Enhancing Data Interoperability in Multi-Platform Lakehouses with Apache Iceberg

Muhammad Hassan Shafiq, Zheying Zhang[0000−0002−6205−4210], and Kostas Stefanidis[0000−0003−1317−8062]

Tampere University, Tampere, Finland
hassanshafiq17@outlook.com,zheying.zhang@tuni.fi,konstantinos.stefanidis@tuni.fi

Abstract. Managing data across diverse platforms poses significant challenges, including data duplication, vendor lock-in, and inconsistent governance. Lack of a unified table format often leads to complex pipelines, increased storage costs, and hindered interoperability. Apache Iceberg, with its platform-agnostic design, presents a solution by providing a consistent table format for large-scale analytical workloads while addressing cross-platform data accessibility. In this paper, we study the use of Apache Iceberg as a unified table format to enable interoperability between Snowflake and Databricks, with data stored on Amazon S3. Experimental setups include accessing Snowflake-managed Iceberg tables in Databricks and vice versa. Key focus areas include examining query performance, metadata synchronization, and the challenges of managing consistent data across platforms. Optimization strategies, specifically data reordering, were applied to test improvements in query performance for various workloads. The results show that Iceberg reduces the complexity of data management by automating metadata handling and synchronization, ensuring real-time data consistency. Query performance showed improvement in medium-complexity queries with optimized Iceberg tables, while highlighting potential areas for further optimization in full-table scans. These findings underscore Iceberg's potential as a scalable, efficient solution for modern data lake architectures.

Keywords: Data Interoperability · Metadata Synchronization · Data Lakes.

1 Introduction

The rapid expansion of data in enterprises has driven organizations to adopt cloud-based solutions due to their flexibility, ease of resource provisioning, and cost-effective pay-as-you-go models. Traditional data warehouses have been widely used for structured data processing, offering strong consistency and query optimization. However, they are expensive and often lack the scalability needed for handling semi-structured and unstructured data [1,7,11], leading to the rise of data lakes. While data lakes provide scalable and cost-efficient storage, they suffer from governance issues, data inconsistency, and slow query performance, making them inefficient for enterprise analytics [6]. To address these challenges,

the data lakehouse architecture has emerged, combining the transactional support of data warehouses with the flexibility and scalability of data lakes [2].

According to a report from Virtana [12], over 80% of enterprises have adopted a multi-cloud strategy, with 78% running workloads across more than three public cloud platforms. Enterprises use these multiple cloud platforms, including Databricks [1], Snowflake[2], and others to address diverse needs: some excel in big data and machine learning, while others specialize in SQL analytics and business reporting. This multi-platform approach introduces significant challenges, including data duplication, manual synchronization, and increased storage costs. Moving large volumes of data between platforms requires custom integration pipelines, leading to high operational overhead and governance complexities. Furthermore, metadata management varies across platforms, making schema evolution and access control enforcement inconsistent. Query performance also differs due to variations in storage optimization and indexing strategies. As Mone has pointed out, "Metadata management has emerged as a significant bottleneck in big data systems, where different platforms often have disparate ways of accessing and managing data" [3]. Without a standardized approach, ensuring real-time data synchronization across platforms remains a major challenge.

To mitigate these issues, organizations require a unified table format that facilitates efficient metadata handling, transactional consistency, and interoperability across cloud platforms. Furthermore, a unified table format would enhance cross-platform data sharing and accessibility. By having a common metadata structure, organizations could streamline the integration of data pipelines, reducing the risk of data mismatches and versioning errors. This consistency would support more accurate analytics and reporting, enabling them to make data-driven decisions with greater confidence. As multi-platform data ecosystems expand, a unified table format offers a clear path to building flexible, efficient architectures that leverage each platform's strengths without the complexity of managing separate systems. It enables a single source of truth, reduces operational overhead, and supports diverse analytics without redundant data copies. Several open table formats, including Delta Lake[3], Apache Hudi[4], and Apache Iceberg[5], have been developed to address these needs. While Delta Lake is tightly integrated with the Databricks ecosystem and Apache Hudi is optimized for real-time ingestion, Apache Iceberg has gained significant traction due to its vendor-neutral architecture, broad compatibility, and advanced data management capabilities. Iceberg provides ACID (Atomicity, Consistency, Isolation, Durability) transactions, hidden partitioning, schema evolution without full table rewrites, and time travel for historical analysis. Its ability to support

[1] https://www.databricks.com/
[2] https://www.snowflake.com/en/emea/
[3] https://delta.io
[4] https://hudi.apache.org/
[5] https://iceberg.apache.org

multiple query engines, including Spark[6], Trino[7], Flink[8], Hive[9], Databricks[10], Snowflake[11], and Redshift[12], makes it a strong candidate for enabling multi-platform data interoperability.

This work aims to understand the structure of Apache Iceberg, its features, how it fits in the modern data architecture for cross-platform integrations, and the reasons behind its growing adoption across cloud providers. This study also explores the use of Iceberg as a unified table format for cross-platform data access between Snowflake and Databricks, leveraging S3 [13] as the underlying storage layer. It aims to assess query performance across different workloads, analyzing execution latency for simple, medium, and complex queries. Additionally, it investigates Iceberg's ability to maintain real-time data synchronization and schema evolution across platforms, ensuring consistency without requiring extensive manual intervention. By identifying potential limitations and areas for improvement, this research contributes to a deeper understanding of Iceberg's role in modern multi-platform data architectures and its potential to drive seamless, efficient, and interoperable data management.

The rest of the paper is structured as follows: Section 2 covers related work; Section 3 uncovers the current integration option and structure of Iceberg Table; Section 4 presents a bi-directional integration between Snowflake and Databricks; Section 5 outlines the experimental design; Section 6 presents results; and Section 7 concludes with key contributions.

2 Background and Related Work

In the early days of data management, data was primarily stored in flat files, which provided a simple storage mechanism but suffered from redundancy, inconsistency, and lack of efficient query capabilities. As data volumes increased, Relational Database Management Systems (RDBMS) emerged, inspired by the Relational Model [4], to offer structured storage with ACID transactions ensuring data integrity. While RDBMS provided efficient indexing and query optimization, they were limited by proprietary storage formats that were tightly coupled with their specific implementations. The increasing demand for large-scale distributed data processing led to the development of Hadoop [4], which introduced the Hadoop Distributed File System (HDFS) for scalable storage and MapReduce as a parallel processing framework. However, writing complex MapReduce jobs required significant expertise, limiting its usability. To address this, Apache

[6] https://iceberg.apache.org/spark-quickstart/
[7] https://trino.io/docs/current/connector/iceberg.html
[8] https://iceberg.apache.org/docs/1.4.3/flink-connector/
[9] https://iceberg.apache.org/docs/latest/hive/
[10] https://docs.databricks.com/en/delta/uniform.html
[11] https://docs.snowflake.com/en/user-guide/tables-iceberg
[12] https://docs.aws.amazon.com/redshift/latest/dg/querying-iceberg.html
[13] https://aws.amazon.com/s3/

Hive[14] introduced an SQL-like language, i.e. HiveQL, making it easier for analysts to query big data without extensive programming knowledge.

As data processing evolved, columnar file formats such as Apache Parquet[15], ORC[16] were introduced to improve storage efficiency and query performance. These formats enabled faster analytical processing by optimizing data compression and retrieval patterns. As data needs grew beyond traditional systems, the concept of data lakes emerged, offering a more flexible, scalable way to store vast amounts of raw and processed data. Modern data lakes are cloud-based storage systems designed to accommodate data in its original format, whether it is structured, semi-structured, or unstructured [10]. Unlike traditional file systems or databases, data lakes decouple storage from computing, making it possible to scale them independently based on demand. Building on the foundation of HDFS, data lakes take the benefits of distributed storage to the next level. While HDFS was designed for on-premises environments, data lakes leverage cloud-native architecture to provide infinite scalability, high availability, and cost-effective storage. Unlike files in directories in HDFS, Data is stored as objects in data lakes, which are durable and can be accessed from anywhere. However, data lakes suffered from a lack of transactional consistency, governance challenges, and poor query performance. This makes it difficult to maintain data integrity across multiple analytical workloads [9].

To address the limitations of traditional data lakes, modern open table formats were developed to introduce transactional consistency, schema evolution, and efficient metadata management and support the lakehouse architecture [8]. The open table architecture (see Figure 1) extends traditional data lake storage by implementing a metadata layer that tracks schema changes, partitions, and transactional states. By decoupling the table structure from the underlying storage, these formats offer enhanced data consistency, reliability, and easy management of large datasets in data lakes. These formats separate logical and physical data organization by abstracting the physical file structure. They track the table's state, including partitions, within a metadata layer at the file level.

Existing research has largely focused on comparing data warehouse and data lake architectures, examining how data lakes enable the storage and processing of both structured and unstructured data [1,10]. Additionally, studies have explored the emergence of the lakehouse architecture, which integrates the strengths of both approaches, offering enhanced performance and efficiency compared to querying raw files directly in data lakes [2].

Further research has extensively compared the most widely used lakehouse table formats, such as Delta Lake, Apache Hudi, and Apache Iceberg. These studies focus on performance benchmarking, transaction guarantees, schema evolution capabilities, and metadata handling efficiency [9]. Comparisons have analyzed read and write performance, assessing each format's ability to handle real-world workloads involving frequent updates, deletes, and large-scale batch processing.

[14] https://hive.apache.org/
[15] https://parquet.apache.org/
[16] https://orc.apache.org/

Data integration remains a fundamental challenge since the early days of database systems, particularly in merging data from diverse sources into a unified, coherent view. Research has explored the issues associated with cross-platform data integration, including schema heterogeneity, data quality, and query rewriting—especially in autonomous, distributed systems [13,14]. These challenges are further amplified in modern architectures, where data is spread across different cloud platforms, technologies, and formats.

Research has also explored various data integration techniques and identifies recurring challenges such as data heterogeneity, scalability, and performance across platforms [14]. These limitations often stem from a lack of standardized data representation and tight coupling between processing engines and storage formats.

This study tackles the challenges outlined above by proposing the use of the Apache Iceberg open table format to standardize and automate data integration across platforms. It specifically examines Iceberg's effectiveness in a multi-platform environment, focusing on its query performance, metadata synchronization, schema evolution, and scalability when used with Snowflake, Databricks, and data stored in Amazon S3. Unlike previous implementations that rely on copying terabytes of data across platforms—a process that is both tedious to maintain and prone to scalability and performance issues—this research aims to provide a comprehensive understanding of Iceberg's potential as a unified table format for seamless cross-platform data sharing, all while maintaining a single copy of the data.

3 Enabling Cross-Platform Data Interoperability

3.1 Introduction to Medallion Architecture in Modern Lakehouse Systems

As organizations increasingly adopt cloud-native architectures and leverage multiple data platforms, the need for structured, scalable data processing pipelines has grown substantially. One widely adopted paradigm that addresses this need is the Medallion Architecture, which provides a layered approach to data organization within lakehouse environments [15]. The architecture is composed of three layers:

- **Bronze Layer**: This foundational layer captures raw, unfiltered data from diverse sources, such as transactional systems, IoT devices, logs, and APIs, and stores it in a cost-effective, schema-flexible format within a data lake or lakehouse. This layer ensures that all incoming data is preserved in its original form for future processing or reprocessing.
- **Silver Layer**: The intermediate layer applies data transformation workflows including cleansing, deduplication, normalization, and the enforcement of referential integrity. These transformations are commonly performed using ETL or ELT tools and can occur within the same platform or be transferred to another engine optimized for data processing.

– **Gold Layer**: At the top of the medallion stack, this layer contains curated datasets tailored for business intelligence, advanced analytics, and machine learning workflows. Data in this layer is typically aggregated, joined, and enriched to support domain-specific use cases, such as real-time dashboards, forecasting models, and operational analytics.

To fully leverage each layer's capabilities, different tools and platforms optimized for specific tasks, such as cloud warehouses for BI, lakehouses for transformation, and separate engines for machine learning, are deployed in different layers based on the architecture and use case. This heterogeneous tooling introduces the need for robust data integration pipelines that ensure consistency and accessibility across layers and systems. Traditionally, several different integration approaches have been developed to enable cross-platform data integration [14]:

1. **ETL Pipelines**: These extract data from one system, transform it, and load it into another. While widely used, ETL workflows are often rigid, complex to scale, and prone to latency and data duplication.
2. **Data Virtualization**: Offers real-time access to data across platforms without physical movement. However, it introduces performance bottlenecks and lacks robust support for transactional guarantees, particularly in analytical workloads.
3. **API-Based Integration**: Exposes data or services through APIs to facilitate communication between platforms. While flexible, APIs require custom development, introduce security and versioning challenges, and may not be optimized for large-scale, high-throughput data transfers.
4. **Centralized Data Lakes**: Use cloud object storage to aggregate raw and processed data from multiple systems. Though flexible, data lakes typically lack standardized support for table-level operations, making schema evolution, version control, and concurrent access management difficult.

3.2 Standardizing Data Integration Across Platforms Using Apache Iceberg

To address the fragmentation caused by legacy methods, we will adopt the Iceberg format to enhance cross-platform integrations. Iceberg tables are an open-source, high-performance table format designed to enable reliable and efficient data lake analytics. Developed initially by Netflix and now governed by the Apache Software Foundation, Iceberg addresses many limitations of older table formats. The official definition describes Iceberg as "an open table format for huge analytic datasets" that brings SQL-like reliability and performance to the realm of distributed data lakes [8]. Iceberg's architecture is optimized for cloud environments, allowing efficient integration with modern data processing frameworks while eliminating issues related to file listing operations in object stores.

Layered Architecture of Iceberg Tables Iceberg employs a structured, layered architecture that enhances scalability, query performance, and data consistency. The architecture consists of three key layers as seen in Figure 2.

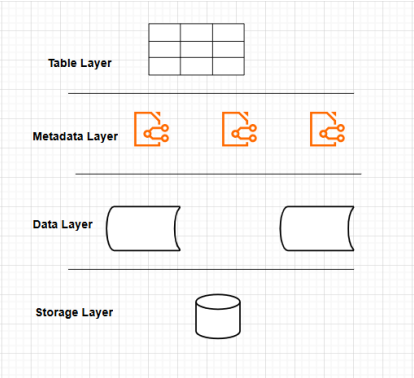

Fig. 1. Open Table Format Architecture

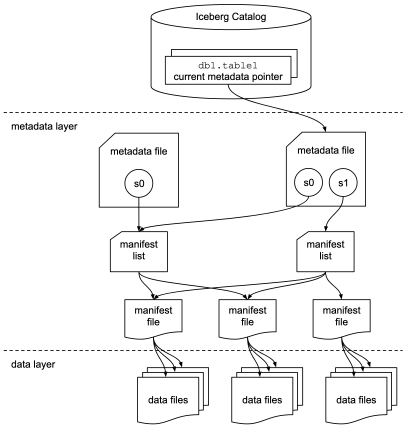

Fig. 2. Iceberg Table Structure.

The Metadata Layer in Iceberg plays a crucial role in managing table state through structured metadata files. The Snapshot Metadata File captures schema, partition details, and links to the Manifest List, enabling time travel and rollback. The Manifest List maps snapshots to Manifest Files, which store data file paths and column-level statistics for optimized query performance. The Data Files, stored in formats like Parquet, ORC, or Avro, ensure compatibility with various tools while supporting partition evolution without rewriting data. The Data Storage Layer supports diverse storage systems like Amazon S3 and Google Cloud Storage, preserving native formats without conversion overhead. The iceberg catalog integrates with execution engines like Spark, Flink, Trino, and Presto, leveraging metadata to optimize queries, ensure consistency, and prune unnecessary files. This structured approach enhances data lake visibility, allowing efficient data tracking via metadata pointers instead of costly file scanning. Iceberg's layered architecture improves performance, scalability, and interoperability across analytics platforms.

Why Choose Iceberg? Iceberg's architecture is purpose-built for efficient cross-platform integration. Its metadata layer sits atop the data lake, allowing query engines to quickly access the latest snapshot without scanning the full dataset. The pointer-based structure adds clarity and control, bringing true visibility to the data lake. As a result, Iceberg is gaining widespread adoption and native support across diverse processing engines. With extensive support from both commercial and open-source engines, and the recent announcement

that S3 [17] supports storing tables in Iceberg format, Iceberg ensures flexibility, interoperability, and a vendor-neutral approach for modern data management. By offering sophisticated metadata handling, schema evolution, and partition pruning, Iceberg enables precise data management, unlocking the true potential of data lakes for high-performance analytics. Its vendor-agnostic design ensures compatibility with a wide array of query engines and storage systems, empowering organizations to maintain flexibility and avoid vendor lock-in. This open and scalable architecture positions Iceberg as a key enabler for modern data engineering, seamlessly integrating with both existing data lakes and cutting-edge analytics platforms. For these reasons, companies are increasingly focusing their attention on the iceberg format, leading to its growing adoption.

4 Bi-Directional Data Integration Between Snowflake and Databricks

To address the challenges of data interoperability and consistency across platforms, in this section we will investigate the practical implementation of Apache Iceberg tables as a unified table format between Snowflake and Databricks, with data stored in Amazon S3. The experiments are designed to explore the research question regarding the challenges and limitations of using Iceberg to enable cross-platform data interoperability. This approach aims to simplify the complexities of manual pipeline management while ensuring compatibility and interoperability across diverse systems.

4.1 Architecture

The diagram 3 illustrates a generalized architecture for integrating Apache Iceberg tables across platforms (Snowflake and Databricks). The architecture comprises four main components. The source engine manages the Iceberg table, ingesting incoming data, typically representing the silver or gold layers in the Medallion Architecture, and writing it to a centralized data lake, such as Amazon S3. It also interacts with a catalog that maintains metadata files and pointers to the latest snapshot, ensuring transactional consistency. Both data and metadata are stored in the data lake, which serves as the central repository. The source engine requires write access to S3, typically controlled through IAM roles. A query engine, used for analytics, BI, or ML workloads, connects to the catalog to fetch the latest snapshot metadata and then reads the corresponding data files from the lake. Access to the catalog can be established via JDBC/ODBC, service principals, or native connectors, depending on the system. The query engine also requires appropriate read permissions to access the data in S3. This architecture ensures consistent, scalable, and efficient cross-platform data access using a single, authoritative Iceberg table. The study uses the publicly available

[17] https://aws.amazon.com/about-aws/whats-new/2024/12/amazon-s3-tables-apache-iceberg-tables-analytics-workloads/

Yellow Taxi Trip Records [18] from NYC TLC, covering the first quarter of 2024. With over 9.5 million rows and 19 columns, the dataset offers rich, real-world attributes like timestamps, geolocations, and payment info—ideal for data lake and analytical workload scenarios.

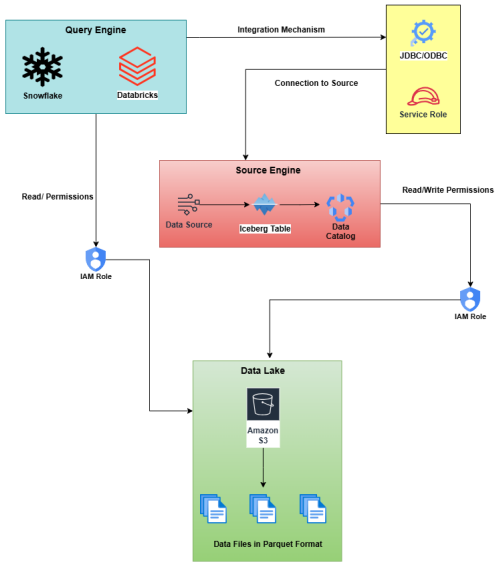

Fig. 3. Querying Iceberg Tables Across Platforms

5 Experimental Design

This section presents the methodology for evaluating Iceberg table integration across platforms, emphasizing real-world scenarios encountered by data teams. The experiment aimed to assess how efficiently Snowflake and Databricks can access and process Iceberg-managed tables while ensuring real-time synchronization and query optimization.

5.1 Integration Feasibility

Several detailed steps were followed [19], [20] to allow bidirectional integration of Iceberg tables between Snowflake and Databricks.

[18] https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page
[19] https://medium.com/snowflake/how-to-integrate-databricks-with-snowflake-managed-iceberg-tables-7a8895c2c724
[20] https://www.databricks.com/blog/read-unity-catalog-tables-in-snowflake

Snowflake Managed Iceberg Tables in Databricks
Step 1: Create an Amazon S3 bucket in the `eu-north-1` region.
Step 2: Configure IAM role with read/write permissions for Snowflake to access S3.
Step 3: Create external volume to manage data transfers between Snowflake and S3.
Step 4: Create Snowflake-managed Iceberg table using the external volume and taxi dataset.
Step 5: Apply security restrictions by creating a dedicated read-only user for Databricks.
Step 6: On Databricks side, configure IAM role with read-only permissions.
Step 7: Set up Databricks cluster with 10.4 LTS runtime and required libraries (Iceberg, AWS bundle, JDBC) and the Snowflake user.

Databricks Managed Iceberg Tables in Snowflake
Step 1: Enable UniForm on the Delta table in Databricks (during creation or via `ALTER TABLE`).
Step 2: Register Unity Catalog in Snowflake via Databricks service principal and OAuth credentials.
Step 3: Grant Snowflake access to S3 with IAM role, trust policies, and define external volume.
Step 4: Create Iceberg table in Snowflake referencing Unity Catalog metadata.
Step 5: Test integration by updating data in Databricks and verifying reflection in Snowflake (via auto-refresh or `ALTER ICEBERG TABLE REFRESH`).

5.2 Data Synchronization with Varying Dataset Sizes

To evaluate performance and scalability, datasets of 3 million, 6 million, and 9 million rows were loaded into Iceberg tables, and synchronization performance was observed. The test also included simultaneous access to multiple Iceberg tables to assess concurrent query execution.

5.3 Schema Evolution Across Platforms

Schema evolution refers to structural changes in a dataset, such as adding new columns, removing existing ones, or modifying column definitions (e.g., changing a column's data type or renaming it). It often represents a major challenge in data pipelines, as changes to the structure of data can disrupt workflows and lead to pipeline failures during data movement or integration. Schema evolution was tested by adding a new `RATING column` (VARCHAR) to an Iceberg table and updating it with default values. Additionally, type conversion from VARCHAR to INT, which required a workaround—creating a new column `RATING_INT`, applying `TRY_CAST`, and renaming the column after validation.

5.4 Latency and Performance Testing

Latency and performance form a critical part of this experiment, as they demonstrate how queries perform on both internally managed and externally managed Iceberg tables. The dataset used for this evaluation contains over 9.5 million

rows. To enhance the analysis, external tables were also included in the evaluation. External tables refer to datasets that reside in an external data lake, such as AWS S3 in this case, where the query engine directly accesses the data files without a dedicated metadata management layer. While this setup is similar to Iceberg tables in terms of data location, the key difference lies in the presence of Iceberg's metadata layer. Furthermore, an optimization was applied to the Iceberg table to enhance query performance. The data was reordered based on a TIMESTAMP column, which should allow the query engine to read data more efficiently by leveraging the sorted structure. Queries were designed in three categories:

- **Simple Queries** – Basic operations like COUNT(*) to measure row counts.
- **Medium Queries** – Arithmetic or conditional computations based on column grouping with conditional logic.
- **Complex Queries** – Aggregations, with full join.

6 Results and Discussion

In this section, we will discuss the results to evaluate the performance and efficiency of the integrations.

6.1 Integration Feasibility

The results showed that Iceberg table integrations between Snowflake and Databricks were efficient and effective. When accessing Snowflake-managed Iceberg tables from Databricks, Snowflake handles only lightweight metadata retrieval, while Databricks performs the actual computation. Testing showed metadata retrieval times within milliseconds. When Snowflake queried Databricks-managed Iceberg tables, metadata retrieval worked as expected, but additional setup was required to establish connectivity with Databricks' Iceberg catalog.

6.2 Data Synchronization with Varying Dataset Sizes

Scaling the dataset size had no impact on synchronization. Netflix, managing petabytes of data with Iceberg [21], further proves its scalability. The results were the same for multiple datasets as they could be accessed concurrently without issues, demonstrating the reliability of the integration. This capability ensures that both current and future datasets can be easily managed and queried across platforms without requiring significant reconfiguration or manual intervention. If this process were implemented without Iceberg tables, it would require significant overhead. For instance, syncing data between platforms would involve setting up multiple services on both ends to ensure data transfer. Additionally, scheduled jobs would be needed to refresh tables, taking considerable time and

[21] https://netflixtechblog.com/optimizing-data-warehouse-storage-7b94a48fdcbe

Table 1. Differences between column data types

Snowflake Datatypes	Databricks Datatypes
Number	Decimal
Float	Double
Varchar	String
Timestamp Ntz(6)	Timestamp

resources. Data quality checks would also be necessary at each step to verify the accuracy and consistency of the data. By contrast, Iceberg simplifies this process considerably. Since the data remains in the data lake and is accessed via metadata updates, there is no need for duplicative jobs or complex synchronization processes. As data resides solely in the data lake, any platform that supports Iceberg can query it, giving organizations the flexibility to use different platforms for varying use cases.

6.3 Schema Evolution Across Platforms

Schema evolution tests showed that Iceberg handles changes like adding, removing, or updating columns efficiently, with updates reflected instantly on querying platforms—thanks to its metadata layer as they read the schema definition directly from the latest metadata files. This eliminates the need for manual schema updates when the schema isn't explicitly defined. However, direct type conversion (e.g., VARCHAR to INT) isn't supported and requires workarounds. Despite this, features like column renaming and dropping worked seamlessly. We also observed differences in data type handling across platforms (see Table 1), and noted that Iceberg only supports microsecond precision for TIME and TIMESTAMP types in both v1 and v2. Overall, Iceberg's native support for schema evolution reduces pipeline breakage and simplifies data operations by keeping platforms in sync automatically. This capability significantly simplifies data operations, allowing teams to focus on their analytical and operational goals without worrying about the complexities of the manual schema management.

6.4 Latency and Performance.

Figure 4 shows Databricks-managed Iceberg tables were tested on both Databricks and Snowflake, showing notable performance differences. In Databricks, simple queries showed minimal difference between standard Iceberg and external tables, while medium queries executed 2 seconds faster than external tables and 4 seconds faster than standard Iceberg tables after optimization. For complex queries, external tables outperformed others. In Snowflake, Iceberg tables outperformed external tables by 4.5 times for medium queries and 9 times for hard queries. Optimized Iceberg tables showed a slight performance improvement over standard tables, highlighting the benefits of optimizations.

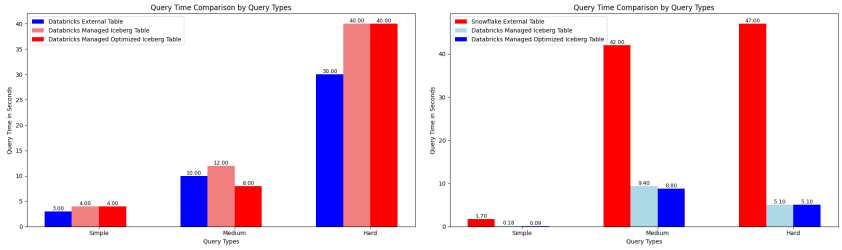

Fig. 4. Databricks managed Iceberg Table in Databricks(left), Snowflake(right)

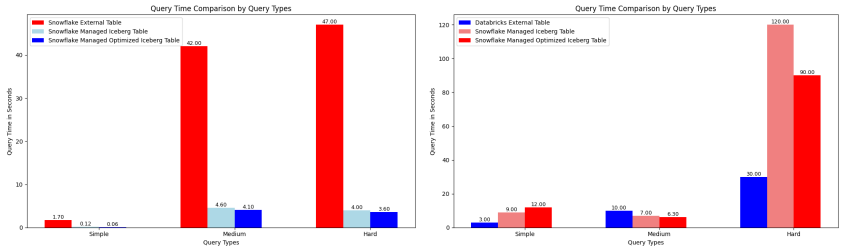

Fig. 5. Snowflake managed Iceberg Table in Snowflake(left), Databricks(right)

Next Snowflake-managed Iceberg tables were tested on both Databricks and Snowflake, showing notable performance differences. In Snowflake, Iceberg tables outperformed external tables, being 9 times faster for medium queries and 12 times faster for hard queries, likely due to Snowflake's use of its own catalog. In Databricks, external tables generally performed better, except for medium-sized queries as seen in Figure 5. Queries leveraging Iceberg features and scanning fewer partitions were notably faster. Performance in Databricks varied based on query type and operations.

This reveals several interesting observations. Iceberg tables outperformed external tables for queries that did not require a full table scan, such as conditional calculations. This was consistent for both self- managed and external-managed Iceberg tables in Databricks. However, in cases of full table scans, external tables demonstrated faster performance. In Snowflake, Iceberg tables outperformed external tables across all query types, particularly when managed by Snowflake. Iceberg tables also offer the advantage of automated synchronization, ensuring near real-time data access and consistency, unlike external tables that require periodic refreshes, leading to delays, extra costs, and the risk of querying outdated data.

6.5 Challenges and Limitations

Integrating external Iceberg tables in Databricks with Unity Catalog [22] enabled presents challenges, as Unity Catalog uses Delta Lake as the primary metadata format, limiting access to external Iceberg tables. While Databricks can read Iceberg tables from external platforms when Unity Catalog is disabled, compatibility issues remain. Another important consideration is that as data size increases, it's crucial to implement policies for expiring outdated metadata and data files. This ensures that the data doesn't grow unnecessarily, helping to keep storage usage in check according to specific requirements. While Iceberg tables offer a promising solution for automating data synchronization across platforms, their query performance for full table scans on certain platforms still falls short and requires further optimization.

7 Conclusions

In this paper, we have explored the evolution of data management, the challenges posed by different offerings, the emergence of the lakehouse architecture, the advent of modern open table formats, and the complexities that arise when using multiple table formats across different platforms. The challenges organizations face in ensuring interoperability among these formats underscore the pressing need for a unified table format. This need becomes even more critical in a data-driven world where data sharing and efficient analytics are pivotal for success.

Apache Iceberg has emerged as a strong solution for modern data challenges, offering performance, scalability, and cross-platform consistency. Its growing adoption enables seamless interoperability between platforms like Snowflake and Databricks, reducing latency, improving query efficiency, and simplifying data management. Our comparative study showed that Iceberg enables near real-time synchronization, handles schema evolution effectively, and minimizes data duplication by maintaining a single data copy. While query performance was strong—especially when avoiding full table scans—some integration aspects still need refinement. Adopting Iceberg can streamline workflows, reduce operational burdens, and unlock the full potential of data assets, empowering organizations to make faster, more effective data-driven decisions.

Future research can explore several promising avenues to extend the findings of this paper. A key focus could be evaluating Databricks functionality when Unity Catalog begins supporting external Iceberg tables, with an emphasis on performance analysis. Additionally, expanding Iceberg's integration with platforms beyond Snowflake and Databricks could highlight its potential as a truly universal table format. Investigating the adoption of advanced catalog systems, such as Apache Polaris [23], for managing metadata instead of relying solely on storage engines, could enhance management and query performance. Furthermore, advance optimization techniques could be tested for improving query

[22] https://www.databricks.com/product/unity-catalog
[23] https://polaris.apache.org/

performance. Iceberg's time travel feature presents an exciting opportunity for assessing its applications in historical data analysis and auditing. Exploring these areas will provide valuable insights into Iceberg's role in modernizing data ecosystems and driving broader adoption across diverse platforms.

References

1. F.Ravat and Y. Zhao, "Data lakes: Trends and perspectives", International Conference on Database and Expert Systems Applications, pp. 304-313, 2019.
2. Armbrust, Michael, et al. "Lakehouse: a new generation of open platforms that unify data warehousing and advanced analytics". CIDR. Vol. 8. 2021.
3. Mone, A. (2020). "The Metadata Challenge in Big Data Systems". Journal of Data Management, 23(4), 15-27.
4. Codd, E. F. (1970). A relational model of data for large shared data banks. Communications of the ACM, 13(6), 377-387.
5. Jain, P., Kraft, P., Power, C., Das, T., Stoica, I., Zaharia, M. (2023). Analyzing and Comparing Lakehouse Storage Systems. In CIDR.
6. F. Nargesian, E. Zhu, R. J. Miller, K. Q. Pu and P. C. Arocena, "Data lake management: challenges and opportunities", PVLDB, vol. 12, no. 12, 2019.
7. V. Christophides, V. Efthymiou and K. Stefanidis, "Entity Resolution in the Web of Data", Synthesis Lectures on the Semantic Web: Theory and Technology, Morgan Claypool Publishers 2015, ISBN 978-3-031-79467-4.
8. Errami, Soukaina Ait, et al. "Spatial big data architecture: from data warehouses and data lakes to the Lakehouse". Journal of Parallel and Distributed Computing 176 (2023): 70-79.
9. Armbrust et al. Delta Lake: High-Performance ACID Table Storage over Cloud Object Stores. PVLDB, 13(12): 3411-3424, 2020.
10. J. Singh, G. Singh and B. S. Bhati, "The Implication of Data Lake in Enterprises: A Deeper Analytics", ICACCS 2022.
11. T. Brasileiro Araújo, V. Efthymiou, V. Christophides, E. Pitoura and K. Stefanidis: TREATS: Fairness-aware entity resolution over streaming data. Inf. Syst. 129: 102506 (2025).
12. https://www.virtana.com/press-release/virtana-research-finds-more-than-80-of-enterprises-have-a-multi-cloud-strategy-and-78-are-using-more-than-three-public-clouds/
13. Doan, AnHai, Alon Halevy, and Zachary Ives. Principles of data integration. Elsevier, 2012.
14. Bagam, Naveen. "Data Integration Across Platforms: A Comprehensive Analysis of Techniques, Challenges, and Future Directions." International Journal of Intelligent Systems and Applications in Engineering 12 (2024): 902-919.
15. Bhatt, S., Sekar, D., Data Warehousing Modeling Techniques and Their Implementation on the Databricks Lakehouse Platform, (2022). https://www.databricks.com/blog/2022/06/24/data-warehousing-modeling-techniques-and-their-implementation-on-the-databricks-lakehouse-platform.html.