Datenbank-Spektrum (2025) 25:187–195
https://doi.org/10.1007/s13222-025-00516-6

SCHWERPUNKTBEITRAG

Opinion Pieces of the BTW 2025 Workshop On Advances in Cloud Data Management

Thomas Bodner[1] · Alexander Böhm[2] · Maximilian Böther[3] · Ana Klimovic[3] · Dominik Durner[4] · Martin Grund[5] · Andreas Kipf[6] · Ismail Oukid[7] · Berni Schiefer[7] · Panos Parchas[8] · Hinnerk Gildhoff[8] · Philipp Unterbrunner[9] · Tomas Karnagel[9] · Jana Giceva[10] · Tobias Ziegler[10] · Martin Hentschel[11]

Received: 19 July 2025 / Accepted: 8 September 2025 / Published online: 18 November 2025

Abstract

The BTW 2025 Workshop on Advances in Cloud Data Management explored recent developments and future directions in cloud data management. Speakers discussed advancements in data warehousing, query optimization, and data pipeline architectures to enhance performance and efficiency in cloud environments. The workshop also addressed challenges in managing data for modern software architectures, such as event-driven microservices, and the complexities of decomposing database systems. Additionally, presentations covered the potential of serverless computing for cost-efficient data processing and the importance of fine-grained access control for data governance.

✉ Thomas Bodner
thomas.bodner@hpi.de

Alexander Böhm
alexander.boehm@sap.com

Maximilian Böther
mboether@ethz.ch

Ana Klimovic
aklimovic@ethz.ch

Dominik Durner
dominik@cedardb.com

Martin Grund
martin@databricks.com

Andreas Kipf
andreas.kipf@utn.de

Ismail Oukid
ismail.oukid@snowflake.com

Berni Schiefer
berni.schiefer@snowflake.com

Panos Parchas
parchp@amazon.de

Hinnerk Gildhoff
hinnerk@amazon.de

Philipp Unterbrunner
ptu@observeinc.com

Tomas Karnagel
tomas@observeinc.com

Jana Giceva
jana.giceva@tum.de

Tobias Ziegler
t.ziegler@tum.de

Martin Hentschel
mhent@itu.dk

[1] Hasso Plattner Institute, University of Potsdam, Potsdam, Germany

[2] SAP, Walldorf, Germany

[3] ETH Zürich, Zürich, Switzerland

[4] CedarDB, München, Germany

[5] Databricks, Berlin, Germany

[6] TU Nürnberg, Nürnberg, Germany

[7] Snowflake Inc., Menlo Park, CA, USA

[8] Amazon Web Services, Berlin, Germany

[9] Observe, Inc., San Mateo, CA, USA

[10] TU München, München, Germany

[11] IT University of Copenhagen, Copenhagen, Denmark

For this special issue of *Datenbank-Spektrum*, we asked the speakers of the workshop to extend the abstracts of their talks by, for example, including their opinions on the future of data management systems in the cloud. Consequently, we titled this collection of articles *Opinion Pieces*. These articles are not peer-reviewed and present the authors' individual viewpoints. Nine speakers submitted articles for publication. Articles are sorted by the last name of the first author. For editorial reasons, there is only one bibliography instead of individual ones, which we would have preferred.
We hope that the idea of publishing opinion pieces will receive positive attention and that this style of submission may be replicated in other areas of the *spectrum* of databases.
We want to thank all speakers again for delivering highly interesting talks at the BTW workshop and for submitting these opinion pieces for publication.
Jana Giceva, Tobias Ziegler, Martin Hentschel, *editors*

1 Elasticity in Cloud Data Processing: a Matter of Infrastructure Economics and Architectural Implications (Thomas Bodner)

Analytical data products, such as business intelligence reports and machine learning models, require processing large amounts of data using extensive computational resources. Traditionally, provisioning resources involves high upfront expenses. The public cloud, as a shorter-term provisioning model, provides cost-effective access to pools of resources, and, as a result, is the standard for deploying data processing systems today. Recently, serverless cloud computing embodies resource pools that are highly elastic. This elasticity has the potential to make cloud-based systems easier to use and more cost-efficient, avoiding complex resource management and under-utilization.

Motivated by the potential impact that serverless cloud infrastructure has on data processing systems, recent work explores the use of this category of highly elastic cloud resources. Evaluation of the performance and cost characteristics of commercial public serverless infrastructure reveals valuable insights. Through comprehensive experiments involving various compute and storage services, in addition to end-to-end analytical workloads, the research identifies distinct boundaries for performance variability in serverless networks and storage. In addition, the research finds economic break-even points for serverless versus server-based storage and compute resources. These insights guide the efficient use of serverless infrastructure for data processing.

There are multiple data processors built entirely on serverless resources. They employ various adaptive and cost-based techniques to operate within the limits where serverless data processing remains practical. These systems show competitive performance and cost with commercial Query-as-a-Service (QaaS) systems for terabyte-scale queries of analytical benchmarks. Furthermore, they leverage the elasticity of their underlying infrastructure for cost efficiency in ad-hoc and low-volume workloads, compared to cloud data systems deployed on virtual servers. Overall, they demonstrate that serverless resources can be a viable foundation for large-scale data processing and complement server-based systems.

The advent of serverless functions draws the market for cloud compute resources closer to a commodity market, akin to that of electricity. Although prices remain significantly above marginal cost because of insufficient competition and regulation, they are organized along a continuum from long-term to short-term delivery. Users have the option to rent virtual servers through annual contracts at reduced rates or at prevailing market prices, while serverless functions are available for immediate delivery at a higher cost. Data system providers can select and combine these resources for their deployments to optimize costs.

To facilitate cost-optimal data processing via elastic resource allocation, vendors must re-architect their systems. They must develop models to predict both the runtime and cost of jobs while balancing these factors according to user constraints. In addition, they must decompose their systems into stateful and stateless components, since serverless compute is inherently stateless and relies on external services for, e.g., logging, caching, and shuffling.

2 The Challenges of Decomposing Database Systems in the Cloud (Alexander Böhm)

Modern cloud native software architectures follow a microservices approach. They decompose complex applications into sets of small, individual services. Ideally, these microservices can iterate quickly, with frequent releases to production, a small blast radius in case of failures, and high degrees of freedom regarding e.g. the choice of the programming language and development style. Moreover, the individual services can be scaled separately, leading to a better, more fine-grained resource allocation and reduced costs.

While the overall benefits of decomposition such as better scalability, elasticity, and the efficient use of resources are typically advertised publicly in corporate blogs and academic publications, decomposition also entails notable downsides that are not prominently discussed and often overlooked: *Disaggregation entails communication* between services. This communication in turn creates additional chokepoints, for example by suffering from bandwidth limitations, or often even more relevant—elevated latency. For database systems, an increased latency is particularly problematic for online transaction processing systems (OLTP), where applications often expect response times of less than a millisecond.

Many cloud-native database systems rely on sophisticated caching solutions to mitigate latency. However, this approach creates high complexity with respect to cache invalidation, distributed state handling, and increases costs as data needs to be kept redundantly.

Our conclusion is that the decomposition of stateful architectures in general and database management systems in particular requires great care to avoid overcomplicated and expensive solutions. We believe that this topic deserves more attention by both industry and academia.

3 A Database Says More than a Thousand Files: LLM Training Data Management (Maximilian Böther, Ana Klimovic)

Training large language models (LLMs) presents new challenges for managing training data due to ever-growing model and dataset sizes. State-of-the-art LLMs train over trillions of tokens that are aggregated from a cornucopia of different datasets, forming collections such as RedPajama, Dolma, or FineWeb. As data collections grow and cover more and more data with different characteristics that come from different sources, managing the data becomes time-consuming, tedious, and prone to errors. Yet, we currently do not see data management techniques being employed for managing LLM training data. As a consequence, we see the following three challenges.

Challenge 1: Inefficient Data Management On Filesystems Most LLM training data is stored and managed as files without a proper data management system, leading to storage overhead, consistency issues and performance bottlenecks.

Challenge 2: High Engineering Effort for Data Preparation The amount of data and complexity of the data mixing process can be overwhelming for model developers. They spend a lot of time writing data processing code, which is time-consuming, error-prone, and for each training run with a new mixture, requires a new copy of the dataset and a new mixing script. Utilizing a vanilla DBMS would burden ML engineers with complexities such as database administration, schema design, and performance tuning, and there still needs to be an interface between storage, query engine, and training framework.

Challenge 3: Mixtures Are Becoming Dynamic Dynamically adjusting the mixture based on training dynamics is an emerging technique, and offline preparation of the mixture, mixing based on fixed directory weights, or using a vanilla DBMS without additional infrastructure does not support this.

With more and more data being collected, we need a system to support the training data management process. We are currently building Mixtera [1], a lightweight data lake for distributed LLM training addressing these challenges. Mixtera can be deployed on top of existing training data collections. It is a centralized, read-only layer and can be declaratively queried from training clients. We presented initial ideas on Mixtera's design at the ACloudDM workshop at BTW'25 as well as at HotInfra at SOSP'24.

4 One System, No ETL, and Unlimited Data: HTAP in the Cloud (Dominik Durner)

Cloud-based systems are transforming data management by offering flexible, scalable, and cost-effective infrastructure.

With high durability and virtually unlimited capacity, cloud object storage has become a cornerstone of modern analytics.

Our work shows that cloud network bandwidth closes the NVMe throughput gap [2], making direct analytical processing on cloud object storage not only feasible but also increasingly efficient. We developed *AnyBlob*, a multi-cloud download manager that delivers throughput comparable to SSD-RAIDs with low CPU overhead; and integrated *AnyBlob* into our database system for high-performance analytics [3, 4]. Building on this, *Colibri* offers a hybrid storage engine designed for HTAP workloads [5]. It organizes hot transactional data in row-based form and cold analytical data in a columnar layout, enabling optimized access patterns for both OLTP and OLAP workloads. This combined architecture reduces data duplication and analytical time-to-insight by integrating analytics and transactions within a single engine.

The increasing skepticism toward large-scale distributed architectures, reflected in claims that "big data is dead" [6], is supported by workload demands and the cost-benefit trade-offs of distributed systems. Studies show that most applications do not require large-scale distributed memory, as very few applications scan and process 100's of terabytes of data [7]. Continued improvements in network bandwidth, from 100 Gbit/s in 2018 [8] to 200 Gbit/s in 2024 [9], further alleviate the demand for distributed systems solely for data transfer. Instead, database users benefit from simpler to deploy, single-instance architectures that are backed by scalable storage and fast networking.

Rising customer demand for data ownership over sensitive data and regulatory requirements has led to a growing popularity of self-managed data infrastructure in private, public, or even hybrid clouds.

In such environments, a simple data infrastructure, provided by single-instance HTAP systems, is crucial to enable organizations to benefit from low-latency insights without complex data pipelines.

In summary, there is a clear trend towards more streamlined cloud data architectures that seamlessly integrate analytics and transactions while addressing data scalability and data ownership needs.

5 Databricks Lakeguard: Supporting Fine-grained Access Control and Multi-user Capabilities for Apache Spark Workloads (Martin Grund*, Stefania Leone, Sven Wagner-Boysen, Sebastian Hillig, Tim Januschowski)

Lakehouse architecture has become the de facto standard for managing data from many different sources in open formats using fast and cheap cloud storage. Enterprises now want to apply fine-grained access control policies to manage increasingly complex data governance requirements. These rich policies should be uniformly applied across all their workloads.

In addition, the definition of governance in this context has evolved. Whereas previously, Lakehouse governance was focused on coarse-grained data governance, this has changed with the introduction of Databricks Unity Catalog [10]. Databricks Unity Catalog is the central catalog for all kinds of assets across data and AI and is, therefore, the foundation of our Data Intelligence Platform.

Customers expect to govern access to all assets in a dynamic, fine-grained way and enforce it at runtime. Traditionally, workloads running on Apache Spark cannot enforce fine-grained governance policies such as views, row filters or column masks due to the lack of proper security boundaries between user code and the query engine.

At Databricks, we addressed this challenge with Databricks Lakeguard [11], our implementation of a unified governance system that enforces any fine-grained data access policies, including row-level filters and column masks across all of an enterprise's data and AI workloads. Lakeguard builds upon two main components: First, it uses Spark Connect, a JDBC-like execution protocol, to separate the client application from the server and ensure version compatibility. Second, it leverages container isolation in Databricks' cluster manager to securely isolate user code from the core Spark engine. With Lakeguard, a user's permissions are enforced at all times: for any workload and in any of the supported language (SQL, Python, Scala, and R) on multi-user compute.

The Lakeguard architecture provides the foundation for innovating how we provision, manage, and upgrade compute infrastructure for our customers. Today, we already leverage the same compute platform for both our classic and serverless offerings to reduce costs and simplify the product experience. We also see a lot of potential to evolve this architecture in the future. For example, the separation of user code from the query engine lets us rethink approaches to execution locality. By offloading GPU workloads, we can run multi-user workloads securely, leveraging one or more GPUs. This provides more flexibility for us to manage resource requirements, such as memory and CPU, more efficiently for our customers. The same flexibility is very useful when building agentic systems that interact with a lot of data. Such tools leverage very often generated or only partially trusted code, and our sandbox management provides the most suitable environment.

To conclude, customers can continue to rely on Apache Spark's extensibility and programmability for their workloads ranging from ETL to low-latency SQL and AI needs. The combination of Databricks Unity Catalog and Lakeguard allows for the continued evolution of the lakehouse architecture, satisfying the enterprise governance requirements of fine-grained access control and central governance.

6 Workload-Driven Indexing in the Cloud (Andreas Kipf)

In this talk, we challenged the long-standing assumption that indexing is too costly to be practical in large-scale data warehouses. By analyzing real-world workloads from Amazon Redshift, we found that these workloads are highly repetitive—both in terms of queries and scans. While systems like Redshift use optimizations such as result caching and materialized views to improve query performance on repetitive workloads, their effectiveness is often undermined by data modifications—including inserts, deletes, and updates—that occur between query repetitions.

We introduced predicate caching [12], a lightweight, workload-aware technique that caches scan predicates and their qualifying row ranges, rather than entire queries. Specifically, a cache entry consists of a full scan expression (key) and a compressed representation of its qualifying row ranges (value). A predicate cache is built on the fly during query execution and supports efficient incremental updates. Despite its simplicity and ease of integration into existing systems, predicate caching can significantly boost query performance by reducing both the number of rows scanned and overall I/O. In addition, it improves join performance by caching the effects of semi-join (Bloom) filters.

Looking ahead, we believe that workload-aware optimization techniques—such as predicate caching—will be central to the future of cloud data warehousing, offering substantial performance gains and cost savings. As Large Language Models (LLMs) become increasingly integrated into data analytics, the need for efficient and cost-effective query execution is more critical than ever. Businesses are eager to harness the power of state-of-the-art models while keeping costs under control. Cloud data warehouse systems are beginning to integrate LLM capabilities—e.g., by supporting semantic predicates as user-defined functions (UDFs)—opening new opportunities for workload-aware optimizations as query patterns evolve.

7 The Fine Art of Work Skipping (Ismail Oukid, Berni Schiefer)

7.1 Work Skipping: Data Pruning

Pruning is a work-skipping technique that significantly improves query performance by skipping data blocks that are not relevant to the query. This is achieved by leveraging metadata to determine which blocks can be safely ignored. In distributed cloud architectures, where data transfer between storage and compute layers can incur significant costs and latency, the ability to skip irrelevant data blocks is particularly valuable.

Research by Van Renen and Leis [13] highlights the significance of this issue, showing that scanning and filtering operations account for approximately 50% of the total query processing time in analytical workloads. Our own measurements corroborate this conclusion, though with a slightly lower figure: for SELECT queries specifically, scanning and filtering operations consume 37% of query processing time[1].

Further analysis of our analytical workloads reveals that more than half of analytical queries have a selectivity of 5% or less. This means that the majority of queries ultimately need only a small fraction of the data they scan, indicating significant pruning opportunities.

To realize this pruning potential, Snowflake implements several types of compile-time and runtime pruning techniques such as filter pruning, join pruning, limit pruning, and Top-K pruning (see Zimmerer et al. [14] for a detailed study of Snowflake's pruning techniques). These techniques are not mutually exclusive, but complementary. Their combined power allows Snowflake to achieve a staggering 99.4% pruning efficiency across all production workloads [14]. As an example, Top-K pruning led to a 53.5% average improvement for long-running Top-K queries.

[1] Approximate figure measured for a single day (2025-02-03) across all Snowflake regions.

Despite the impressive pruning ratio Snowflake achieves today, our telemetry shows even more pruning potential: Approximately 40% of all accessed data blocks today could be pruned. More innovation both in terms of advanced pruning techniques and more powerful metadata is required to realize this pruning potential.

Pruning efficacy relies on metadata quality: presence, correctness, and granularity. Metadata availability is the first requirement, as not all systems provide detailed per-column, per-data-block metadata. Correctness, which can be approximate but with defined bounds for safe pruning, is essential. Granularity, such as file or sub-file levels, also impacts pruning effectiveness.

7.2 Work Skipping: Result Caching and Reuse

Result reuse is a work skipping technique that leverages caching to avoid recomputing the same query result. When a query is executed, Snowflake caches the results for 24 h. If the same query is executed again within this timeframe and the underlying data has not changed, the results are retrieved from the cache, significantly reducing query latency.

The impact of result reuse cannot be overstated: Approximately 25% of all `SELECT` queries in Snowflake hit the result cache, demonstrating both the frequency of repeated queries in analytical workloads and the effectiveness of Snowflake's caching mechanisms.

Snowflake's cloud-native architecture enables result caching and reuse at a scale unattainable in traditional on-premises systems: Snowflake's result cache operates across users and is not constrained by size limitations.

For effective result reuse, Snowflake relies on two key components:

- **Online result reuse candidate discovery**: This component efficiently indexes (using FoundationDB [15]) previously executed queries that can be reused for a current query. The system must quickly determine if a semantically equivalent query has been run before, without introducing significant overhead to the query processing pipeline.
- **Concise query summary**: Snowflake creates an accurate representation of each query to determine if it is semantically equivalent to a previously executed one. This summary encompasses a query hash for ease of indexing and additional, relevant semantic properties.

Not all query results are suitable for reuse. Queries containing entropy-generating functions such as `RANDOM()` and `ENCRYPT()` cannot be reused as they are designed to produce different results each time they're executed.

However, Snowflake intelligently handles a class of functions that, despite being technically non-deterministic, should be allowed for reuse in practice. Examples include `ANY_VALUE()` and `MIN_BY()`.

Data versioning is another crucial aspect of result reuse. If the underlying data or any object definition changes, previously cached results cannot be reused. However, Snowflake distinguishes between substantive changes that affect query results and those that do not. For instance, reorganizing data for better pruning through Automatic Clustering is semantically irrelevant and does not invalidate cached results.

While a 25% result reuse ratio is significant, our telemetry still indicates improvement potential, emphasizing the importance of the quality of the query summary (which can be seen as query metadata) to achieve the full potential of result reuse.

7.3 Conclusion

We have demonstrated that work skipping is paramount for high performance in cloud-based systems and that high quality metadata is its backbone. We also showed that there are further substantial innovation opportunities in both data pruning and result reuse.

Snowflake unifies its core components under format-agnostic APIs, ensuring that most features and performance enhancements, including work-skipping techniques, are compatible with [31]. In fact, we argue that Apache Iceberg is leveling the field, allowing academic researchers and industry practitioners to tackle performance challenges under the same architectural assumptions.

8 Auto-Tuning and Intelligent Scaling in Amazon Redshift (Panos Parchas, Hinnerk Gildhoff)

Amazon Redshift [16] is AWS's cloud data warehouse used by tens of thousands of customers to process exabytes of data daily using existing business intelligence tools.

Each Redshift cluster uses an MPP execution engine spread across multiple EC2 nodes. Data is persisted on S3 and is cached on the local disk of EC2 nodes for faster access. Traditionally, data warehouses comprise of several knobs that need to be fine-tuned in order for the system to perform optimally. These knobs control everything from coarse grained architectural decisions to low level details like table physical tuning. For instance, the size and hardware type of the cluster (i.e., the number and type of EC2 instances) depends on the expected workload, which may vary substantially over time. On the other end of the spectrum, the physical tuning of the data (e.g., distribution and sort keys) may heavily affect the workload performance through join collocation and block level filtering. As the workload evolves and varies, all these decisions may need to be revisited [7].

Throughout its evolution, Redshift has provided mechanisms that alleviate the pain of all this constant monitoring and fine-tuning of the system in various levels. At the first level, Redshift implemented an Advisor engine to serve as a smart DB administrator and take care of physical tuning decisions of the database tables [17]. At a second level, Redshift introduced learned components that heavily integrate with the system and go beyond what even a sophisticated administrator can do for workload management [18], Materialized Views [19, 20] and sorting [12, 21]. The culmination of Redshift autonomics is Redshift Serverless, which allows customers to run and scale analytics without the need to set up and manage data warehouse infrastructure. This third level allows for both horizontal and vertical scaling of the cluster without any customer interference [22].

In what follows we dive deeper into these layers of automation. We provide high-level insight of the problems we faced and the solutions we implemented. We conclude this opinion article with a forward looking statement on the future of autonomics in cloud databases.

8.1 Autonomous DB Administrator

The Redshift Advisor acts as an external database administrator which is automatically analyzing customer workloads to optimize table physical properties such as distribution keys, sort keys and column encoding.

[17] describes the distribution key recommendation in detail; the workload history is represented via a join multigraph and a graph theoretic algorithm based on combinatorial optimization identifies the most beneficial join columns for distribution keys.

Sort keys can improve query performance due to block level filtering (i.e., pruning of entire blocks of data based on min/max metadata aka zone-maps [21]). Sort key recommendation is based on monitoring of the pruning power of the various predicates that are used in the workload. Optimal compression algorithms can be derived by used data types and simple heuristics. These fine-tuning recommendations are either surfaced to the customer's console, or automatically applied (on idle cluster times) via Automatic Table Optimization [23].

Even before the workload arrives, Redshift Advisor employs ML techniques to choose appropriate distribution and sort keys, based on the DB schema, constraints (e.g., primary/foreign key relations) and data statistics (e.g., skew, cardinalities etc) [24]. Once the cluster starts serving queries, those choices may be refined using the workload based techniques discussed above.

8.2 Learned Components: Beyond a DB Admin

An Autonomous DB administrator is not enough to serve the customer needs in a modern cloud-based DBMS. Redshift has moved beyond, providing learned components that are deeply integrated with the system to optimize different stages of query execution.

Automatic workload management (Auto-WLM) is an internal component that uses machine learning to assign cluster resources (e.g., memory) on concurrently executing queries [18] based on their predicted needs. In addition, this component uses globally and locally trained predictors to prioritize short-running queries over long running ones.

Materialized views (MVs) are ubiquitous in databases as a powerful tool that enhances query performance and data retrieval efficiency. Redshift automates the efficient maintenance and use of MVs in four ways [19]. First, by introducing AutoMVs, the system monitors the query workload and automatically creates useful MVs in the background. Redshift constantly evaluates the AutoMVs and decides on their usefulness in the query workload. If the workload changes and the MVs are no longer deemed to be useful, Redshift automatically drops them to clean-up resources. All this is completely transparent to the database users.

Second, Redshift incrementally maintains complex materialized views to reflect changes on base tables. Incremental view maintenance can handle MVs that contain filter, projection, grouping, join and set operators. Since Redshift thrives on batch processing, MVs are maintained asynchronously, so that the transactional workload is not slowed down.

Third, Redshift automates the timing of the maintenance. In particular, Redshift detects which MVs are out of date and maintains a priority queue to choose which MVs to update in the background. The prioritization of refreshes is based on a cost/benefit analysis of the available MVs and depends on (1) the utility of a materialized view in the query workload and (2) the cost of refreshing the materialized view in terms of processing cycles. The goal is to maximize the overall performance benefit of materialized views.

Fourth, Redshift users can directly query an MV but they can also rely on Redshift's sophisticated autorewriting feature that rewrites queries over base tables to use the best eligible materialized views. Autorewriting is cost based and proceeds only if the rewritten query is estimated to be faster than the original query. The rewrite is robust so it can identify applicable MVs for parts of the query (subqueries, UNION ALL legs etc.) and MVs that contain a superset of the queried data (applying any relevant filters). In order to support automatic MV creation, incremental view maintenance and autorewriting, we developed a novel DSL-based query rewriting framework, which enables the Redshift team to keep expanding the SQL scope of these features.

Lastly, we noticed that in some cases, sorting by *any* table column is not enough (e.g., string predicates with wildcards at the beginning or predicates of the form $col\,A > col\,B$ do not benefit by classic sort keys). In such cases, it is more beneficial to sort the table columns based on whether they qualify or not for a given set of common predicates [21] or cache the rowids that qualify using predicate caching [12].

8.3 Instance Optimized System

Other than individual component decisions that were presented in the previous sections, there are system wide choices that determine the system performance. One of these is the hardware that is being used for the cluster and how it should scale up and down based on the workload demands. In Redshift we completely removed these decisions from the customers; with Redshift Serverless, the cluster is up only for the time window that it is used to serve queries or execute fundamental auto-maintenance operations. This has a tremendous effect in Redshift's price performance, since customers do not need to pay for idle clusters [16].

Furthermore, instead of choosing the individual hardware, Redshift employs ML techniques to find the most appropriate cluster size and resize the cluster vertically (assuming a constant flow of expensive queries) or horizontally (through concurrency scaling [16]) in cases of workload spikes. To achieve this, the system continuously monitors the workload and runs multiple what-if analyses on locally and globally trained models to estimate the cost-benefit of scaling up or down [22].

8.4 Outlook

In this article, we briefly presented the evolution of autonomics in Amazon Redshift. We started with automating simple database administrator tasks up to complex internal optimizations at query runtime. Machine learning models are used at various database components to replace classic decision making. Hierarchical structures are helping us to balance fast and optimal decisions from local and global trained models. In the future, we expect to leverage more globally trained machine learning models for different services at AWS executed by a network of background processes with the goal to increase performance and lower costs for our customers. Customers can concentrate on their business application and use simple endpoints without the need to understand a complex cloud architecture with multiple dedicated services under the hood.

9 OBSERVE—Petascale Streaming and the Future of Observability (Philipp Unterbrunner, Tomas Karnagel)

In recent years, the landscape of monitoring and observability has undergone a substantial evolution. Data volumes have exploded, use cases have diversified, and the demand for real-time insights has increased significantly. Observe is a Software-as-a-Service (SaaS) product for monitoring and observability designed to meet these new challenges, built in large part on Snowflake [25, 26], at its core a cloud-based relational data warehouse.

When we founded Observe in 2017, the monitoring and observability market was a hodge-podge of log search, metrics (time series), and distributed tracing products—each with their own custom data models, query engines, and query languages.

Based on the past 50 years of data management, which can be described as a series of waves of post-relational data models and workloads (object-relational, MOLAP, XML, graph databases, key-value stores and others) spinning out and eventually being re-absorbed into the relational ecosystem, we made the contrarian bet that the future of observability would also be relational.

Observe today is hosting 25 PiB of data (compressed) and 200 million queries per day for our customers, validating our bet. And we finally see other systems, such as a HyperDX on ClickHouse [27], adopting a similar architecture as Observe. We believe this trend towards observability products built with cloud-hosted relational technology is going to accelerate, based on several technological undercurrents.

First, the unbundling and rebundling of data services will continue. While specialized data stores for specific workloads will persist, the need for a central, reliable, and scalable relational core for analytical and operational insights will only intensify. Platforms such as Snowflake, Firebolt [28], ClickHouse [29], or MotherDuck [30], demonstrating the power of a fully managed, cloud-native relational data warehouse, are paving the way for a future where the complexities of infrastructure management are entirely abstracted away, allowing users to focus solely on extracting value from their data.

Second, the convergence of data warehousing and data lake capabilities will blur traditional boundaries. The ability to seamlessly ingest, process, and analyze both structured and semi-structured data within a uniformly governed environment, while retaining the ACID properties and tools ecosystem of relational systems, is becoming paramount. The impact and rapid spread of the Iceberg table format serves as evidence of that trend [31]. We anticipate further innovations in schema-on-read capabilities, efficient handling of semi-structured data, and open standards such as Iceberg within relational platforms, making them the principal infrastructure for all organizational data.

Third, real-time data processing and analytics will become deeply integrated into relational systems. The demand for immediate insights, not just historical analysis, is growing as all businesses become increasingly digitized and online. We expect to see advancements in stream processing capabilities directly within or tightly coupled with relational data warehouses, enabling real-time observability, alerting, and decision-making based on continuously flowing data. Our own experience at Observe, processing massive streams of observability data in near real-time on Snowflake, underscores the viability of, and demand for, such systems.

Finally, AI and machine learning are becoming intertwined with relational data management. Beyond simply storing data for AI/ML workloads, we see relational systems include features such as automated schema optimization, intelligent data tiering based on access patterns, or built-in machine learning capabilities for anomaly detection and predictive analytics [32]. The rich context and structured nature of relational data make it an ideal training ground and operational backbone for AI/ML initiatives.

In conclusion, we believe the future of monitoring and observability is not about abandoning the relational model. On the contrary, it lies in the continued evolution and prospering of relational systems in the cloud.

Funding Open Access funding enabled and organized by Projekt DEAL.

References

1. Böther M, Yao X, Kerimoglu T, Graur D, Gsteiger V, Klimovic A (2025) Mixtera: A Data Plane for Foundation Model Training. https://arxiv.org/abs/2502.19790
2. Durner D, Leis V, Neumann T (2023) Exploiting cloud object storage for high-performance analytics. PVLDB 16(11)
3. Cedar D (2024) An ode to PostgreSQL, and why it is still time to start over. https://cedardb.com/blog/ode_to_postgres

4. Neumann T, Freitag MJ (2020) Umbra: A disk-based system with in-memory performance. CIDR
5. Schmidt T, Durner D, Leis V, Neumann T (2024) Two birds with one stone: Designing a hybrid cloud storage engine for htap. PVLDB 17(11)
6. Tigani J (2023) Big Data is Dead. https://motherduck.com/blog/big-data-is-dead
7. Renen A, Horn D, Pfeil P, Vaidya K, Dong W, Narayanaswamy M, Liu Z, Saxena G, Kipf A, Kraska T (2024) Why TPC is not enough: An analysis of the Amazon Redshift fleet. PVLDB 17(11)
8. Amazon (2018) New C5n Instances with 100 Gbps Networking. https://aws.amazon.com/blogs/aws/new-c5n-instances-with-100-gbps-networking
9. Amazon (2024) Amazon EC2 C7gn metal instance is now. https://aws.amazon.com/about-aws/whats-new/2024/03/amazon-ec2-c7gn-metal-instance-available
10. Chandra R, Chen H, Matharu R, Cai S, Chen J, Dutta P, Ghita B, Greenstein T, Holla G, Huang P, Huo Y, Ionescu A, Ispas A, Januschowski T, Karajgaonkar V, Leone S, Lewis D, Li A, Li N, Lian C, Link S, Lu Q, Ma Y, Pettitt C, Prabhakaran V, Raducanu B, Rong K, Roome P, Shetty S, Smith S, Sun X, Tang Y, Wen W, Xia L, Zeng J, Zhang B, Xin R, Zaharia M (2025) Unity Catalog: Open and universal governance for the lakehouse and beyond. SIGMOD
11. Grund M, Leone S, Hövell H, Wagner-Boysen S, Hillig S, Kwon H, Lewis D, Mund J, Poli P-F, Montrieux L, Crelier O, Li X, Xin R, Zaharia M, Petropoulos M, Papathanasiou T (2025) Databricks Lakeguard: Supporting fine-grained access control and multi-user capabilities for apache spark workloads. SIGMOD
12. Schmidt T, Kipf A, Horn D, Saxena G, Kraska T (2024) Predicate caching: Query-driven secondary indexing for cloud data warehouses. SIGMOD
13. Van Renen A, Leis V (2023) Cloud analytics benchmark. PVLDB 16(6)
14. Zimmerer A, Dam D, Kossmann J, Waack J, Oukid I, Kipf A (2025) Pruning in Snowflake: Working smarter, not harder. SIGMOD
15. FoundationDB: FoundationDB. https://www.foundationdb.org
16. Armenatzoglou N, Basu S, Bhanoori N, Cai M, Chainani N, Chinta K, Govindaraju V, Green T, Gupta M, Hillig S, Hotinger E, Leshinksy Y, Liang J, McCreedy M, Nagel F, Pandis I, Parchas P, Pathak R, Polychroniou O, Rahman F, Saxena G, Soundararajan G, Subramanian S, Terry D (2022) Amazon Redshift re-invented. SIGMOD
17. Parchas P, Naamad Y, Bouwel PV, Faloutsos C, Petropoulos M (2020) Fast and effective distribution—key recommendation for Amazon Redshift. PVLDB 13(11)
18. Saxena G, Rahman MA, Chainani N, Lin C, Caragea G, Chowdhury F, Marcus R, Kraska T, Pandis I, Narayanaswamy BM (2023) Auto-WLM: Machine learning enhanced workload management in Amazon Redshift. SIGMOD
19. (2022) Optimize your Amazon Redshift query performance with automated materialized views. https://aws.amazon.com/blogs/big-data/optimize-your-amazon-redshift-query-performance-with-automated-materialized-views
20. Svingos C, Hernich A, Gildhoff H, Papakonstantinou Y, Ioannidis Y (2023) Foreign keys open the door for faster incremental view maintenance. SIGMOD
21. Ding J, Abrams M, Bandyopadhyay S, Palma LD, Ji Y, Pagano D, Paliwal G, Parchas P, Pfeil P, Polychroniou O, Saxena G, Shah A, Voloder A, Xiao S, Zhang D, Kraska T (2024) Automated multidimensional data layouts in Amazon Redshift. SIGMOD
22. Nathan V, Singh V, Liu Z, Rahman M, Kipf A, Horn D, Pagano D, Saxena G, Narayanaswamy BM, Kraska T (2024) Intelligent scaling in Amazon Redshift. SIGMOD
23. (2023) Automate your Amazon Redshift performance tuning with automatic table optimization. https://aws.amazon.com/blogs/big-data/automate-your-amazon-redshift-performance-tuning-with-automatic-table-optimization
24. (2023) Amazon Redshift announces enhancements to Advisor sort and distribution key recommendations. https://aws.amazon.com/about-aws/whats-new/2023/12/amazon-redshift-advisor-sort-distribution-key-recommendations
25. Dageville B, Cruanes T, Zukowski M, Antonov V, Avanes A, Bock J, Claybaugh J, Engovatov D, Hentschel M, Huang J et al (2016) The Snowflake elastic data warehouse. SIGMOD
26. (2023) How Observe Uses Snowflake to Deliver the Observability Cloud. https://www.observeinc.com/blog/how-observe-uses-snowflake-to-deliver-the-observability-cloud-part-1
27. (2024) HyperDX—Why We Chose Clickhouse Over Elasticsearch for Storing Observability Data. https://www.hyperdx.io/blog/why-clickhouse-over-elasticsearch-observability
28. Firebolt Cloud Data Warehouse Whitepaper T https://www.firebolt.io/resources/firebolt-cloud-data-warehouse-whitepaper
29. Clickhouse Cloud. https://clickhouse.com/cloud
30. Open Lakehouse Stack T (2025) DuckDB and the Rise of Table Formats. https://motherduck.com/blog/open-lakehouse-stack-duckdb-table-formats
31. Apache Iceberg: Apache Iceberg. https://iceberg.apache.org
32. Zhou X, Chai C, Li G, Sun J (2023) Database meets artificial intelligence: A survey (extended abstract). ICDE