Volume 10, Issue 1, 2026 LGU J. Comput. Sci. & IT
Original Article https://doi.org/10.54692/lgurjcsit.v10i1.197

A Lightweight Middleware for Secure Data Sharing Between Legacy Databases and Blockchain Systems

Najeebullah*, Abdul Haseeb Malik, Qazi Ejaz Ali, Waheed Ur Rehman, Saif Ur Rehman, Inam Ullah

Department of Computer Science, University of Peshawar, Peshawar, Pakistan.

Abstract

Important documents are stored in institutional databases, but they are unable to demonstrate to an external party that they have not been altered. Moving a complete institutional database onto a distributed ledger is impractical due to the expense, technological complexity and infrastructural constraints that most institutions encounter. However, blockchain technology provides a tamper-evident ledger that precisely solves this issue. This paper presents a lightweight middleware that adopts a more practical approach: Hyperledger Fabric saves a cryptographic proof of each committed modification, while the operational database continues to function normally. The middleware tracks database changes in real time using Debezium Change Data Capture. A Node.js pipeline that automatically maps relational schemas using INFORMATION_SCHEMA metadata processes events that are sent via Apache Kafka. It employs SHA-256 hashing to detect duplicate events at the event and content levels, enforces role-based access control, and removes personally identifiable information right before hashing. It generates canonical JSON representations, transmits verified payloads to Hyperledger Fabric with batching, automatic retry, and read-back finality testing, and logs the identity and source of every database actor using MySQL triggers and an audit context table. A web-based monitoring dashboard allows administrators to view the pipeline in real time without requiring any knowledge of blockchain. Different actual batches insert occurrences, and six correctness situations were tested. All of the scenarios were successful. For all workload sizes, throughput remained constant at 0.48 events per second. The findings support steady scalability, policy dependability, and functional correctness in an institutional deployment setting with limited resources.

Keywords: Middleware; Hyperledger Fabric; Blockchain; Legacy Database; Change Data Capture; Debezium; Apache Kafka; Schema Mapping; Duplicate Detection; Actor Attribution; PII Filtering; Data Integrity.

I. Introduction

For many years, relational databases have served as the foundation for institutional data management. They are used by universities to keep academic credentials and student records. Clinical records and patient histories are kept at hospitals. They are used by government offices for land records and civil registrations. These systems are operationally mature, quick, and well-understood. They are unable to demonstrate to an outsider that their data has not been covertly altered.

This is not an engineering failure. Serving the company that owns them is simply the purpose for which centralized systems were designed. However, centralized databases do not provide a means for independent verification when records must be shared across organizational boundaries or when an external auditor wishes to confirm that a specific entry existed before a specific date and was never altered thereafter. There is no neutral ground, and the administration has complete power [1], [2].

The need for reliable institutional information systems, metadata management, and structured data transformation has also been highlighted in previous research on decision support and data warehousing architectures. Such systems require dependable workflow management, structured data processing, and effective metadata utilization to support organizational transparency and informed decision-making [3], [4], [5].

Blockchain technology was created specifically to offer that level playing field. A blockchain ledger is kept up to date by several separate nodes, and numerous participants must agree before a new record can be committed. It is computationally impossible to change a record without notice after it is on the chain and blocks have been built on top of it. Hyperledger Fabric is a good fit for institutional data verification because it uses these characteristics in an enterprise setting where participants are identifiable and access is controlled by established procedures [6].

For daily operations, blockchain cannot take the position of a relational database, which presents a practical barrier. Instead of milliseconds, blockchain

Received: 25 April 2026; Revised: 21 May 2026; Accepted: 03 June 2026
*Corresponding Author: Najeebullah <najeebullah9172@uop.edu.pk>
© 2026 by the Author(s). Published by LGURJCSIT, under the CC-BY license

transactions take seconds. The intricate queries needed for institutional reporting are not supported by the ledger. Furthermore, whenever entire records that contain personal information are kept on a shared ledger, there are significant privacy risks. Keeping the operational database exactly as it is and using the blockchain just for cryptographic evidence, hashes, timestamps, and provenance metadata that let any party independently check whether a particular record has been altered is a sensible approach that is well backed by the literature [6], [7].

This hybrid design philosophy is also consistent with earlier research on semantic transformation and context-aware computing, where heterogeneous information sources were converted into structured and interoperable representations to support automated processing and knowledge sharing across distributed environments [8], [9], [10], [11].

It is more difficult than it seems to implement this hybrid architecture. The database must be continuously monitored for updates without interfering with the programs that rely on it. Every modification must be converted into a hashable, deterministic representation. Before anything is added to the shared ledger, sensitive personal information must be removed. It is necessary to identify and prevent duplicate events. Along with the data on the blockchain, the identity of the individual who performed the modification must be identified. All of these requirements are not met by any of the current integration systems examined for this work [12], [13], [14].

The lightweight middleware presented in this paper was created, tested and designed to meet all of these needs in a single functional system. The middleware uses an event-driven pipeline based on Debezium, Apache Kafka and Node.js to link MySQL to Hyperledger Fabric. Four criteria, correctness, latency, throughput, and scalability, were used to evaluate it. This is how the rest of the paper is structured. In Section II, the pertinent literature is reviewed, and the specific gaps that this work covers are identified. The system design and research methodology, including the proposal architectural diagram, are covered in Section III. The findings of the experiment are shown in Section IV. These findings are discussed in relation to earlier research in Section V. Conclusions and future study directions are outlined in Section VI.



Figure 1. Nine-Step Policy Enforcement Pipeline Implemented in the Proposed Middleware

Figure 1 illustrates the complete policy enforcement workflow implemented in the proposed middleware. Each captured database event passes through a sequence of validation, security and integrity verification stages before being committed to Hyperledger Fabric. This layered processing ensures access control enforcement, privacy preservation, duplicate prevention and verifiable data integrity while maintaining compatibility with existing relational database systems.

II. Literature Review

2.1 Blockchain-Database Integration Approaches

In both academic study and business practice, the concept of employing blockchain as a verification layer on top of an existing database rather than as a substitute for it has gained support. Using Hyperledger Fabric smart contracts as the goal for transformed relational data, Moalagh and Ebrahimi Ghadi [6] illustrated this hybrid paradigm within the framework of an enterprise resource planning system. Their research shows that the two systems can coexist, with the blockchain offering an unchangeable

audit record and the database managing operational duties. A theoretical basis for this strategy was given by Lai et al. [7], who demonstrated that, provided the transformation and hashing processes are deterministic, combining deterministic database behaviour with private blockchain storage can ensure correctness.

Several middleware tools have been put forth to automate the link between blockchain networks and relational databases. Although Sa et al. [12] developed an API-based middleware that showed that data could be transferred between the two systems, their solution necessitated manual schema configuration and did not directly record changes from the database log. Yessenbayev et al. [13] created a more complex system that used batching to reduce latency by routing events through a local private chain before committing them to a public blockchain. However, their solution relied on infrastructure resources that are unavailable in the majority of institutional settings with limited resources. Once more, without automated schema mapping or real-time change capture from the binary log, Bao et al. [15] linked Hyperledger Fabric with a spatial database and verified that the Fabric SDK can function as an efficient submission interface.

Beyond blockchain–database integration, the broader challenge of interoperability has been extensively investigated in semantic transformation and context-aware systems research. These studies emphasize the importance of structured representation, automated knowledge conversion, and adaptive information processing for reliable communication across heterogeneous platforms [8], [9], [10], [11].

2.2 Change Data Capture and Event Streaming

Instead of executing queries against application tables, Change Data Capture reads committed database changes straight from the transaction log. This method is crucial because it records changes at the row level, including the full before and after states of each impacted row, and adds no query load to the production database. The most popular open-source CDC tool for relational databases is now Debezium. The database server is unaware that CDC is taking place because it connects to MySQL using the same replication protocol used by database replicas. For each committed row change, Debezium sends out a structured Kafka message that contains rich metadata from the binary log, such as the file name, the location inside the file, the server identifier, the thread identity, and the commit timestamp [16], [17].

The event transport layer that connects Debezium to the middleware processing pipeline is provided by Apache Kafka. Kafka sends messages in partition order, stores them for a long time, and uses offset management to let users know where they are. This implies that the middleware can continue exactly where it left off without losing any events if it pauses abruptly. The middleware requires idempotency constraints because of Kafka's at-least-once delivery guarantee: Kafka may provide the same event more than once during recovery and the middleware must be able to identify and prevent duplicate processing [16].

The staged event-processing architecture adopted in this work is conceptually aligned with previous ETL workflow management and data warehousing studies, where layered transformation pipelines were shown to improve reliability, maintainability, and decision-support effectiveness in large-scale information systems [3], [4].

2.3 Privacy, Governance and Provenance

When complete records with personally identifiable information are stored on a shared blockchain ledger, privacy compliance issues arise. The conventional approach is to maintain the raw data in the database and simply store cryptographic hashes and minimal metadata on the ledger. However, because adding a sensitive value to the canonical representation commits the middleware to that value even if the raw record is never shared, the middleware must actively filter sensitive fields before calculating the hash [18], [19].

The usefulness of blockchain-based access auditing in electronic health record systems was shown by Ullah et al. [18], where the chain offered a tamper-evident record of who accessed what and when. Unauthorized events never become ledger records because the middleware in this paper imposes access control before any data enters the blockchain, in contrast to their system, which implemented access control at the application layer. Sultanum et al. [19] made the more general claim that provenance metadata, which includes information about the origin of data, who handled it and when, is essential to institutional data trust.

2.4 Identified Research Gaps

A number of gaps consistently show up when reading the reviewed literature. Automated schema mapping powered by real-time database metadata is not implemented by any middleware solution under study. Although it is rarely shown in real middleware implementations, real-time Change Data

Capture from the binary log is theoretically presented. At the database session level, actor identity, the particular user account and host that carried out a database operation is hardly ever recorded and kept on chain. All of the systems under study lack content-level duplicate detection, which identifies instances in which the same business data enters through various database operations. None of the systems under examination have access control or PII filtering implemented at the middleware layer instead of the application layer. Interfaces for operational monitoring are not offered. It is uncommon to evaluate performance in resource-constrained contexts using actual batch workloads. All of these gaps are filled in this paper.

III. Framework and Methodology

An experimental prototyping strategy was used in the research methodology. The entire system was created, put into use, tested in real-world database environments, and improved through iterative debugging until consistent, stable end-to-end operation was attained. Developing a real-time data pipeline that detects database changes and executes blockchain transactions; designing an algorithm for automatic RDBMS-to-blockchain schema mapping; and assessing the middleware in terms of latency, throughput, scalability, and correctness. The prototype data flow diagram is shown in Figure 2.

The proposed framework follows the broader principle that institutional information systems require structured transformation pipelines, metadata-driven processing, and workflow-oriented data management to ensure dependable information exchange across heterogeneous platforms and organizational boundaries [3], [4], [5].

3.1 Automatic RDBMS-to-Blockchain Schema Mapping

Creating an algorithm that could automatically convert a relational database schema to a blockchain-compatible format without the need for manual configuration was the initial goal. The majority of current integration systems address this by having a developer create a fixed mapping file that outlines the tables and columns that need to be recorded. Because the mapping file does not update itself, this method fails as soon as the database schema is altered. The algorithm created here adopts a different strategy: each time an event is processed, it queries the live database metadata.

The middleware immediately executes two queries against INFORMATION_SCHEMA after extracting the database and table names from the event source block upon receiving a CDC event. In order to maintain the original column definition sequence, the first query targets INFORMATION_SCHEMA.COLUMNS to obtain the name, data type, nullability flag and key designation for each column in the source table, arranged by ORDINAL_POSITION. Only true foreign key constraints are returned by the second query, which targets INFORMATION_SCHEMA.KEY_COLUMN_USAGE ,restricted to rows where REFERENCED_TABLE_NAME is not null. Primary key and unique index items that also occur in that view are excluded.

The metadata-driven mapping strategy employed in the middleware shares similarities with earlier semantic transformation approaches, where knowledge structures were systematically converted into alternative formal representations to improve interoperability, automated reasoning, and knowledge management capabilities [8], [9], [11].

The algorithm creates a schema block from these two query results, which includes the table name; is a list of column descriptors that each contain the column name, data type, a boolean primary-key flag, a boolean nullable flag, a list of primary key column names gathered separately for direct reference, a list of foreign key hint objects that each record the local column, the referenced table and the referenced column and a mapperMode field that is set to the value metadata-driven. The final field is crucial for auditability: It verifies that the schema was acquired via INFORMATION_SCHEMA introspection as opposed to a manually updated configuration file in the ledger record itself. Every blockchain payload includes the schema block, thus the Hyperledger Fabric ledger contains both the modified data and a comprehensive structural description of the source table at the time the modification was recorded.

3.2 Real-Time Data Pipeline Design and Operation

Building a data pipeline that records RDBMS updates in real time and completes the associated blockchain transactions was necessary. The entire pipeline architecture as suggested and put into practice is depicted in Figure 2.

3.3 Proposed Middleware Architecture

The application tables and an audit_context table, which is kept up to date by three database triggers,

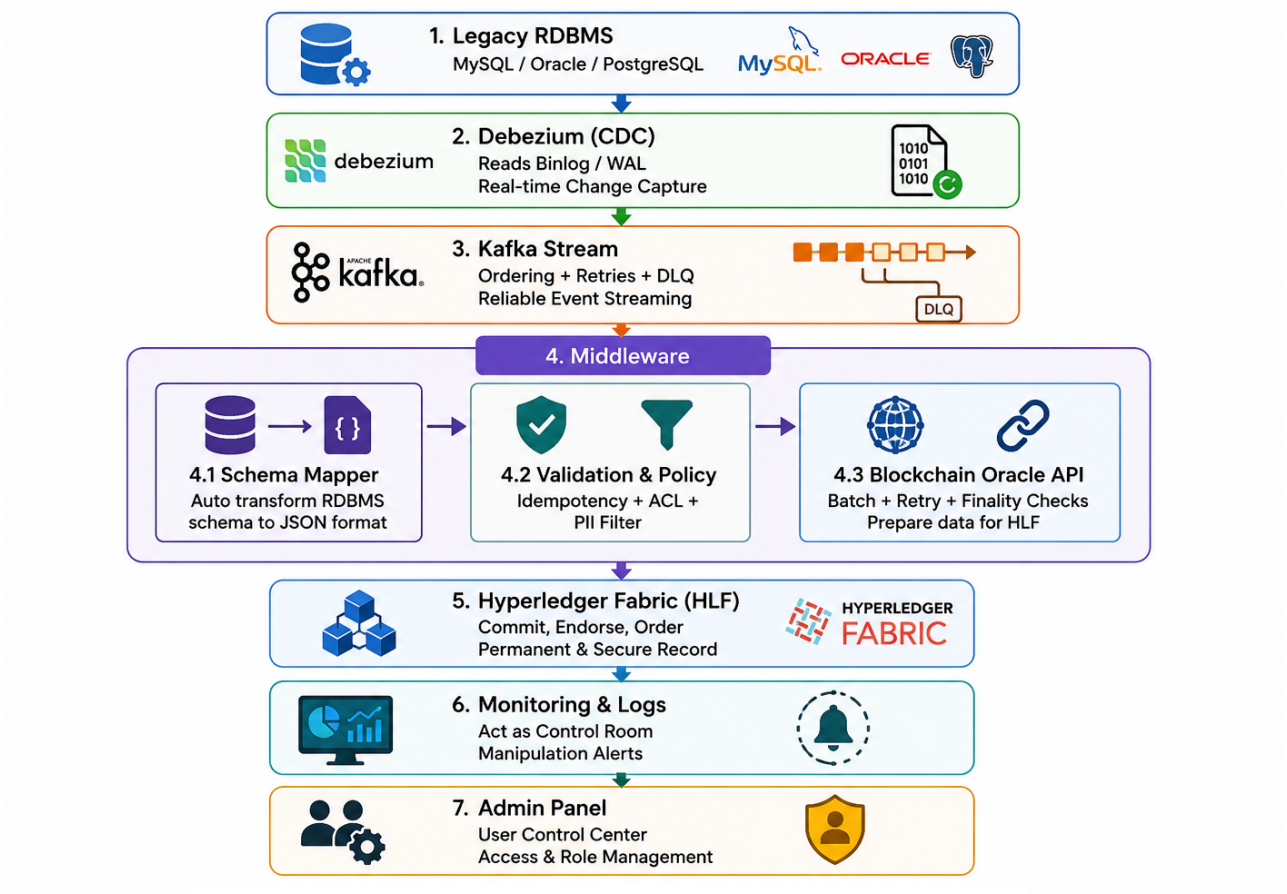


Figure 2. Data Flow From Legacy RDBMS to Hyperledger Fabric

are stored in the MySQL source database, where the pipeline starts. A row containing the actor identifier from MySQL USER(), the inferred actor role, the source system name, the source host from @@hostname, the source channel, the database user from CURRENT_USER(), the session user and the event timestamp is written by the triggers, which fire synchronously prior to each insert, update and delete on the monitored tables. The audit_context row is always accessible when the middleware subsequently requests for it since the triggers are triggered prior to the operation being committed to the binary log.

Debezium uses the server replication protocol to connect to MySQL, keeps an eye on the binary log and sends out a structured Kafka message for every committed row update. Together with the binlog file name and position, each message includes the before and after row images, which together create a globally unique identifier for that particular committed operation.

These messages are received by Apache Kafka, which stores them indefinitely until the middleware consumer retrieves and processes them.

Every Kafka event is picked up by the Node.js middleware consumer, which then runs it through a nine-step policy enforcement pipeline. PII fields are eliminated in the first step from a functional duplicate of the row image. The binlog file name, location, row number and operations type are used in step two to create the event key. In order to find event-level duplicates, step three compares this event key to both the Hyperledger Fabric ledger and an in-memory cache. The SHA-256 hash of a canonical object with the database name, table name and filtered record is used in step four to calculate a content key. In order to identify content-level duplicates, step five compares this content key to the ledger. This identifies instances in which the identical data is received through a different database operation with a different event key. The actor and origin details for the current event are retrieved in step six by querying audit_context. The actor role is compared to the ACL configuration file in step seven. The eighth step confirms that the filtered record has all necessary fields and is not empty. When every check is successful, step nine returns the accepted result set.

The middleware then builds the schema block for accepted events using the schema mapping algorithm, assembles the entire payload, recursively sorts all object keys alphabetically to create a canonical JSON string, calculates the final SHA-256 hash, and puts the payload in a batch queue. Each payload is

submitted via the StoreHash chaincode function on Hyperledger Fabric; a read-back finality check is performed by calling ReadHash and comparing the returned record against the submitted payload; finality is confirmed when the event key, hash, and content key all match. The blockchain oracle flushes the batch queue when a size threshold or timer interval is reached. Unsuccessful submissions are resubmitted with gradual backoff delays, and instances where all efforts are made are noted in a recovery log for future resubmission without data loss.

The middleware's structured log files are read by the monitoring and administrative panel at localhost:3001, which then displays accepted event counts, rejected event counts, average latency, and categorized lists of recent occurrences. The panel is correct even when the middleware is stopped because it reads from log files instead of the active middleware process.

To provide a consolidated view of the security and integrity controls incorporated within the proposed middleware, Table 1 summarizes the major protection mechanisms and their respective roles. These mechanisms collectively ensure secure event processing, privacy preservation, traceability, and tamper-evident verification before data are committed to the blockchain network.

Table 1. Security and Integrity Mechanisms Implemented in the Proposed Middleware

Mechanism	Purpose
ACL Validation	Prevent unauthorized blockchain submissions
PII Filtering	Remove sensitive information before hashing
SHA-256 Hashing	Generate tamper-evident integrity proofs
Duplicate Detection	Prevent repeated event processing
Actor Attribution	Record user and source system identity
Finality Verification	Confirm successful blockchain commitment
Audit Logging	Maintain traceability of all operations
Schema Mapping	Automatically adapt to schema changes

The incorporation of actor identity, contextual metadata, and provenance information is also consistent with context-aware computing frameworks, where environmental and user-related context are treated as essential components for accountability, traceability, and adaptive system behavior [8], [10].

3.4 Evaluation Plan

Evaluating the middleware in relation to four distinct dimensions, latency, throughput, scalability and correctness, was necessary. The amount of time that passes between the Kafka consumer receiving a CDC event and Hyperledger Fabric verifying the transaction following the read-back finality check is known as latency. The number of accepted events divided by the total processing time in seconds was used to compute throughput. By examining pipeline behaviour across two distinct workload levels and determining if performance declined as the number of events rose, scalability was evaluated. Six different test scenarios encompassing every operation type and every policy consequence were used to assess correctness.

Every experiment was carried out on a Windows computer running MySQL 8.0 with binary logging enabled in a Docker-based environment using WSL. ZooKeeper, Kafka Connect, and Kafka 3.4.0 operated as independent containers. In order to monitor the testdb database, Debezium 2.3.4 was installed as a MySQL connector plugin. With two peer organizations and a two-organization endorsement strategy, Hyperledger Fabric 2.x functioned as the typical test network. Using the Org1 administrator account to connect to Fabric via the SDK gateway, the middleware consumer and monitoring panel operated as Node.js processes in WSL.

The use of multiple evaluation dimensions, including correctness, throughput, latency, and scalability, is consistent with prior decision-support and comparative data mining studies, where a combination of performance and reliability metrics was used to assess practical deployment suitability [5], [20].

IV. Implementation

4.1 Correctness Evaluation

Six test scenarios were used to assess correctness. Five criteria were applied to the middleware console output for each scenario: the filtered record matched the source row with only the designated PII fields removed; the actor and origin fields matched the audit_context entry for that operation; the SHA-256 hash was a valid digest of the canonical JSON string of the assembled payload; the ledger response verified on-chain storage with field values identical to those in the submitted payload; and the decision status matched what was expected for the scenario type. The results are summarized in Table 2.

The anticipated result was obtained in each of the six cases. Tests 1 through 3 verified that the

Table 2. Correctness Evaluation Using Six Scenarios, 100% Pass Rate

Test	Operation	PII Filtered	Actor / Origin	Hash Correct	Ledger Stored	Decision	Result
1	Insert (c)	Yes	Yes	Yes	Yes	ACCEPTED	Pass
2	Update (u)	Yes	Yes	Yes	Yes	ACCEPTED	Pass
3	Delete (d)	Yes	Yes	Yes	Yes	ACCEPTED	Pass
4	Duplicate Event	Yes	Yes	Yes	No	REJECTED_DUP_EVENT	Pass
5	Duplicate Content	Yes	Yes	Yes	No	REJECTED_DUP_CONTENT	Pass
6	ACL Violation	Yes	Yes	Yes	No	REJECTED_ACL	Pass

Table 3. Latency Measurement Results

Batch	Events	Min (ms)	Max (ms)	Average (ms)	Total Time (s)	Acceptance Rate
Batch 1	10	2033	2138	2073	20.73	100%
Batch 2	20	2011	2113	2062	41.24	100%
Batch 3	30	2011	2217	2066	61.97	100%
Batch 4	50	2050	2397	2089	104.45	100%
Batch 5	100	2044	2128	2082	208.20	100%
Batch 6	500	2076	2164	2071	1035.50	100%
Batch 7	710	2106	2214	2075	1410.12	100%

pipeline applies PII filtering, actor attribution, schema mapping, canonicalization, hashing and blockchain finality verification in each instance, appropriately transforming and committing valid insert, update and delete events. Tests 4 through 6 verified that each category of non-compliant event is accurately identified, blocked and assigned the proper rejection status by the policy enforcement layer. The passing of rejection tests is just as crucial as the passing of acceptance tests: a system does not offer a meaningful integrity guarantee if it simply permits valid events without also preventing illegitimate ones.

4.2 Latency Results

Two batch workloads were used to measure latency. Ten department records with dept_id values 101 through 110 were added in the first batch. Twenty records with dept_id values ranging from 111 to 130 were added in the second batch and so on. The results are shown in Table 3.

Each batch's minimum and maximum latency ranged by about 100 ms, which is tiny in comparison to the overall per-event duration and suggests a steady pipeline free of unpredictable behaviour. The Hyperledger Fabric peer connection warm-up is reflected in the slight decrease in average latency from the first batch to the second. This is because some connection establishment overhead from the initial few events is amortized over the course of the session as the peer maintains an active endorsement pipeline across multiple consecutive transactions. The Fabric endorse-order-commit cycle, which takes about two seconds per transaction in the test network environment, is the main cause of latency in every event.

To further illustrate the runtime behavior of the proposed middleware, Figure 5 presents the complete event processing sequence from a database modification to blockchain commitment. The sequence demonstrates how changes captured through MySQL binary logs are propagated through Debezium and Apache Kafka before entering the middleware validation pipeline and ultimately being committed to Hyperledger Fabric.

Figure 4 illustrates the relationship between workload size and processing time for the proposed middleware. The results demonstrate a near-linear increase in execution time as the number of processed events grows, indicating predictable and stable scalability characteristics.

As shown in Figure 4, the middleware maintains consistent behavior across all evaluated workload sizes. The linear growth trend confirms that no significant bottlenecks or resource saturation effects were observed during testing. The results indicate that the proposed architecture can accommodate increasing transaction volumes while preserving stable throughput and processing efficiency.

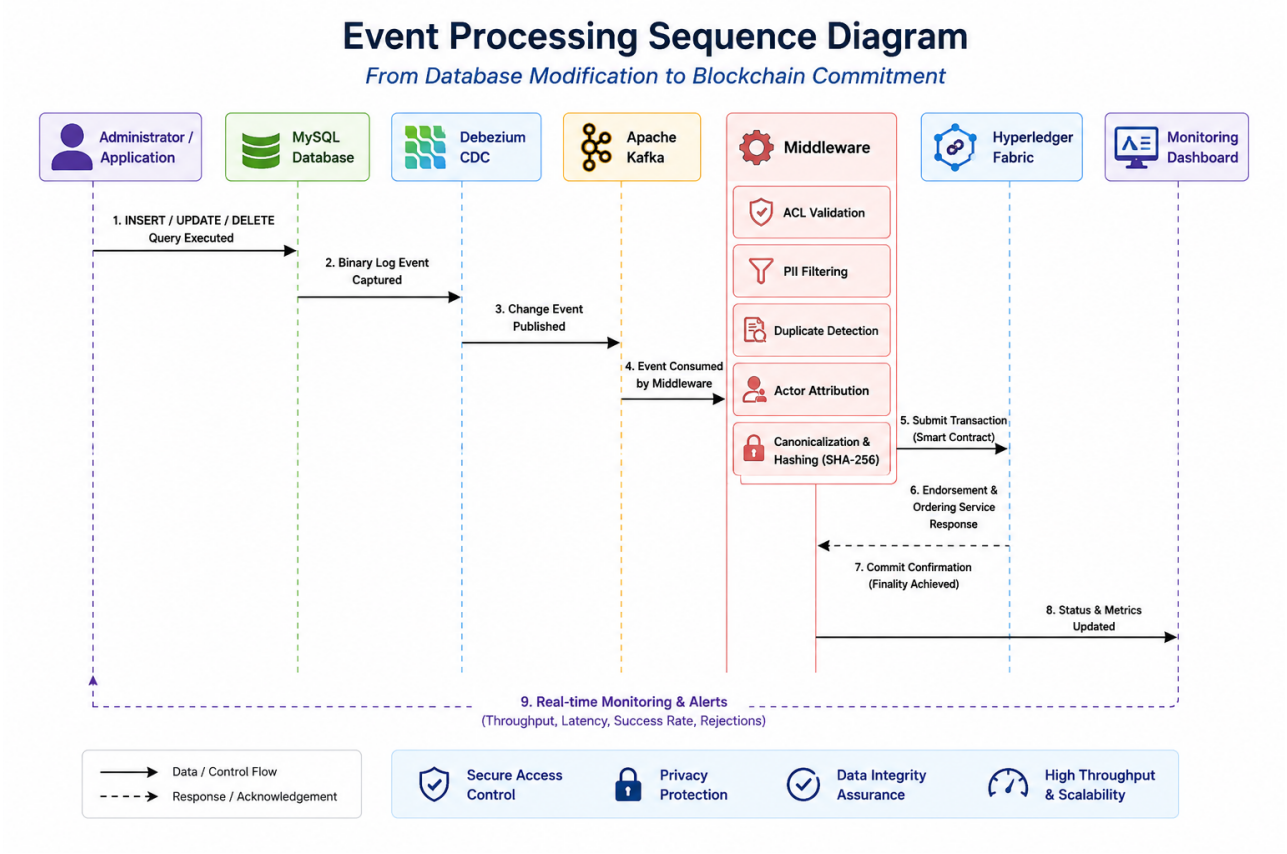


Figure 3. Event Processing Sequence from Database Modification to Blockchain Commitment

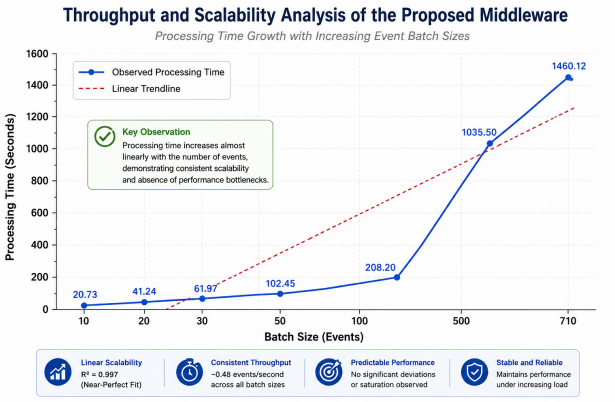


Figure 4. Processing Time Growth with Increasing Event Batch Sizes Demonstrating Linear Scalability

4.3 Throughput and Scalability Results

Throughput was computed by dividing the number of accepted events by the overall processing time for a batch. This results in 10 / 20.73 = 0.48 events per second for the ten-event batch. It yields 20 / 41.24 = 0.48 occurrences per second for the twenty-event batch. When all thirty occurrences are added together, the result is 30/61.97 = 0.48 events per second, and so on. The scalability comparison between the two workload sizes is shown in Table 4.

All three measurements yielded the same throughput figure of 0.48 events per second. Among the seven batch sizes, the total processing time precisely doubled, indicating constant per-event overhead without accumulation effects. As the workload grew from 10 to twenty to thirty events, the system did not deteriorate. Rather than any limit in the middleware itself, the per-event Fabric transaction delay determines the throughput ceiling in the test scenario. Throughput would be increased and per-transaction latency would be reduced in a production Fabric deployment with dedicated endorsing peers and an optimized endorsement policy.

4.4 Monitoring Panel Verification

Over the course of the entire experimental session for 10, 20, 30 and 50 events, the monitoring panel at localhost:3001 recorded 66 accepted events and 7 rejected events, with an average session latency of 2168 ms. Five REJECTED_DUPLICATE_CONTENT events and two REJECTED_ACL events comprised the seven rejections. This breakdown verified that the policy enforcement mechanisms were active and operating correctly, not only during isolated test runs but throughout the entire session. The log-reading mechanism correctly captures the real system behaviour, as seen by the panel statistics agreement with the individually measured batch values.

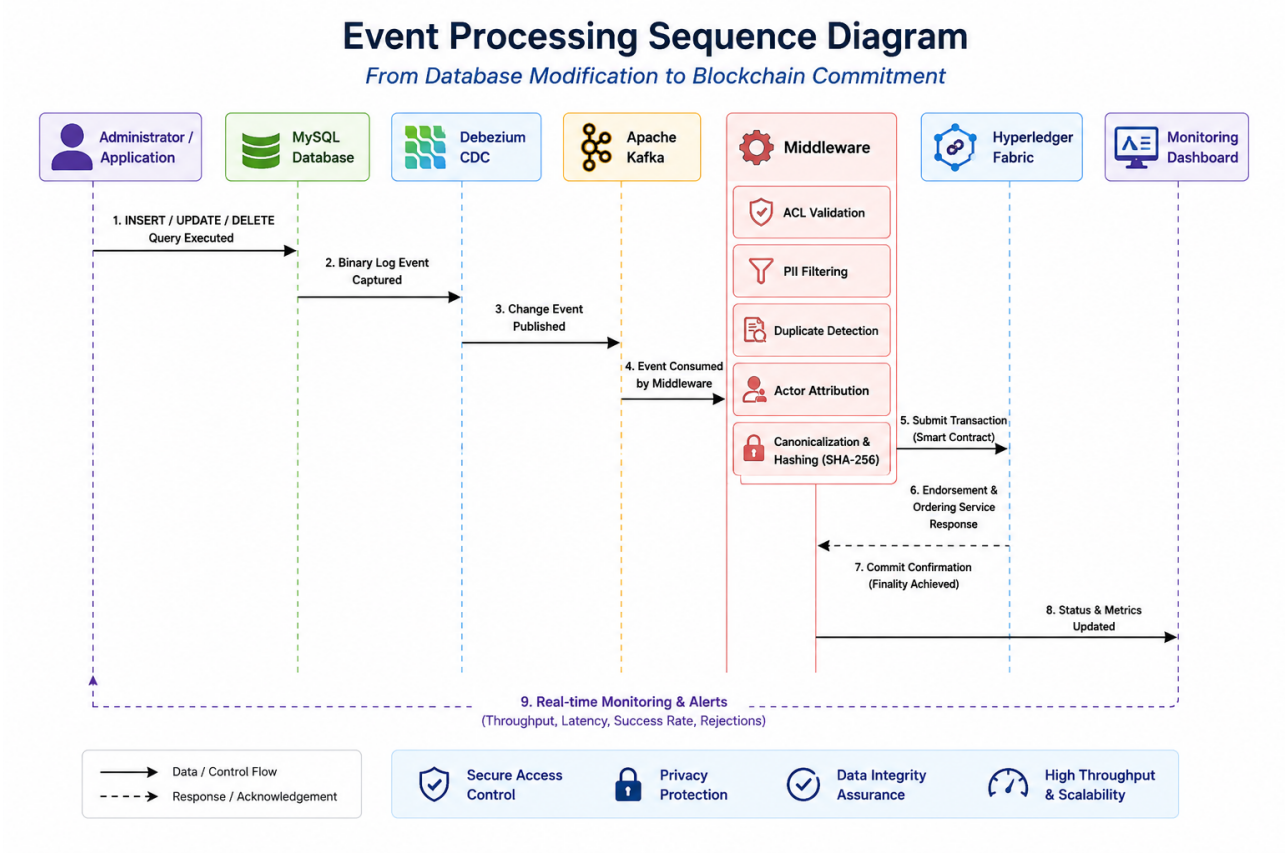


Figure 5. Event Processing Sequence from Database Modification to Blockchain Commitment

Table 4. Throughput and Scalability Results

Metric	Batch 1	Batch 2	Batch 3	Batch 4	Batch 5	Batch 6	Combined	Observation
Total Processing Time (s)	20.73	41.24	61.97	102.45	208.20	1035.50	1460.12	Scales exactly linearly
Average Latency (ms)	2073	2062	2066	2089	2082	2071	2075	Stable slight warm-up effect
Throughput (events/s)	0.48	0.48	0.48	0.48	0.48	0.48	0.48	Identical across all sizes
Success Rate (%)	100	100	100	100	100	100	100	No degradation under load

V. Discussion

5.1 Comparison with Related Work

In Table 5, the implemented system is compared to the most similar previous middleware systems in nine areas that influence practical deployability. Instead of only hypothetical suggestions, the comparison is limited to systems that have actually been constructed and presented functional prototypes.

All nine dimensions are covered by the current implementation, whereas only two or three are covered by each of the previously reviewed systems. Although Sa et al. [12] showed that relational and blockchain databases can be connected via an API-based middleware, they did not record changes from the database log and instead depended on a manually maintained schema configuration. Batching and a partial duplicate detection method were implemented by Yessenbayev et al. [13], but their system relied on infrastructure availability, which is unrealistic in low-resource environments. Although they did not try schema mapping, CDC, or monitoring, Bao et al. [15] verified the Fabric SDKs feasibility for blockchain submission. The database session identity of the actor who carried out each operation and stored that identity on the blockchain was not recorded by any of the systems under examination.

Table 5. Feature Comparison with Related Middleware Systems

Feature	This Work	Sa et al. [12]	Yessenbayev [13]	Bao et al. [15]	Tahora [21]
Automatic Schema Mapping	Yes	No	No	No	No
Real-Time CDC from Binlog	Yes	No	No	No	No
Actor Attribution on Chain	Yes	No	No	No	No
Dual Duplicate Detection	Yes	No	Partial	No	No
PII Filtering in Middleware	Yes	No	No	No	No
ACL Enforcement in Middleware	Yes	No	No	No	No
Batch + Retry + Finality Check	Yes	No	Partial	No	No
Monitoring and Admin Panel	Yes	No	No	No	No
Real Batch Evaluation Data	Yes (710 events)	No	Partial	No	Partial

5.2 Significance of the Results

The most fundamental outcome is the 100% accuracy rate across all six scenarios, including the three rejection scenarios. A middleware does not truly offer a meaningful integrity guarantee if it processes lawful events correctly but does not prevent unauthorized, duplicated, or incomplete events. The accuracy evaluations of two sides are equally important.

A different interpretation is given to the consistent throughput of 0.48 events per second across all workload sizes. This throughput offers a highly comfortable operating margin for the institutional use cases that inspired this study, where database changes are produced by human administrators at rates measured in events per hour rather than events per second. The pipeline has no accumulation or degradation effects within the tested range, which is the practical scalability result that institutional IT administrators would need to observe before contemplating implementation. This is confirmed by the similar figure throughout the ten-event and twenty-event batches.

The monitoring panel introduces operational openness, a factor that is simple to ignore but actually crucial for institutional adoption. It is challenging to trust and troubleshoot a system that processes events correctly but provides administrators with no visibility into what it is doing. The panel makes the pipeline visible to non-blockchain engineers, which is a practical prerequisite for implementation in the majority of institutional settings.

The observed results further reinforce findings from previous research in semantic transformation, ETL workflow management, context-aware systems, and decision-support architectures. Collectively, these studies highlight the importance of structured data processing, operational transparency, traceability, and dependable information exchange in institutional environments [3], [4], [5], [8], [10].

VI. Conclusions

This study introduced a lightweight middleware that uses an event-driven, policy-enforcing Change Data Capture pipeline to link a MySQL relational database to a Hyperledger Fabric blockchain network. The solution solves a practical issue that many organizations in developing nations encounter: the requirement for externally verifiable, tamper-evident data records without the expense and inconvenience of updating current database infrastructure.

To create a metadata-driven schema block that automatically changes when database schemas change and doesn't require human reconfiguration, the automatic schema mapping algorithm searches INFORMATION_SCHEMA during event processing time. Real-time database changes are captured by the data pipeline from the MySQL binary log, routed through Apache Kafka and processed through a nine-step policy enforcement pipeline that filters personally identifiable information (PII), enforces access control, detects duplicates at the event and content levels, attributes each event to its source actor and system, canonicalizes and hashes the payload and submits it to Hyperledger Fabric with retry and finality verification. The assessment verified 100% accuracy in six different scenarios, a steady throughput of 0.48 events per second in two workload levels, linear scalability without deterioration, and precise monitoring panel statistics throughout 73 processing cycles.

The Hyperledger Fabric transaction latency in a single-machine test scenario is reflected in the throughput ceiling of 0.48 events per second, which is not a feature of the middleware architecture. Throughput would be significantly increased in a production Fabric network with specialized hardware and an optimized endorsement strategy.

This work could be significantly expanded in a number

of ways. By using composite primary keys and junction table identification, the schema mapper might be expanded to manage many-to-many relationships. When ACL rules or PII field lists change, the monitoring panel might support policy modifications via the web interface, removing the requirement to restart the middleware. To more accurately describe the scalability threshold, throughput might be assessed at bigger workload sizes of 50, 100 and 200 events. With the appropriate Debezium connectors, PostgreSQL and Oracle might be supported by the CDC pipeline. In contrast to a controlled laboratory setting, a pilot deployment at the University of Peshawar would yield input from actual institutional officials. The endorsement behavior pertinent to actual data-sharing consortium deployments amongst institutions would be tested using a multi-organization Fabric network.

VII. REFERENCES

[1] A. Chandan, V. Potdar, and M. John, "Systematic literature review of blockchain technology's technical challenges: A tertiary study," *Information*, vol. 15, p. 475, 2024.

[2] A. Alzahrani et al., "Design and implementation of a decentralized trustless data standardization framework for blockchain interoperability using smart contracts," *Alexandria Engineering Journal*, vol. 129, pp. 168–191, 2025.

[3] M. S. Rehman, S. U. Khan, et al., "Revised framework for etl workflow management for efficient business decision-making," *International Journal of Computer Theory and Engineering*, vol. 5, no. 3, 2013.

[4] S. U. Khan et al., "Data warehouse enhancement manipulating materialized view hierarchy," in *International Conference on Data and Information Management*, IEEE, 2013.

[5] S. U. Khan et al., "Design and analysis of human resource decision support systems for academic institutions," in *ICOMS*, 2013.

[6] M. Moalagh and A. E. Ghadi, "Blockchain-based erp system: Architecture and opportunities for future," *Journal of Information Technology Management*, vol. 14, pp. 211–243, 2022.

[7] Z. Lai, C. Liu, and E. Lo, "When private blockchain meets deterministic database," *Proceedings of the ACM on Management of Data*, vol. 1, no. 1, p. 98, 2023.

[8] H. M. U. Haque, S. U. Khan, and I. Hussain, "Semantic knowledge transformation for context-aware heterogeneous formalisms," *International Journal of Advanced Computer Science and Applications*, vol. 10, no. 12, 2019.

[9] S. U. Khan, "Transformation of semantic networks into frames," *International Journal of Innovation, Management and Technology*, vol. 4, no. 1, 2013.

[10] H. M. U. Haque and S. U. Khan, "A context-aware reasoning framework for heterogeneous systems," in *Proceedings of ICACS*, IEEE, 2018.

[11] S. U. Khan, M. Khan, and M. Nauman, "Semi-automatic knowledge transformation of semantic network ontologies into frame structures," in *INTECH*, IEEE, 2016, pp. 712–717.

[12] R. A. Sa, L. O. Moreira, and J. C. Machado, "Improving interoperability between relational and blockchain-based database systems: A middleware approach," in *Proceedings of SBBD 2023*, 2023, pp. 115–127.

[13] O. Yessenbayev et al., "A middleware for hybrid blockchain applications: Towards fast, affordable and accountable integration," in *Service-Oriented Computing*, 2023, pp. 307–322.

[14] C. Zhu et al., "A survey on the integration of blockchains and databases," *Data Science and Engineering*, vol. 8, no. 2, pp. 196–219, 2023.

[15] Y. Bao et al., "Spatial blockchain: Enhancing spatial queries through integrating blockchain and spatial database technologies," *Electronics*, vol. 12, no. 20, 2023.

[16] S. Banerjee, "Modernizing healthcare master data management (mdm): Harnessing real-time processing, iot and blockchain," *International Journal of Computing and Engineering*, vol. 7, pp. 24–39, 2025.

[17] M. Ataei et al., "Publish/subscribe method for real-time data processing in massive iot leveraging blockchain for secured storage," *Sensors*, vol. 23, no. 24, p. 9692, 2023.

[18] F. Ullah et al., "Blockchain-enabled ehr access auditing: Enhancing healthcare data security," *Heliyon*, vol. 10, no. 16, e34407, 2024.

[19] N. Sultanum, D. Bromley, and M. Correll, "Data guards: Challenges and solutions for fostering trust in data," in *Proceedings of IEEE VIS 2024*, 2024.

[20] S. U. Khan, "Classification of parkinson's disease using data mining techniques: A comparative analysis," *Journal of Parkinson's Alzheimer's Disease*, vol. 2, no. 1, 2015.

[21] S. Tahora et al., "Blockchain technology in higher education ecosystem," in *Proceedings of the IEEE 47th Annual COMPSAC*, 2023, pp. 1047–1056.