



Incremental View Maintenance Strategies for Lakehouse Tables Under Streaming Ingestion Workloads: A Narrative Review

Duc Tremblay^{1*}, Seoyeon Wolde², Dana Girard³

¹Department of Computer Science, Niger School of Computer Science, Japan

²Department of Computer Science, Carpathian Research Institute of Computer Science, Guyana

³Department of Computer Science, Elbe School of Computer Science, Colombia

Article Info

Volume 1, Issue 1, 2026

doi: [10.62715/ijdwdm.2026.v1.i1.pp1-7](https://doi.org/10.62715/ijdwdm.2026.v1.i1.pp1-7)

Received: 14 May 2026

Accepted: 11 July 2026

Published: 06 September 2026

Abstract

Materialized views over lakehouse tables are refreshed far more often than the warehouse views of earlier decades, because the tables beneath them now receive a commit every few seconds or minutes from streaming writers. Recomputing each view in full at that rate is wasteful, and incremental view maintenance, which derives the change to a view from the change to its inputs, is the obvious alternative. The theory is old and well developed, but it was written for engines with mutable rows, indexes and triggers, and a lakehouse offers immutable columnar files, snapshot commits and little indexing. This narrative review brings the two bodies of work together. It summarises the classical results on delta derivation, aggregate maintenance and refresh timing, describes how open table formats expose change, and then compares six maintenance strategies that range from partition-level refresh to general incrementalisation of dataflow programs. Event time, late data and exactly-once delivery are treated as constraints that every strategy must respect. The review closes with problems that the published literature leaves open, among them join deltas over unindexed tables, the interaction of maintenance with compaction, cost-based switching between strategies, and the absence of a shared benchmark.

Keywords: *Incremental view maintenance, Materialized views, Lakehouse, Open table formats, Stream processing, Change data capture*

© 2026 Duc Tremblay et al. This is an open access article under the CC BY license (<https://creativecommons.org/licenses/by/4.0/>), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

1. Introduction

A materialized view is a query result that has been stored so that later queries can read it instead of recomputing it. The price of that convenience is maintenance: when the base tables change, the stored result must change with them. Database research has studied the problem since the 1980s, when Blakeley et al. (1986) showed how to update select-project-join views from the inserted and deleted tuples alone, and the survey by Gupta and Mumick (1995) already described a mature field. A later monograph collects two further decades of results (Chirkova and Yang, 2012).

The setting in which views are maintained has since moved. Many organisations now keep analytical

* Corresponding author: Duc Tremblay, Department of Computer Science, Niger School of Computer Science, Japan. E-mail: duc.tremblay@niger.edu

© 2026 Duc Tremblay et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

data in a lakehouse: columnar files in cloud object storage, organised into tables by a metadata layer that supplies transactions, schema enforcement and versioning (Armbrust *et al.*, 2021). These tables are commonly fed by streaming jobs that commit small batches of records without pause, and downstream tables of cleaned, joined and aggregated data are expected to follow closely behind. Those downstream tables are materialized views in all but name.

Incremental maintenance suits this workload in principle, since each commit touches a tiny share of a large table. In practice the classical algorithms assume things a lakehouse does not provide: cheap access to individual rows, indexes on join keys, and triggers that fire inside the writing transaction. This article reviews what the literature offers for closing that gap, and where it remains open.

2. Scope and Approach

This is a narrative review of the published literature. It draws on three groups of sources: database research on view maintenance, research on stream processing models and engines, and the papers that describe lakehouse storage systems. Sources were chosen for their influence on the designs now in use and not through a systematic search, and no claim of completeness is made. Product documentation changes too quickly to be cited reliably and has been left aside in favour of the peer-reviewed descriptions. The review reports no measurements of its own, and where it compares strategies it does so on the basis of their published designs.

3. Classical Foundations

Three results from the earlier literature shape every strategy discussed later: how a delta is derived, which aggregates can be maintained from deltas alone, and when maintenance is carried out.

3.1. Deriving the delta

The algebraic approach treats a view as an expression over base relations and derives a second expression that yields the change to the view from the changes to its inputs. For a join of R and S , rows inserted into R must be joined with S , rows inserted into S with R , and the two sets of insertions with each other. The delta is therefore small, but computing it requires reading the unchanged side of the join. Deletions add a difficulty under projection, because a view row may have several derivations and should disappear only when the last one does. The counting algorithm of Gupta *et al.* (1993) stores the number of derivations with each view row, and Griffin and Libkin (1995) gave the corresponding rules for relations with duplicates, which is the semantics SQL engines use.

3.2. Aggregates

Aggregate views are the commonest kind in analytical pipelines. The classification introduced with the data cube operator separates distributive functions such as SUM and COUNT, algebraic functions such as AVG that reduce to a fixed number of distributive ones, and holistic functions such as MEDIAN that do not (Gray *et al.*, 1997). SUM and COUNT can be maintained under insertion and deletion if a row count is kept for each group. MIN and MAX survive insertions but not the deletion of the current extreme, which forces the group to be recomputed from the base data. Palpanas *et al.* (2002) proposed selective recomputation of the affected groups as a general answer for such functions.

3.3. Timing and consistency

Maintenance may be immediate, inside the transaction that changes the base table, or deferred to a later point, in which case the system must keep enough of a log to reconstruct what the view missed (Colby *et al.*, 1996). Zhou *et al.* (2007) described lazy maintenance, in which the work is postponed until a query needs the view or the system has idle capacity. In a warehouse that is decoupled from its sources there is a further hazard: a maintenance query sent to a source may see updates that arrived after the one being processed, and the view is then corrupted. Zhuge *et al.* (1995) identified this anomaly and corrected it with compensating queries, and later algorithms lowered the cost of the compensation (Agrawal *et al.*, 1997).

3.4. Higher-order and dataflow formulations

Two later lines of work generalised the delta idea. DBToaster observes that the delta of a query is itself a query whose result can be materialized and maintained by its own, simpler delta; applying the idea recursively replaces joins at update time with lookups in auxiliary views, at the cost of the memory those views occupy (Koch et al., 2014). Differential dataflow represents every collection as a set of differences indexed by partially ordered timestamps, which lets iterative computations be updated incrementally (McSherry et al., 2013); it runs on the timely dataflow model of Naiad (Murray et al., 2013). More recently DBSP defined incrementalisation as an operator on streams of weighted sets and showed that it can be applied mechanically to a rich query language, recursion included (Budiu et al., 2023).

4. What a Lakehouse Changes

A lakehouse table is a set of immutable data files, usually in Parquet format, together with metadata recording which files belong to each version of the table. In Delta Lake the metadata is a transaction log stored beside the data, and writers commit by adding a log entry under optimistic concurrency control (Armbrust et al., 2020). Apache Iceberg and Apache Hudi reach similar guarantees with different metadata structures, and a published comparison of the three sets out where they differ in transactions, metadata handling and update strategy (Jain et al., 2023).

Several properties of this design bear directly on view maintenance. Change is recorded first at the level of files: a commit lists the files it adds and the files it removes. For a table that only receives appends, the added files are exactly the inserted rows. For updates and deletes the format either rewrites every file that contains an affected row, known as copy-on-write, or writes the change to a side structure that readers merge with the base file, known as merge-on-read. The second approach resembles the log-structured merge-tree, in which writes accumulate in small components that are merged in the background (O'Neil et al., 1996). In neither case does the file list alone reveal which rows changed, so a row-level change feed has to be produced by the writer or reconstructed by comparing file contents.

Versioning works in favour of maintenance. Because every commit produces a named snapshot, a maintenance job can read each base table exactly as it stood at a chosen version, and the anomaly that troubled decoupled warehouses does not arise so long as the job records which versions it has consumed. Indexing works against it. Lakehouse tables rarely have secondary indexes; readers locate data through partition pruning and per-file column statistics. A join delta that must probe a large unchanged table therefore turns into a scan unless that table happens to be clustered on the join key.

Streaming ingestion sharpens these effects. Engines such as Spark Structured Streaming (Armbrust et al., 2018) and Apache Flink (Carbone et al., 2015) commit at short intervals, which leaves many small files and a long metadata history. Background compaction rewrites those files into larger ones without altering the table's contents. A maintenance process has to recognise such commits as logically empty, or it will treat every compacted row as deleted and inserted again.

5. Maintenance Strategies

The strategies described in the literature can be arranged by how much of the classical machinery they use. Table 1 summarises six of them, and Figure 1 sets out the three design choices from which each is assembled: how change is captured, how the delta is computed and when the view is updated.

Full recomputation is the baseline and remains the right answer more often than its reputation suggests. It is correct for every query, needs no change capture and is cheaper than incremental work once a large fraction of the input has changed. Partition-level refresh is the simplest improvement. When both the base table and the view are partitioned by event date, a commit that adds files to two partitions requires only those two partitions of the view to be rebuilt. The approach needs nothing beyond the file lists in the commit metadata, and it tolerates any query whose result for a partition depends only on that partition of the input.

Append-only delta processing reads only the files added since the last consumed version and applies

insertion rules. It fits raw event tables well, and distributive aggregates can be merged into the stored view directly. Its weakness is that it gives silently wrong answers when a correction or a deletion request reaches the base table, so it has to be guarded by a check on the kinds of commit seen. Row-level delta queries remove the restriction. A change feed supplies inserted rows, deleted rows and the before and after images of updated rows, and the classical rules apply with derivation counts kept in the view. The cost concentrates in joins, for the reason given in the preceding section.

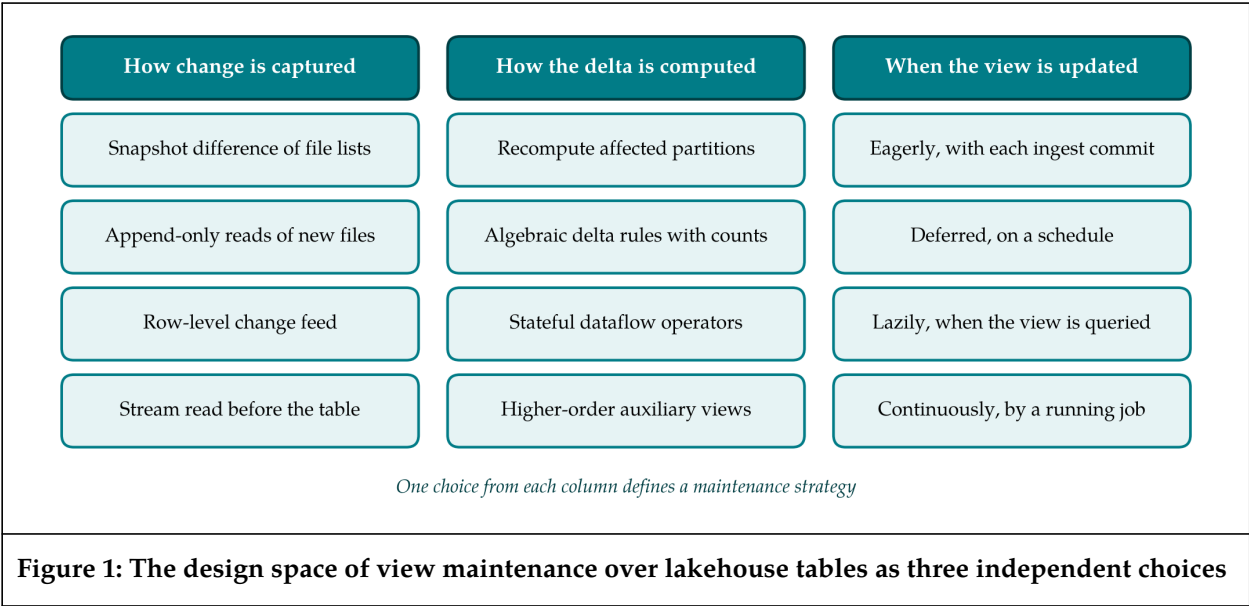
Table 1: Six maintenance strategies for views over lakehouse tables, compared by design			
Strategy	How change is used	Suited to	Main limitation
Full recomputation	Ignored; the view is rebuilt at each refresh	Any query; small tables	Cost follows table size, not change size
Partition-level refresh	Commits identify touched partitions, which are rebuilt	Views aligned with a time partition key	Late data and misaligned keys spread the rebuild
Append-only delta	Newly added files are read as insertions	Event streams with distributive aggregates	Incorrect once updates or deletes occur
Row-level delta queries	A change feed drives algebraic delta rules with counts	Joins and aggregates over upserted tables	Join deltas probe large unindexed tables
Stateful streaming dataflow	Operators hold state and emit view changes	Low-latency and windowed views	State size, recovery, agreement with batch results
General incrementalisation	Auxiliary views or weighted-set circuits are maintained	Nested and recursive queries	Storage and memory for auxiliary state

The last two strategies move maintenance out of the batch engine. In a stateful streaming dataflow the view is the output of a continuously running job whose operators keep the join and aggregation state they need, so no probe of the base tables is required. The state must be checkpointed and can grow without bound for joins unless a retention limit is set. General incrementalisation, in the manner of DBToaster or DBSP, extends the same idea to queries that streaming engines handle poorly, such as nested aggregates and recursion, by maintaining auxiliary results alongside the view. Both strategies raise a question the batch strategies avoid: whether the maintained view agrees exactly with what a batch query over the same snapshot would return.

6. Constraints Imposed by Streaming Ingestion

Streaming sources deliver records out of order. The dataflow model distinguishes the time at which an event happened from the time at which it is processed, and uses watermarks to estimate when the input for a given event-time window is complete (Akidau et al., 2015). A watermark is a heuristic, so some records arrive after their window has been declared closed. A maintained view has three options for such records: drop them, hold results back until lateness is unlikely, or emit a correction. The third option requires the view to accept retractions, which returns the problem to the deletion rules discussed above. Begoli et al. (2019) argue that streams and tables should be treated as two renderings of the same time-varying relation, and that framing describes the lakehouse case closely: the table is the accumulated stream and the change feed is the stream recovered from the table.

Delivery guarantees matter as much as ordering. If a maintenance job fails after writing to the view but before recording its progress, a restart will apply the same delta twice, and a SUM will then be wrong with no outward sign of error. The remedy used with transactional tables is to commit the output and the consumed input position atomically, so that a replayed batch can be recognised and skipped (Armbrust et al., 2018). The broader point, made at length by Kleppmann (2017), is that derived data stays trustworthy only when the derivation is deterministic and can be replayed from a log.



7. Open Problems

Four problems stand out as unresolved in the published work. The first is the join delta. Without indexes, probing a large dimension or fact table for the partners of a few changed rows costs a scan, and the proposed remedies each give something up: clustering helps one join key only, auxiliary views cost storage, and streaming state costs memory. The second is the interaction between maintenance and table housekeeping. Compaction, reclustering, snapshot expiry and schema evolution all alter files or history that a deferred maintenance job may still need, and the literature offers no agreed protocol for coordinating them.

The third problem is choosing a strategy. Incremental maintenance is cheaper than recomputation only when deltas are small in proportion to the table and the query permits efficient delta rules, and both conditions vary from one commit to the next. A system should cost the alternatives at each refresh and switch freely between them, yet cost models for delta queries over file-based tables are poorly developed. The fourth is evaluation. Decision-support benchmarks measure query speed over static data and streaming benchmarks measure throughput and latency of a pipeline; neither measures the cost of keeping a set of views fresh under sustained ingestion with updates and late records. Until a shared workload of that kind exists, comparisons between strategies will rest on design arguments like the ones in this review.

Smaller gaps deserve mention as well. Views that involve outer joins, window functions or non-deterministic expressions are often excluded from incremental support altogether. Consistency across several views that depend on the same base tables requires either multi-table commits or an explicit record of the versions each view reflects. And a view that is maintained incrementally for months accumulates the effect of any undetected error, which argues for periodic reconciliation against a full recomputation as a routine control.

8. Conclusion

The theory of incremental view maintenance transfers to the lakehouse more completely than the change of storage model might suggest. Delta rules, derivation counts and the classification of aggregates apply unaltered, and snapshot versioning removes a consistency hazard that earlier warehouses had to engineer around. What does not transfer is the assumption of cheap row access, and most of the open problems identified here follow from that single difference. For practitioners the literature supports a layered position: partition-level and append-only refresh where the workload allows, row-level delta queries where updates must be honoured, streaming or generally incrementalised dataflows where latency or query shape demands them, and full recomputation kept available as both a fallback and a check.

Conflict of interest

To be declared by the author(s) before publication.

Funding

To be declared by the author(s) before publication.

References

- Agrawal, D., El Abbadi, A., Singh, A. and Yurek, T. (1997). [Efficient view maintenance at data warehouses](#). *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, 417-427.
- Akidau, T., Bradshaw, R., Chambers, C., Chernyak, S., Fernández-Moctezuma, R.J., Lax, R., McVeety, S., Mills, D., Perry, F., Schmidt, E. and Whittle, S. (2015). [The dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing](#). *Proceedings of the VLDB Endowment*, 8(12): 1792-1803. <https://doi.org/10.14778/2824032.2824076>
- Armbrust, M., Das, T., Sun, L., Yavuz, B., Zhu, S., Murthy, M., Torres, J., van Hovell, H., Ionescu, A., Łuszczak, A., Świtakowski, M., Szafranski, M., Li, X., Ueshin, T., Mokhtar, M., Boncz, P., Ghodsi, A., Paranjpye, S., Senster, P., Xin, R. and Zaharia, M. (2020). [Delta Lake: High-performance ACID table storage over cloud object stores](#). *Proceedings of the VLDB Endowment*, 13(12): 3411-3424. <https://doi.org/10.14778/3415478.3415560>
- Armbrust, M., Das, T., Torres, J., Yavuz, B., Zhu, S., Xin, R., Ghodsi, A., Stoica, I. and Zaharia, M. (2018). [Structured Streaming: A declarative API for real-time applications in Apache Spark](#). *Proceedings of the 2018 International Conference on Management of Data*, 601-613. <https://doi.org/10.1145/3183713.3190664>
- Armbrust, M., Ghodsi, A., Xin, R. and Zaharia, M. (2021). [Lakehouse: A new generation of open platforms that unify data warehousing and advanced analytics](#). *Proceedings of the 11th Conference on Innovative Data Systems Research (CIDR)*.
- Begoli, E., Akidau, T., Hueske, F., Hyde, J., Knight, K. and Knowles, K. (2019). [One SQL to rule them all: An efficient and syntax-idiomatic approach to management of streams and tables](#). *Proceedings of the 2019 International Conference on Management of Data*, 1757-1772. <https://doi.org/10.1145/3299869.3314040>
- Blakeley, J.A., Larson, P.-Å. and Tompa, F.W. (1986). [Efficiently updating materialized views](#). *Proceedings of the 1986 ACM SIGMOD International Conference on Management of Data*, 61-71. <https://doi.org/10.1145/16894.16861>
- Budiu, M., Chajed, T., McSherry, F., Ryzhyk, L. and Tannen, V. (2023). [DBSP: Automatic incremental view maintenance for rich query languages](#). *Proceedings of the VLDB Endowment*, 16(7): 1601-1614. <https://doi.org/10.14778/3587136.3587137>
- Carbone, P., Katsifodimos, A., Ewen, S., Markl, V., Haridi, S. and Tzoumas, K. (2015). [Apache Flink: Stream and batch processing in a single engine](#). *IEEE Data Engineering Bulletin*, 38(4): 28-38.
- Chirkova, R. and Yang, J. (2012). [Materialized views](#). *Foundations and Trends in Databases*, 4(4): 295-405. <https://doi.org/10.1561/19000000020>
- Colby, L.S., Griffin, T., Libkin, L., Mumick, I.S. and Trickey, H. (1996). [Algorithms for deferred view maintenance](#). *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, 469-480.
- Gray, J., Chaudhuri, S., Bosworth, A., Layman, A., Reichart, D., Venkatrao, M., Pellow, F. and Pirahesh, H. (1997). [Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals](#). *Data Mining and Knowledge Discovery*, 1(1): 29-53. <https://doi.org/10.1023/A:1009726021843>
- Griffin, T. and Libkin, L. (1995). [Incremental maintenance of views with duplicates](#). *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, 328-339.
- Gupta, A. and Mumick, I.S. (1995). [Maintenance of materialized views: Problems, techniques, and](#)

- applications. *IEEE Data Engineering Bulletin*, 18(2): 3-18.
- Gupta, A., Mumick, I.S. and Subrahmanian, V.S. (1993). *Maintaining views incrementally. Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, 157-166.
<https://doi.org/10.1145/170035.170066>
- Jain, P., Kraft, P., Power, C., Das, T., Stoica, I. and Zaharia, M. (2023). *Analyzing and comparing lakehouse storage systems. Proceedings of the 13th Conference on Innovative Data Systems Research (CIDR)*.
- Kleppmann, M. (2017). *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*. O'Reilly Media, Sebastopol, CA.
- Koch, C., Ahmad, Y., Kennedy, O., Nikolic, M., Nötzli, A., Lupei, D. and Shaikhha, A. (2014). *DBToaster: Higher-order delta processing for dynamic, frequently fresh views. The VLDB Journal*, 23(2): 253-278.
<https://doi.org/10.1007/s00778-013-0348-4>
- McSherry, F., Murray, D.G., Isaacs, R. and Isard, M. (2013). *Differential dataflow. Proceedings of the 6th Biennial Conference on Innovative Data Systems Research (CIDR)*.
- Murray, D.G., McSherry, F., Isaacs, R., Isard, M., Barham, P. and Abadi, M. (2013). *Naiad: A timely dataflow system. Proceedings of the 24th ACM Symposium on Operating Systems Principles*, 439-455.
<https://doi.org/10.1145/2517349.2522738>
- O'Neil, P., Cheng, E., Gawlick, D. and O'Neil, E. (1996). *The log-structured merge-tree (LSM-tree). Acta Informatica*, 33(4): 351-385. <https://doi.org/10.1007/s002360050048>
- Palpanas, T., Sidle, R., Cochrane, R. and Pirahesh, H. (2002). *Incremental maintenance for non-distributive aggregate functions. Proceedings of the 28th International Conference on Very Large Data Bases*.
- Zhou, J., Larson, P.-Å. and Elmongui, H.G. (2007). *Lazy maintenance of materialized views. Proceedings of the 33rd International Conference on Very Large Data Bases*, 231-242.
- Zhuge, Y., Garcia-Molina, H., Hammer, J. and Widom, J. (1995). *View maintenance in a warehousing environment. Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, 316-327. <https://doi.org/10.1145/223784.223848>