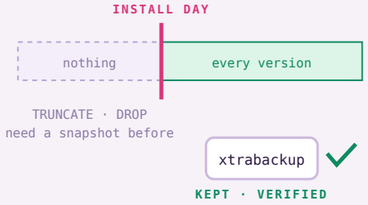
AVAILABILITY



Not high availability

The app never points here. Nothing fails over.

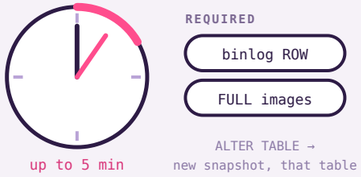
BACKUP



Not your backup

Starts the day you install it. Keep the physical one; this checks it.

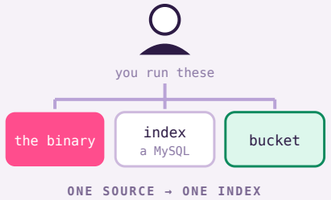
LAG



Minutes behind, not seconds

Your setting. ROW binlog with full images is the price of entry.

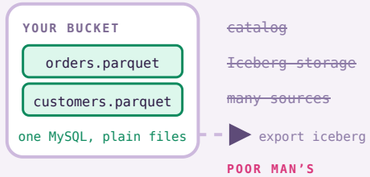
OPERATIONS



Yours to operate

Index and disk, like one more replica. One source per index.

FORMAT



A poor man's lakehouse

No catalog, no table format, one MySQL per copy. Plain Parquet in your bucket.

Said in the same voice as the strengths. A copy you can trust is one whose edges you know.

TAKE IT HOME

Apache 2.0. All of it.

Capture, index, console, recovery, the Parquet you open in DuckDB, and the MCP server that lets an assistant drive it. Every command in this talk runs on the free binary.

github.com/dbtrail/dbtrail

Daniel Guzman Burgos Questions welcome, and so are the hard ones

