Completing the Composable Data Stack: Rise of the Storage Layer

Pascal Ginter
Technical University of Munich
pascal.ginter@tum.de

ABSTRACT

Traditionally, composable data management systems are divided into three layers: the language frontend, the query optimizer, and the query engine/runtime [7]. Between these layers, a common intermediate representation (IR) is used to exchange query plans in terms of relational algebra (e.g., Substrait [8]). One layer that is notably absent from this classification is the storage layer. In practice, industry has converged on a standardized approach to data storage based on Parquet combined with a thin layer of metadata called an open table format, such as Apache Iceberg or Delta Lake. As a result, many responsibilities traditionally associated with data storage have been absorbed into the query engine. Although libraries for these formats are available, many query engines have implemented their own integration with the standard data format in the pursuit of performance, leading to a tight coupling between query engine and table format.

Yet, there is a need for specialization in the storage layer. With the increase in network bandwidth in the public cloud, Parquet's reliance on general-purpose compression has caused decompression to become the new bottleneck, leading to the development of lightweight encoding schemes tailored to columnar data [4]. Additionally, new file formats have been developed for specialized workloads. For instance, machine learning workloads must handle large columns, which motivated Lance [6], while the need for random data access spurred the development of Vortex [1].

Due to the tight integration between query engine and data storage, the query engine must handle not only decoding, but also access path selection, caching frequently accessed data, durability of freshly ingested data, concurrency control, and retry strategies in the case of failed commits. All of these have traditionally been handled by a separate storage layer in data management systems, Even modular storage engines like RocksDB [2] have been adopted across systems.

We argue that the query engine is the wrong component to integrate all this complexity, and that the composable stack is missing a crucial layer. We envision a dedicated storage layer that interfaces with the query engine in the following way, as illustrated in Figure 1:

(1) The query engine communicates projections and filter predicates to the storage layer through the established common IR of the composable data stack.
(2) The storage layer leverages access paths and caching to minimize I/O and decoding time.
(3) The storage layer returns the filtered data to the query engine in a common representation that is independent of the physical data format (e.g., Apache Arrow).

The storage layer could be either a standalone service [3], offering the additional benefit of interoperability between different query engines, or a library behind a common interface embedded into the query engine, a direction exemplified by Delta Kernel [5].

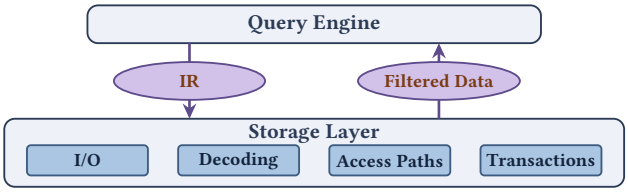

Figure 1: Components of the proposed storage layer and interaction with the query engine.

Our goal is to raise awareness of the missing storage layer in the composable data stack and to stimulate research on the design of truly composable storage components.

VLDB Workshop Reference Format:
Pascal Ginter. Completing the Composable Data Stack: Rise of the Storage Layer. VLDB 2026 Workshop: Fourth International Workshop on Composable Data Management Systems.

ACKNOWLEDGMENTS

Funded/Co-funded by the European Union (ERC, CODAC, 101041375). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

REFERENCES

[1] Vortex Developers. 2024. Vortex. https://vortex.dev/. Accessed June 29, 2026.
[2] Siying Dong, Andrew Kryczka, Yanqin Jin, and Michael Stumm. 2021. Evolution of Development Priorities in Key-value Stores Serving Large-scale Applications: The RocksDB Experience. In *FAST*. USENIX Association, 33–49.
[3] Pascal Ginter and Viktor Leis. 2026. Active Data Lakes: Regaining Physical Data Independence Without Losing Interoperability. *Proc. VLDB Endow* (2026).
[4] Maximilian Kuschewski, David Sauerwein, Adnan Alhomssi, and Viktor Leis. 2023. BtrBlocks: Efficient Columnar Compression for Data Lakes. *Proc. ACM Manag. Data* 1, 2 (2023), 118:1–118:26.
[5] Nick Langham and Tathagata Das. 2026. Delta Kernel - Building Delta Lake connectors, made simple. https://delta.io/blog/delta-kernel/. Accessed June 29, 2026.
[6] Weston Pace. 2024. Designing a Table Format for ML Workloads. https://www.lancedb.com/blog/designing-a-table-format-for-ml-workloads. Accessed June 29, 2026.
[7] Pedro Pedreira, Orri Erling, Konstantinos Karanasos, Scott Schneider, Wes McKinney, Satyanarayana R. Valluri, Mohamed Zaït, and Jacques Nadeau. 2023. The Composable Data Management System Manifesto. *Proc. VLDB Endow.* 16, 10 (2023), 2679–2685.
[8] substrait io. 2021. Substrait: Cross-Language Serialization for Relational Algebra. https://github.com/substrait-io/substrait. Accessed June 29, 2026.

Proceedings of the VLDB Endowment. ISSN 2150-8097.