Journal of Information Systems Engineering and Management
2026, 11(1s) e-ISSN: 2468-4376 https://www.jisem-journal.com/ **Research Article**

Data-First Application Architecture: Unifying Transactional and Analytical Workloads on Modern Data Platforms

Ashrith Reddy Mekala
Cloudwick Inc, USA

ARTICLE INFO

Received: 20 Jan 2026

Revised: 23 Jan 2026

ABSTRACT

Customarily, enterprise data architectures have been understood to be built on the separation of transactional systems and analytical systems, leading to separate repository data stores, extract-transform-load (ETL) processes, and latency to business-critical decision-making. Modern cloud-native data platforms are beginning to challenge this model with a new architecture that breaks down the boundary between operational systems and analytical systems. In a data-first application architecture, the data platform is the system of record, and applications pull from it. A hub-and-spoke model is often used with medallion data layers to maintain multiple representations of business-ready data as it passes through various levels of refinement from raw ingested data to business-ready data structures. Hybrid transactionalanalytical processing allows a mixture of OLTP and OLAP workloads with dual-format data representations, query optimizers, and related techniques. Streaming ingestion and change data capture enable streaming data with subsecond data freshness and ACID transaction semantics using isolation levels such as serializable snapshot isolation. Event-driven architectures complement this pattern by propagating data in the form of immutably logged events to distributed consumers. They can respond to business event flows through choreography without tight coupling. Practical implementation aspects of workload-optimized querying, multi-tier caching, elastic resource management, and infrastructure consolidation yield economic benefits and extremely low-latency performance for business events consumed by users. Those organizations using unified transactional-analytical architectures report dramatic improvements in time-to-understand, reduced infrastructure costs, accelerated development lifecycles, and improved support for machine and artificial intelligence-based systems that require a fresh and consistent view of data on which prescriptive models can act. Operational-analytical workload convergence is an architectural step change to make organizations more agile in data-rich environments where speed of business understanding and decisions are critical to competitive differentiation.

Keywords: Hybrid Transactional-Analytical Processing, Data-First Architecture, Event-Driven Systems, Streaming Data Ingestion, Unified Data Platforms

1. Introduction: Breaking Down the OLTP-OLAP Divide

Enterprise architecture has historically split transactional systems for high-volume, consistent write transactions (online transaction processing—OLTP) and analytical systems for complex queries across historical data (online analytical processing—OLAP). Originally, hardware and performance differences between these types of systems led to this distinction, which was considered a compromise at the time. However, in practice, this distinction has resulted in duplicate data stores, brittle ETL pipelines, and challenges with data synchronization. To address these issues, self-driving database systems have emerged that leverage autonomous optimization and adaptive tuning based on learned workload patterns [1]. These smart database systems automatically employ machine learning techniques to adjust database configuration settings, predict resource requirements, and optimize query execution plans, providing an advanced solution for managing mixed workload databases without human intervention.

At a high level, the architecture is expensive and simple: operational databases capture the transactions, ETL pipelines batch-extract and batch-transform this transactional data, and analytical warehouses are independent systems of perception. This leads to a batch architecture with many problems: high latency and inconsistent operational and analytical data, high infrastructure costs, and development teams maintaining a number of disparate data models. To overcome these limitations, new techniques have been proposed to provide consistency and isolation with strong guarantees while still supporting concurrent transaction processing [2]. Serializable snapshot isolation techniques provide the highest level of isolation without the overhead of strictly serializable systems. This approach provides a more consistent interface for transactional and analytical workloads to coexist, with the strongest possible isolation guarantees.

However, the emergence of cloud-native data platforms with distributed computing, columnar storage, query optimization, hybrid processing engines, and other features has made it feasible to capture both transactional and analytical workloads on the same data platform. This has resulted in the emergence of a new data-first application architecture, where the data platform becomes the system of record for all applications, and where the boundaries between the system of record and the system of insight have been eliminated.

2. The Hub-and-Spoke Model: The Data Platform as Central Nervous System

In this approach, a hub and spoke model is used: the data platform or warehouse is the hub, with applications, services, and analytics consumers as spokes. This is the reverse of the classic application architecture pattern, where each application has its own dedicated database, and a network of databases at the edge of the network feeds data to a data warehouse in periodic batches. This architectural jump to cloud-native data platforms means shared data management for multiple workload types without any degradation in performance or consistency. In a much less conventional approach to solving mixed workloads, self-driving database systems that identify resource parameters to tune, based on predictions of resource saturation in the near future, have been proposed [1]. These systems are based on Machine Learning models trained with the patterns of queries and resource usage, taking into account the mechanisms of indexes, materialized views, and query routing to deliver optimal performance to operational and analytical consumers without the need for database administrator intervention.

As a hub-and-spoke data platform, the Lakehouse is the system of record for authoritative business event history, the integration backbone receiving data from multiple systems through streaming ingestion, API write, and many others; the serving layer performing best optimization for operational and analytical workloads; and the consistency layer enforcing data quality and schema evolution. The Lakehouse implementation is typically represented through a medallion architecture, with Bronze, Silver, and Gold layers representing the progression from raw ingestion, through refinement, to business-ready datasets with different access patterns and latency profiles. The Bronze layer captures raw data and its complete lineage. The Silver layer performs the data standardization and deduplication processes to produce clean datasets. The Gold layer contains aggregated datasets that have been improved with business logic for specific consumption patterns. Depending on their latency and quality requirements, applications access the appropriate layer. User-facing applications, which have sub-second SLAs, access materialized views from the Gold layer.

The advantages of centralized architecture are the elimination of duplicated data across application silos, the reduction of the total cost of ownership (TCO) due to shared infrastructure, the creation of a single source of truth to read from for a data consumer, the simplification of data governance via a common access control, and the acceleration of data engineering development on reused curated datasets. These include the requirement of central platform availability, handling distributed contention over shared compute resources, minimizing latency across a network between applications, hub integration, and developing wide-ranging observability to understand where events are coming from in many consumers. Therefore, to reduce these trade-offs, organizations are generally recommended to implement the appropriate failover and caching strategies when leveraging centralized data in a managed way. As an example of this, self-driving databases offer autonomous tuning of their parameters, which can be tedious when manually tuning for different access patterns and workload characteristics [1].

Layer	Primary Function	Data State	Typical Use Cases	Performance Characteristics
Bronze	Raw data landing zone	Unprocessed, asreceived from sources	Audit trails, data lineage, and regulatory compliance	High write throughput, append-only operations
Silver	Cleansed and standardized data	Deduplicated, validated, conformed	Cross-application integration, shared business entities	Balanced read-write performance, normalized structures
Gold	Business-ready aggregated data	Pre-computed, enriched, optimized	Operational dashboards, user-facing applications, ML features	Ultra-low latency reads, materialized views
Consumption	Application-specific views	Highly specialized, domain-optimized	Real-time personalization, fraud detection, recommendations	Sub-second query response, edge caching

Table 1: Hub-and-Spoke Architecture Layers and Characteristics [3, 4]

3. Achieving Sub-Second Data Freshness with Transactional Integrity

Transactional-analytical systems maintain current data and support the ACID properties of operational systems. Current systems focus on using excellent technology to provide these capabilities. As such, their deployment has demonstrated that the proper design of low-latency data ingestion and data query performance is not mutually exclusive in the same infrastructure. Advanced research into memory-optimized storage architectures has shown the ability to create dual representations of data in support of transactional and analytical workloads without sacrificing performance for either use case [3]. Such architectures can support row-based and columnar representations of the same data and may be capable of automatically synchronizing the two representations with only a small amount of overhead on transactional paths.

In contrast to batch processing ETL architectures, data-first architectures leverage Change Data Capture (CDC) strategies that continuously read and process source system transaction logs and consume every row-level change (insert/update/delete) with minimal performance overhead on source systems. Change events are streamed into the data platform in quasi-real-time via message queues and stream processing frameworks that provide event buffering and ordering guarantees that support correct data.

The streaming changes are then delivered to the platform's storage layer, where they are typically processed in micro-batches, which are batches of changes that are committed frequently to provide a near-continuous update while maintaining transactional semantics. Hybrid Transactional and Analytical Processing systems apply complex dual storage representation patterns in a single system where row-oriented storage structures optimize transactional workloads with lowlatency indexed lookups, and column-oriented storage structures are optimized for analytics-style scans that perform aggregations across millions of rows [4]. In these architectures, smart optimizers explore a query's individual statements, classify them as either latency-sensitive or throughputoriented, and examine the selectivity and access pattern before directing the query towards the storage representations best suited for the task at hand.

Selecting the appropriate transaction isolation level that balances concurrency and correctness guarantees is crucial for maintaining strong consistency under mixed workloads. Modern implementations use snapshot-based approaches to guarantee serializability while avoiding the performance overhead of lock-based protocols [5]. Serializable snapshot isolation techniques provide the same level of isolation as strict serializability, and they avoid write-write conflicts and serialization anomalies. The availability of dependency tracking allows read operations to avoid blocking other concurrent transactions and provide serializability at higher throughput, reducing the overhead for mixed transactional analytical workloads. Likewise, read-committed isolation allows analytical reporting queries to read concurrently from the database without blocking concurrent writes, while snapshot isolation provides point-in-time consistent reads for long-running analytical queries. Multiversion concurrency control improves concurrency by maintaining multiple copies of each record simultaneously, allowing operations not to block each other and periodic garbage collection to free old versions. When coupled with read-your-writes consistency, strict freshness guarantees that each application is immediately notified of the completion of its transaction. For analytical workloads with a push toward slightly stale results, eventual consistency with bounded staleness may increase performance by restricting queries from seeing data more than a certain amount of time behind the present.

Isolation Level	**Consistency Guarantee**	**Concurrency Characteristics**	**Typical Applications**	**Performance Trade-offs**
Read Uncommitted	Minimal, allows dirty reads	Maximum concurrency, no blocking	Background analytics, approximate aggregations	Highest throughput, lowest consistency
Read Committed	Prevents dirty reads	Good concurrency, short read locks	Standard analytical queries, reporting	Balanced performance and consistency
Snapshot Isolation	Point-in-time consistency	Excellent concurrency, MVCC-based	Long-running analytics, data exports	Minimal contention, version overhead
Serializable Snapshot	Full serializability guarantees	Good concurrency with conflict detection	Financial transactions, inventory management	Slight overhead for anomaly detection

Table 2: Consistency Models and Isolation Mechanisms [5, 6]

4. Event-Driven Architectures for State Synchronization

The event-driven pattern extends the hub-and-spoke pattern by enacting state changes in interested consumers that require real-time notifications from the data platform. Event-driven patterns alter the way systems interact and maintain consistency among distributed components. Decoupling is achieved from a request-response model to an event-driven model, making strong consistency guarantees with a carefully structured event stream and the idempotency property of processing. Messaging streaming platforms are widely adopted as the backbone of highly scalable event-driven architectures with durability and ordering guarantees, capable of more than a million events per second [6]. Such systems have a fault-tolerant, distributed commit log, multi-tiered message queue, storage engine, and stream-processing engine, allowing applications to post events that can be consumed by multiple independent consumers in a parallel fashion without synchronization and coordination overhead or performance penalties.

Event sourcing models change in the system as an immutable sequence of events that describe the business transactions, unlike mutable records that are rewritten to the database with changes to their properties. Event sourcing can ease audit trails, temporal querying, and debugging because it preserves a detailed sequence of all changes to the system state [7]. Applications and other analytics users can subscribe to streams of events that interest them. This saves clients money on polling and speeds up response times and resource use. Events are automatically written to the data platform's event log, which acts as a system of record for events and a notification mechanism. Immutable event logs allow full audit trails and enable temporal queries to recreate what the state looked like at any moment. To maintain low-latency aggregation queries, these aggregates are incrementally maintained in materialized views, with systems simply determining which aggregates have been invalidated by the change and only updating the changed partitions rather than all partitions. As a result, incremental maintenance is done continuously, and these views are always up to date within seconds of the source.

Materialized views can be used for low-latency queries of commonly used aggregated data, allowing end-users to receive up-to-date data at the required speed. Together with event-driven patterns, they help a set of systems act on the same underlying platform events in a choreographed way, with each system being notified of a state change in a context sensitive to its own business concerns and maintaining use-specific derived state. Distributed log-processing messaging systems automate this choreography via high-throughput, low-latency message delivery with strong ordering guarantees within partitions [8]. For example, when customers place orders, the order event is published to several consumers that process inventory updates, payment processing, analytics dashboards, machine learning pipelines, and so on, independently, on the same event stream. Choreography is more loosely coupled and resilient compared to orchestration, as there is no centralized controller. In a loosely coupled system based on choreography, if one event consumer fails, the others continue processing events. Simply adding new event consumers adds further functionality, while the preexisting deployed components remain unchanged. This can achieve a balance between consistency, performance, and flexibility with architectures based on event sourcing, materialized view maintenance, and choreographed workflows.

Component	Role in Architecture	Key Capabilities	Integration Patterns	Scalability Characteristics
Event Log	Immutable source of truth	Append-only storage, temporal queries, complete audit trail	Pub-sub, stream processing, event sourcing	Horizontal partitioning, distributed commit log
Stream Processors	Real-time transformation	Filtering, enrichment, aggregation, and windowing operations	Dataflow pipelines, continuous queries	Parallel processing, stateful operations

Materialized Views	Pre-computed aggregations	Incremental maintenance, indexed access, continuous updates	Application serving layer, caching tier	Optimized for readheavy workloads
Event Consumers	Downstream applications	Independent processing, idempotent handlers, offset tracking	Choreographed workflows, reactive systems	Consumer group parallelism, fault isolation

Table 3: Event-Driven Architecture Components [7, 8]

5. Practical Implementation Considerations

Transitioning to a data-first architecture comes with multiple technical and organizational challenges. Transactional and analytical systems in the organization must be unified, carefully planned, and executed to minimize disruption, and the benefits of the unified architecture need time to materialize. In modern elastic data warehouses, compute and storage can be elastically scaled independently to meet specific workloads. Organizations can provision multiple elastic and independent compute clusters that share the same data while keeping the workloads separate from and isolated from each other, both preventing data duplication and maintaining consistency of the data. For example, some cloud-native data warehousing platforms will automatically scale up or down compute resources based on query workload and performance targets.

To efficiently support such workload mixtures, a complex query optimization is needed that detects access patterns and dispatches each query to its appropriate execution engine with an optimal resource allocation. At query compilation time, a decision is made whether the query is a latencysensitive operational query or a throughput-oriented analytical query. The selection is done by consulting the resource pools for latency-sensitive queries and the elastic resource pools for throughput-oriented analytical queries. Resource governors in these pools dynamically limit query execution timeout, memory, and CPU resource allocation to avoid the monopoly of resources by queries. Adaptive execution resource governor automatically optimizes resources by adjusting them based on workload to achieve the most efficient resource usage. Workload management is vital when large numbers of users concurrently query operational applications, which have different access patterns than analysts making ad hoc queries. Techniques such as multi-tier caching, smart partition pruning, and pushdown of predicates can be used to avoid unnecessary scans.

Although modern data platforms support low-latency queries, additional techniques may be used to support sub-second latency applications. These include edge caching, local storage of results from queries with a small time-to-live and bounded staleness, and asynchronous writes that are posted to a local transaction log, which is then synchronously updated by the data platform. The result is eventual consistency. In addition to supporting use cases, this strategy has an economic advantage due to the much lower number of databases, database licenses, and disk space needed. Cost savings can occur when the data platform replaces dozens of application databases and data warehouses. Data deduplication and compression can also contribute. Modern systems achieve predictable performance in the presence of unpredictable workloads through adaptive query processing. This occurs when systems adapt the query execution plan in-flight based on runtime measurements, rather than solely depending on static statistics [10]. These techniques are particularly useful for unbalanced data distributions, time-varying query workloads, and workload interference, as they can help stabilize performance across a variety of operational and analytical access patterns without requiring an important investment in development. Development time can be translated into economic value in these types of systems through the reuse of an existing set of curated datasets (instead of constructing a new schema in each project) and through resource sharing between operational and analytical workloads.

Strategy	**Optimization Target**	**Implementation Technique**	**Measurable Benefits**	**Applicable Scenarios**
Query Routing	Workload segregation	Machine learning classification, resource pool allocation	Reduced contention, improved latency	Mixed transactionalanalytical workloads
Partition Pruning	Scan reduction	Metadata-based filtering, predicate pushdown, zone maps	Minimized I/O, faster queries	Large fact tables, time-series data
Result Caching	Redundant computation elimination	Multi-tier cache hierarchy, TTL-based invalidation	Reduced compute costs, lower latency	Repeated queries, dashboard refreshes
Adaptive Execution	Dynamic plan adjustment	Runtime statistics, midquery re-optimization	Consistent performance across data skew	Unpredictable data distributions

Table 4: Implementation Optimization Strategies [9, 10]

The emergence of "reverse ETL" tools—designed to synchronize data from analytical warehouses back to operational systems—represents an acknowledgment of the limitations inherent in traditional separated architectures. However, these solutions introduce additional complexity through yet another data movement layer, potential consistency gaps, and operational overhead. The data-first architecture proposed in this article eliminates the need for reverse ETL entirely by enabling operational applications to consume directly from curated data layers, treating the data platform as the authoritative source for both analytical queries and operational reads. Rather than adding bidirectional synchronization mechanisms between separate systems, organizations can consolidate on a single platform where applications query the appropriate medallion layer based on their latency and consistency requirements. This approach not only reduces infrastructure complexity but also ensures that all consumers—whether analytical dashboards or user-facing applications—operate on the same consistent view of business data without the reconciliation challenges that plague reverse ETL implementations.

Conclusion

The merging of transactional and analytic workloads on a single data platform is arguably the most fundamental and transformative architectural change in the way organizations build applications and use their information assets. It results in entirely new operational capabilities and business models and erodes the historic divide between systems of record and systems of insight. Leading enterprises are achieving greater agility, consistency, and immediacy in their data management processes, resulting in a newly competitive edge in financial services, retail and e-commerce, and supply chain operations. For example, implementations of data-first architectures have consistently transformed the way companies work, from identifying fraudulent transactions in seconds to delivering a highly personalized customer experience by continually analyzing behavioral trends to optimizing inventory levels through real-time demand sensing to avoid excess inventory and stockouts. Data-first architecture principles are particularly valuable to organizations wanting to deploy artificial intelligence and machine learning solutions at scale, where the accuracy of predictive models depends on access to up-to-date, complete, and reliable training data and on being able to operationalize the model through transactional processes without the overhead of hybrid architectures. As the volume of enterprise data continues to grow exponentially and market conditions dictate a more rapid response, the unified transactional-analytical architecture provides a sustainable, scalable, and cost-effective technical foundation that reduces rather than increases complexity. Organizations using this new architectural model will benefit from emergent technologies such as autonomous AI agents, automated decisioning systems, and predictive analytics. This unified model has none of the friction or latency found in a dual system architecture that uses operational databases for transaction processing and data warehouses for analytics. The suggested plan for moving to a cloud data strategy involves working together to create a step-by-step approach, focusing on important projects that can quickly help the business, gradually making decisions about the hub-and-spoke architecture, training staff to enhance their skills, and ensuring that monitoring and governance are in place to keep the platform strong and compliant during the transition to the new cloud setup. High-functioning transformations combine technical and organizational readiness, where development teams adapt to new programming models, operations staff evolve to become platform experts, and applications migrate incrementally from on-premises or edge sources into consolidated platforms without any service disruptions. The long era of architecturally separated transactional and analytical systems is fading as the technology for distributed computing, columnar storage engines, query optimization, and hybrid processing advances. Once unified architectures become not only possible but also the most cost-effective option compared to previous siloed architectures, organizations that embrace this inflection point and the data-first architectural principles will have decisively helpful footprints that enable them to operationalize new insights in near real time, considerably accelerate application development cycles, and flexibly adapt to shifting business requirements. The architecture thus evolves from a passive warehouse or data lake positioned on the periphery of the organization to become the central nervous system of the digital enterprise, the authoritative unified foundation for operational excellence and analytical understanding generation. This architecture allows companies to take advantage of the increasingly data-driven nature of the world, where the speed of perception generation and the ability to operationalize analytics insights within transactional systems will determine market leadership and long-term business success in the real-time, AI-augmented enterprise of the future. These architectural principles have been validated through multiple enterprise and public sector implementations, where the transition from siloed systems to unified data platforms has consistently delivered measurable improvements in data freshness, operational costs, and decision-making velocity.

References

[1] Andrew Pavlo et al., “Self-Driving Database Management Systems,” CIDR 2017. [Online]. Available: https://www.cs.cmu.edu/~pavlo/papers/p42-pavlo-cidr17.pdf

[2] Michael Stonebraker and Uĝur Çetintemel, ““One size fits all”: an idea whose time has come and gone," Making Databases Work: the Pragmatic Wisdom of Michael Stonebraker, 2018. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/3226595.3226636

[3] Sijie Shen et al., "Bridging the Gap between Relational OLTP and Graph-based OLAP," 2023 USENIX Annual Technical Conference, 2023. [Online]. Available: https://www.usenix.org/system/files/atc23-shen.pdf

[4] Tirthankar Lahiri et al., "Oracle Database In-Memory: A dual format in-memory database," 2015 IEEE 31st International Conference on Data Engineering, 2015. [Online]. Available: https://ieeexplore.ieee.org/document/7113373

[5] Tech Lessons, "A guide to Serializable Snapshot Isolation in Key/Value storage engine," 2024.

[Online]. Available: https://tech-lessons.in/en/blog/serializable_snapshot_isolation/

[6] Neha Narkhede et al., “Kafka: The Definitive Guide,” O'Reilly Media, 2017. [Online]. Available: https://www.oreilly.com/library/view/kafka-the-definitive/9781491936153/

[7] Microsoft Azure, "Event Sourcing pattern," 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/architecture/patterns/event-sourcing

[8] Jays Kreps et al., "Kafka: A distributed messaging system for log processing," Github. [Online]. Available: https://github.com/jeffrey-xiao/papers/blob/master/systems/kafka-a-distributedmessaging-system-for-log-processing.pdf

[9] Benoit Dageville et al., "The Snowflake Elastic Data Warehouse," SIGMOD/PODS, 2016. [Online]. Available: https://www.cs.cmu.edu/~15721-f24/papers/Snowflake.pdf

[10] P. Unterbrunner et al., "Predictable Performance for Unpredictable Workloads," VLDB Endowment, 2009. [Online]. Available: https://www.vldb.org/pvldb/vol2/vldb09-323.pdf