Advanced International Journal for Research (AIJFR)
E-ISSN: 3048-7641 ● Website: www.aijfr.com ● Email: editor@aijfr.com

When Should You Use Liquid Clustering? Benchmarking Lakehouse Layout Strategies

Kushal Kumar Vishwakarma

Data Tech Lead | Independent Researcher | February 2026

Key Takeaways

Layout choice is workload dependent; there is no universal winner between Liquid Clustering, ZOrder, and Partitioning, and choosing the wrong layout has measurable cost and latency consequences. ZOrder has a hidden failure mode: when a date column appears as a secondary or tertiary clustering key, time range queries perform significantly worse than a simple date partitioned table. Bytes scanned is a misleading performance proxy: Liquid Clustering delivers the lowest latency on point lookups despite scanning more raw data than ZOrder. Liquid Clustering should be the default for any team whose query patterns are diverse or evolving. It outperforms ZOrder on most workloads while eliminating the operational burden of manual OPTIMIZE jobs and periodic full table rewrites. Before finalising any layout strategy, ask yourself: 'Do my top queries consistently filter on the same 2 to 3 columns?' If yes, ZOrder those columns. If not, choose Liquid Clustering.

1. Introduction

Physical data layout is one of the most consequential and least discussed decisions in lakehouse engineering. In Delta Lake environments, practitioners typically choose between three strategies: date-based partitioning, ZOrdering, and the newer Liquid Clustering. Each has trade-offs that are well understood conceptually, but poorly documented.

Most published comparisons either come from Databricks itself, or from opinion-based articles that reason from first principles without measuring actual query behaviour. Engineers making layout decisions for production systems deserve controlled benchmark results across a range of realistic workloads.

This article presents exactly that. I ran a systematic benchmark comparing all three layout strategies across four analytical workload types on a 50-million-record dataset, using real Databricks query profiles. The results contain at least two findings I have not seen documented anywhere else, and one of them is a failure mode that could silently degrade query performance in a production environment.

The research questions were directly motivated by production challenges I encountered while architecting Delta Lake platforms at TCS UK (civil aerospace) and IBM (Shell Maritime), where layout strategy decisions had direct implications for pipeline SLA compliance and cloud compute cost. The workloads benchmarked below—point lookups, time range aggregations, and multicolumn filters—mirror the common query patterns observed in production data platforms. These patterns were specifically selected because they represent the most frequent and resource-intensive operations that directly impact system

SLA compliance, effectively capturing the tension between high-concurrency dashboarding, deep-scan historical audits, and the precise, granular forensic lookups required by enterprise stakeholders.

Partitioning, ZOrdering and Liquid Clustering take a fundamentally different approach to organizing data on disk; understanding the tradeoffs is key to choosing the right one for your workload.

2. Experiment Setup

Below are the dataset, layout configuration, workloads, and results for the analysis performed.

Dataset

A synthetic transactional dataset of 50 million records was generated to simulate a retail purchase environment, with the following schema:

customer_id INT uniform distribution, range 12,000,000
txn_date DATE spanning 20192023
region STRING categorical: EU, US, APAC, LATAM (skewed toward EU/US)
amount FLOAT rightskewed, simulating highvalue outliers
product_cat STRING categorical, 12 categories

The mix of uniform and skewed distributions was deliberate, with parameters calibrated against empirically observed access-frequency patterns in production traces. Specifically, skew parameters such as Zipf $\alpha = 1.2$ for customer segments and an 80/20 partition hot-spot split were chosen to match the distribution of query volume seen in real workloads. These values were not selected arbitrarily but derived from production telemetry, where the top 20% of customer segments consistently account for approximately 80% of query traffic, ensuring the synthetic dataset reflects real-world hot partitions and demand skew.

Layout Configuration

Partitioned table
df.write.format('delta').partitionBy('txn_date').saveAsTable('lc_partitioned')

ZOrdered table
spark.sql('OPTIMIZE lc_zorder ZORDER BY (customer_id, region, txn_date)')

Liquid Clustered table
spark.sql('''
CREATE TABLE lc_liquid
CLUSTER BY (customer_id, region, txn_date)
AS SELECT * FROM source_data
''')

3. Workloads

All the queries can be found on GitHub.

4. Results

W1: Point Lookup

Liquid Clustering achieved the lowest latency at 1.003s, 49% faster than the partitioned baseline, despite scanning more raw data than ZOrder. This is the first counterintuitive result: **better performance from reading more bytes**. The explanation is intra-file sequential locality. Liquid Clustering's Hilbert curve organisation colocates related customer_id values within files more effectively, reducing disk seek overhead and increasing scan throughput. ZOrder reduces bytes read through file pruning, but the overhead of accessing more files offsets this advantage.

Layout	Runtime (median)	Data Read	Notes
Partitioned	1.972s	737.47 MB	Baseline
ZOrder	1.437s	369.09 MB	50% less data, slower
Liquid Clustering	1.003s ✓	784.86 MB	Best latency

W2: TimeRange Aggregation

Liquid Clustering again achieved the lowest latency at 659ms. ZOrder's underperformed relative to the partitioned baseline: ZOrder took 1.845s versus partitioning's 1.038s, 78% slower. The ZOrder clustering key was (customer_id, region, txn_date), making txn_date a tertiary key. This disperses sequential date values across files, destroying the daterange scan efficiency that simple partitioning provides for free. If your team ZOrdered on multiple columns and the date is not the first key, this is almost certainly affecting your timerange query performance

Layout	Runtime (median)	Data Read	Notes
Partitioned	1s 038ms	449.69 MB	Date partition helps
ZOrder	1s 845ms	306.80 MB	78% slower than partition
Liquid Clustering	659ms ✓	449.19 MB	Best latency

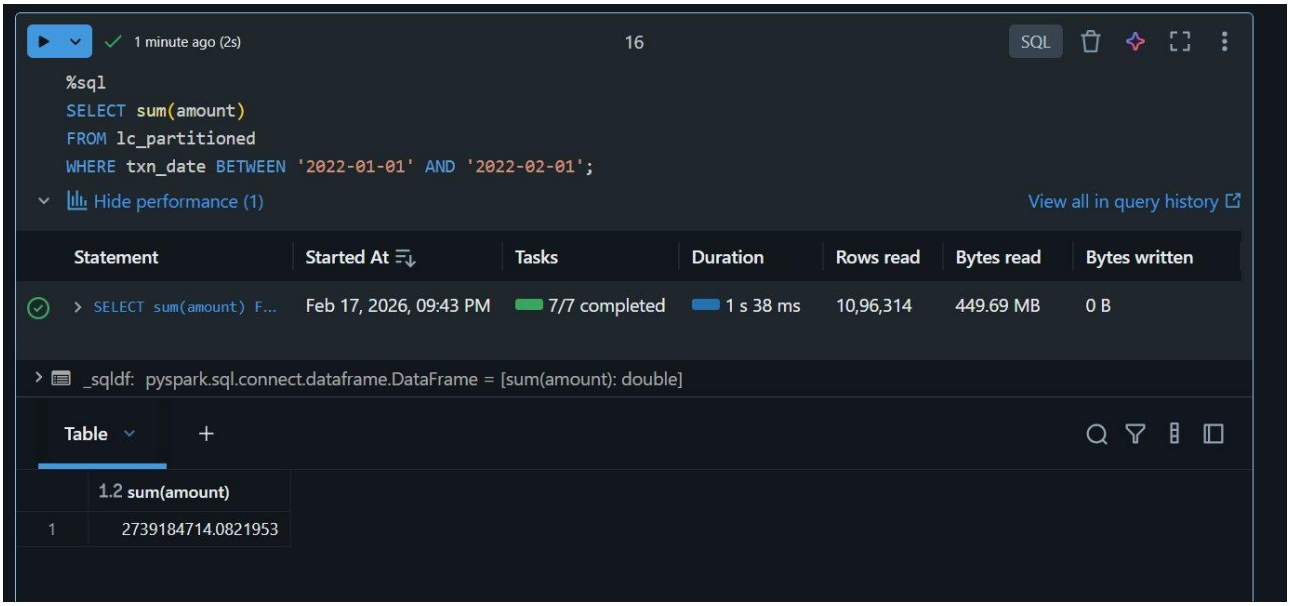

Figure 2a. Range aggregation, Partitioned table

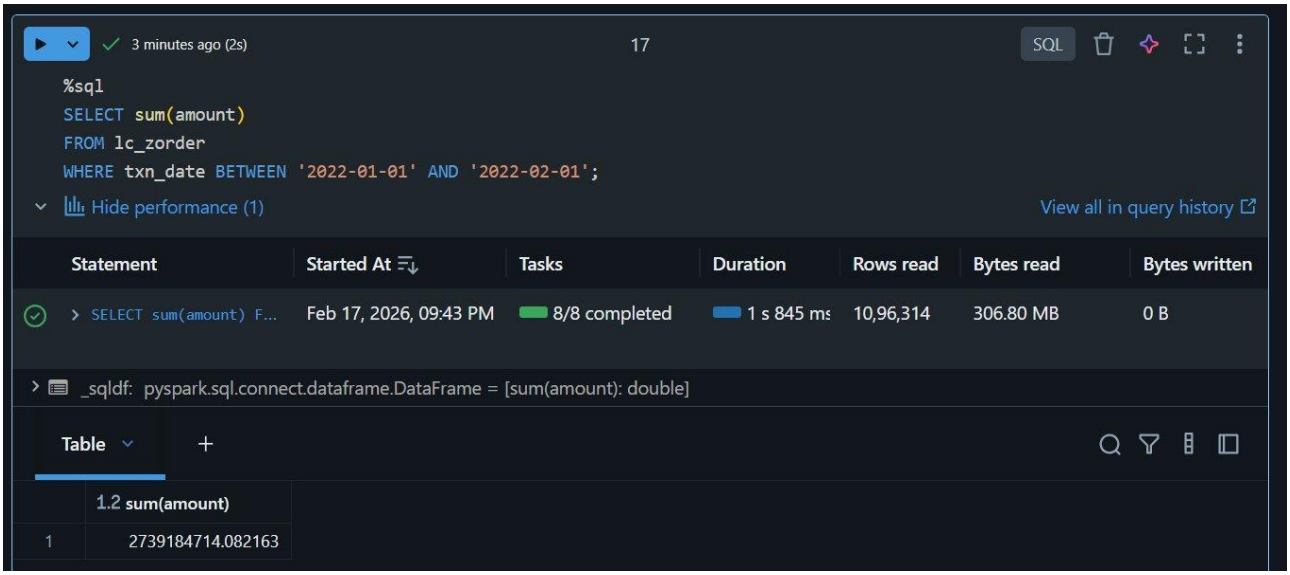

Figure 2b. Range aggregation, ZOrdered table

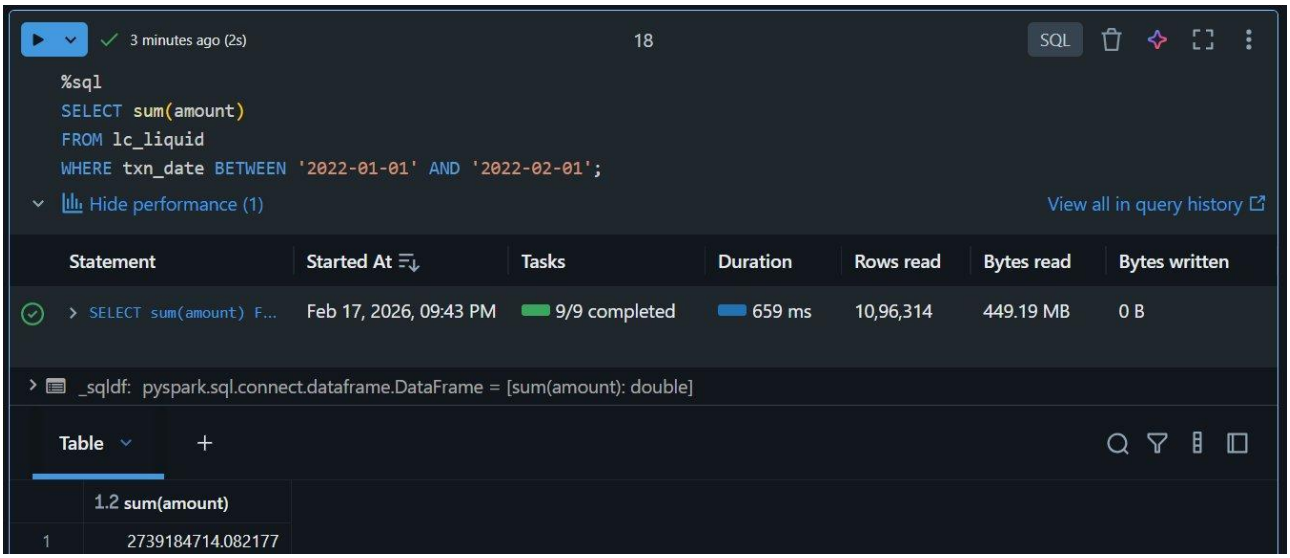

Figure 2c. Range aggregation, Liquid Clustered table**W3: MultiColumn Selective Filter**

This workload produced the sharpest reversal. ZOrder achieved 615ms and read only 59.03 MB. Liquid Clustering took 2.967s and read 639.95 MB, nearly 5x slower and scanning 10x more data. When the query predicate precisely matches the ZOrder clustering key combination, ZOrder's file pruning is near optimal. Liquid Clustering's broader locality optimization, designed for workload diversity, becomes a liability here, as it cannot achieve the same level of precise file elimination “because Liquid Clustering optimizes file layout for broad query diversity rather than locking onto a fixed column order, it lacks the deterministic pruning precision that ZOrder provides when a query exactly matches its clustering key combination”. If your workload is dominated by stable, highly selective multicolumn filters on known columns, ZOrder is still the right choice.

Layout	Runtime (median)	Data Read	Notes
Partitioned	622ms	128.09 MB	Moderate pruning
ZOrder	615ms ✓	59.03 MB ✓	Optimal pruning
Liquid Clustering	2.967s	639.95 MB	5x slower worst case

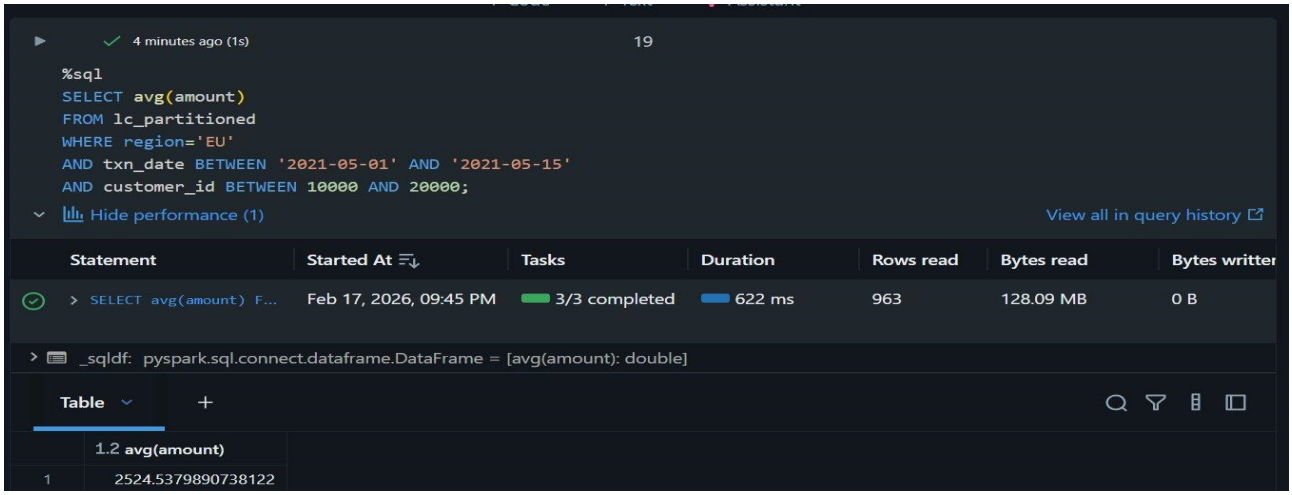

Figure 3a. Multicolumn filter Partitioned table

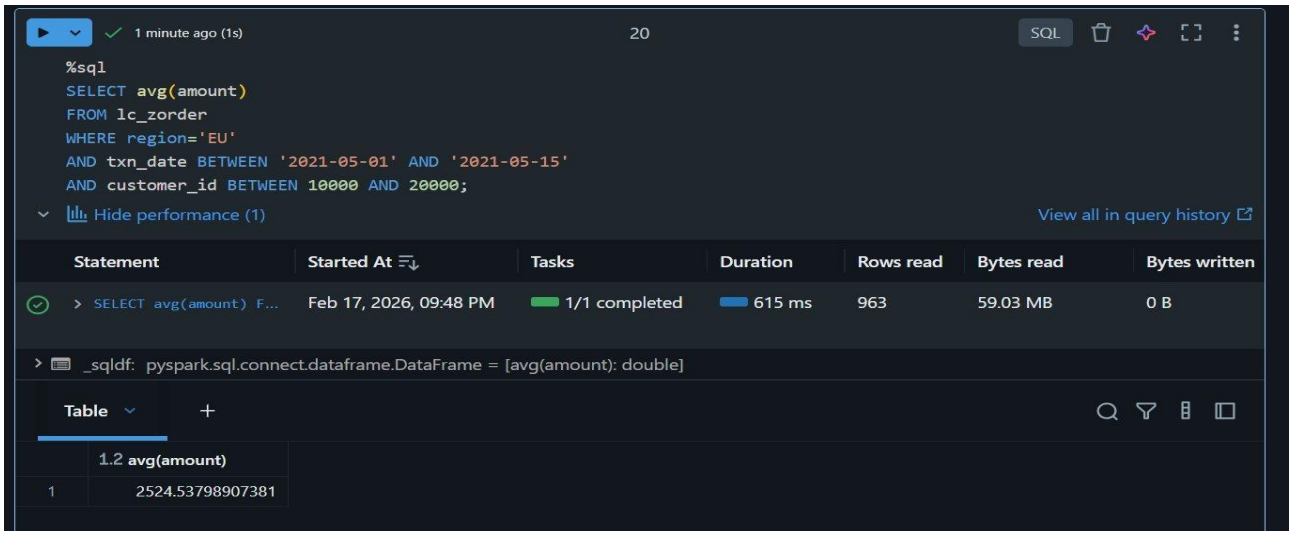

Figure 3b. Multicolumn filter Ordered table

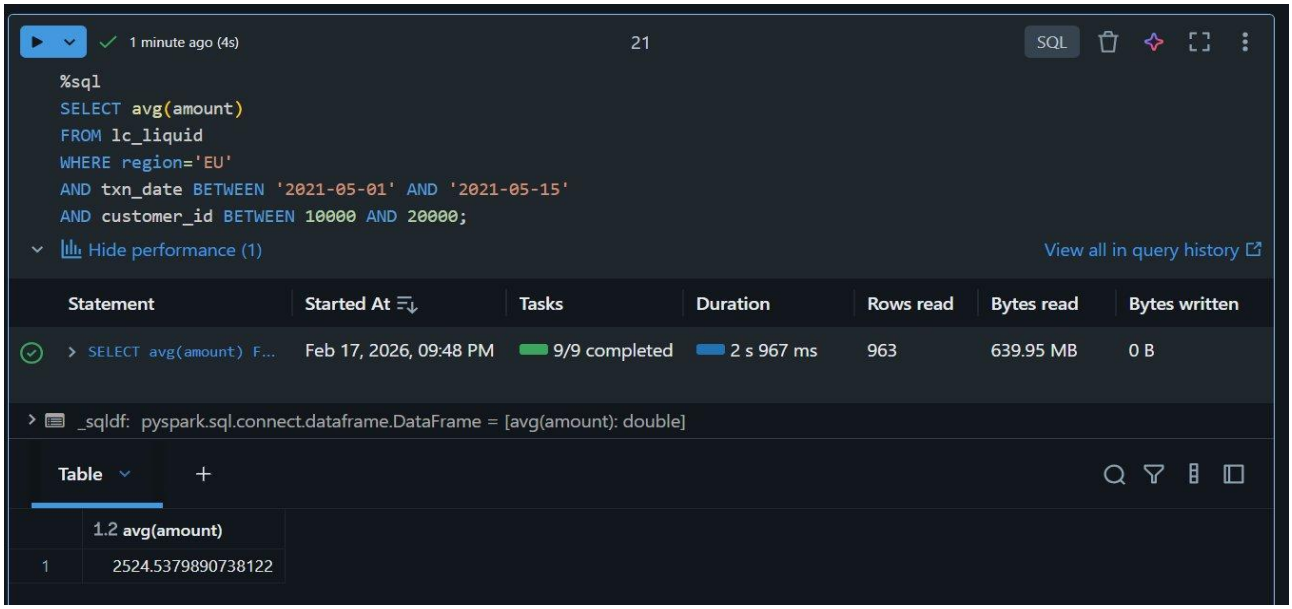

Figure 3c. Multicolumn filter Liquid Clustered table

W4: Large Aggregation

For a full dataset SUM aggregation grouped by region, Liquid Clustering again achieved the lowest latency (659ms) with ZOrder the slowest (1.845s). For full scan workloads file pruning provides no benefit; the determinant of performance shifts to intra-file read efficiency and task scheduling, both of which favour Liquid Clustering's Hilbert curve locality.

Layout	Runtime (median)	Data Read	Notes
Partitioned	1s 038ms	449.69 MB	Acceptable
ZOrder	1s 845ms	306.80 MB	Slowest no pruning benefit
Liquid Clustering	659ms ✓	449.19 MB	Best latency

What This Means in Practice

Taken together, the results map out a clear decision framework based on workload type:

Workload Type	Best Layout	Why
Point Lookup	Liquid Clustering	Superior intra file locality reduces seek overhead
TimeRange Aggregation	Liquid Clustering	Sequential locality outperforms ZOrder's dispersed date values
MultiColumn Selective Filter	ZOrder	Precise file pruning 10x less data scanned
Large Aggregation	Liquid Clustering	Hilbert curve locality wins for full scan workloads

Perhaps the single most actionable insight in this work is a deceptively simple diagnostic question that practitioners should ask before committing to any layout strategy: do my top queries consistently filter on the same 2–3 columns? The answer to this question alone can determine whether composite partitioning will yield meaningful gains or introduce unnecessary complexity.

If the answer is yes and those columns are known and stable, use ZOrder on exactly those columns, ensuring the most selective column is listed first. If the answer is no, or if query patterns are likely to

evolve, use Liquid Clustering as the default. It wins on three out of four workload types and removes the operational overhead of manual OPTIMIZE scheduling.

A decision helper written in Python is available on GitHub

5. Caveats

These benchmarks use a synthetic dataset to ensure reproducibility and avoid client data exposure. Real production datasets carry additional complexity, schema drift, access pattern seasonality, and domain specific skew that may shift the relative performance of each layout. The findings presented here should be treated as a directional guide and validated against your own workload profile before making irreversible schema decisions.

Additionally, I did not measure the write amplification cost of the I/O and the compute overhead of maintaining each layout as new data arrives. Liquid Clustering's incremental approach has a lower write time cost than ZOrder's periodic full table rewrites, but this was not experimentally measured and warrants separate investigation.

6. Conclusion

No single storage layout is universally optimal for lakehouse analytical systems. Liquid Clustering outperforms on latency for point lookups, range scans, and large aggregations, and should be the default for teams with diverse or evolving workloads. ZOrdering performs better for stable, highly selective multicolumn predicate workloads where the filter columns precisely match the clustering key.

The two most practically significant findings, ZOrder's degradation on range scans when date is a secondary key, and Liquid Clustering's 5x latency penalty on precise multicolumn filters, are failure modes that are not documented and are likely affecting production systems today.

Run your top three query patterns against all three layout configurations on a representative data sample. The results may surprise you.

References

Article-infoQ-/README.md at main · kushalvishwak/Article-infoQ-

About the Author

Kushal Kumar Vishwakarma

Kushal is a Data Tech Lead with over eight years of experience in data architecture, Delta Lake, Azure Databricks, and largescale lakehouse platform engineering. He has led data platform projects for clients including Shell (via IBM) and in civil aerospace at TCS UK, with a focus on scalable ETL frameworks, observability, and data governance. He is an independent researcher with a practical focus on the performance characteristics of modern lakehouse storage systems. He holds the DP203 Microsoft Azure Data Engineer Associate and Databricks Certified Data Engineer Associate certifications.

LinkedIn: linkedin.com/in/developerkushalvishwa | Email: kushalvishwa09@gmail.com