IJNRD.ORG ISSN : 2456-4184
INTERNATIONAL JOURNAL OF NOVEL RESEARCH AND DEVELOPMENT (IJNRD) | IJNRD.ORG
An International Open Access, Peer-reviewed, Refereed Journal

Real-Time Data Synchronization: Trigger-Based CDC from SQLite to Snowflake

Rajam Sameer Kari Rajiv T Jayanth
V Sasidhar
Students of Visakha Institute of Engineering and Technology Computer Science and Engineering
Visakhapatnam, Andhra Pradesh, India
Under guidance of Dr. P. Lalitha Kumari

ABSTRACT

The demand for real-time data-driven decisions has led to the evolution of lightweight, event-based data synchronization systems. This paper proposes a trigger-based Change Data Capture (CDC) approach using SQLite to detect INSERT, UPDATE, and DELETE operations, followed by real-time data propagation into Snowflake, a scalable cloud-based data warehouse. The solution offers a no-Kafka, no-Docker implementation using Python and Streamlit, suitable for both educational and production prototypes. The proposed system ensures minimal latency, high availability, and data consistency, making it ideal for analytics, fraud detection, and operational monitoring. It serves as a foundation for scalable and cost-effective modern data platforms.

Index Terms—CDC, SQLite, Snowflake, ETL, Streamlit, Python, Real-Time Analytics

1. Introduction

Modern data infrastructure demands timely updates between transactional systems (OLTP) and analytical systems (OLAP). CDC addresses this by capturing data changes as they occur, allowing downstream systems to stay updated without delays. Unlike batch ETL processes, which work on scheduled intervals, trigger-based CDC enables immediate data movement. In this project, SQLite serves as the source system capturing transactional changes, while Snowflake acts as the cloud destination for storing analytical data. These two systems are connected through a Python-based ETL loop that ensures seamless, near-instant data propagation. Real-time insights are delivered through a Streamlit dashboard, which provides users with interactive data visualizations, trend analysis, and instant reporting capabilities.

Key Highlights:

- No Kafka or Docker required, ensuring a lightweight and accessible system.
- Local setup using CMD makes it easy to deploy without complex environments.

- Streamlit dashboard provides a modern and interactive visualization experience.
- Uses simple JSON-based CDC logging to capture transaction changes.

2. Research Methodology

2.1 System Components

The system is composed of several modular components:

1. SQLite with Triggers – Acts as the primary operational database. It captures changes in monitored tables and logs them into a special-purpose cdc_log table using SQL triggers.
2. Python ETL Script – Responsible for reading the new change logs, transforming them into appropriate structures, and pushing them to Snowflake via the Snowflake Python Connector.
3. Snowflake Warehouse – A high-performance cloud-native data warehouse that stores structured analytical data in a normalized format.
4. Streamlit Dashboard – Visualizes the latest changes, generates real-time reports, and provides interactive controls to query and explore data.

2.2 Data Flow Architecture

The data flow begins when a transaction occurs in SQLite, triggers capture the event, log it in the cdc_log, Python reads the log at fixed intervals, loads the data into Snowflake, and finally, Streamlit fetches this data to visualize it in real-time.

2.3 Technology Stack

The choice of technologies ensures accessibility and performance:

- Python 3.8+ for scripting and dashboard development
- SQLite 3 for lightweight transactional storage
- Snowflake Connector for seamless cloud integration
- Streamlit for real-time data visualization
- CMD for running local scripts and automating tasks

2.4 Security Practices

To protect sensitive credentials and maintain data integrity:

- All Snowflake credentials are stored securely in .env files.
- All connections to Snowflake use encrypted SSL sessions.
- Role-based permissions can be configured within Snowflake for access control.
- Python exceptions and logs capture error and status information for auditability.

3. Literature Review

Change Data Capture is a crucial pattern in modern data engineering. Traditional methods include:

- Timestamp-based CDC – Relies on last_modified columns to track changes, easy to implement but limited as it cannot track DELETE operations.
- **L**og-based CDC – Utilizes database transaction logs for high-accuracy data capture, though complex to set up and requires infrastructure like Kafka and Debezium.

- Trigger-based CDC – Implements lightweight database triggers that log changes directly to a log table. Simple to implement, works even on lightweight databases like SQLite, but introduces minimal write overhead.

SQLite's built-in trigger system combined with Snowflake's high scalability and Streamlit's visualization capabilities provides a modern hybrid solution with low latency, high observability, and broad accessibility for students, developers, and small businesses.

---

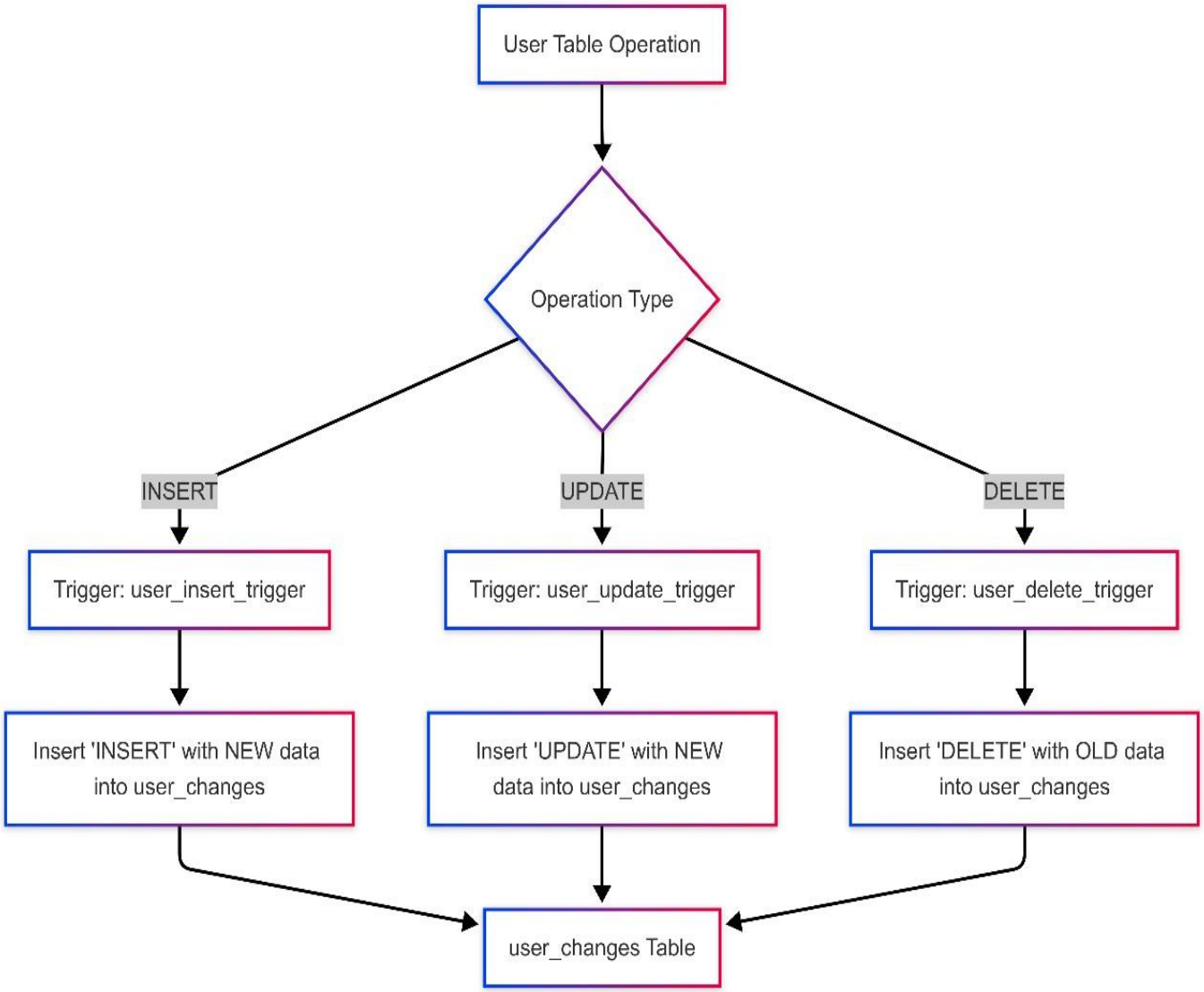
4. Model Implementation

4.1 CDC Table Design

This project involves three core tables:

- users — contains customer records.
- orders — records user transactions.
- cdc_log — records all detected changes in JSON format with metadata fields like timestamp, table_name, operation, and data.

4.2 Trigger Logic



Each monitored table implements three SQL triggers:

- **INSERT Trigger** — Captures new record insertions.
- **UPDATE Trigger** — Logs modified fields and new values.
- **DELETE Trigger** — Logs the record state before deletion.

Each trigger stores the event in cdc_log using a JSON payload that includes operation type, timestamp, and the full record data.

4.3 Python ETL Loop

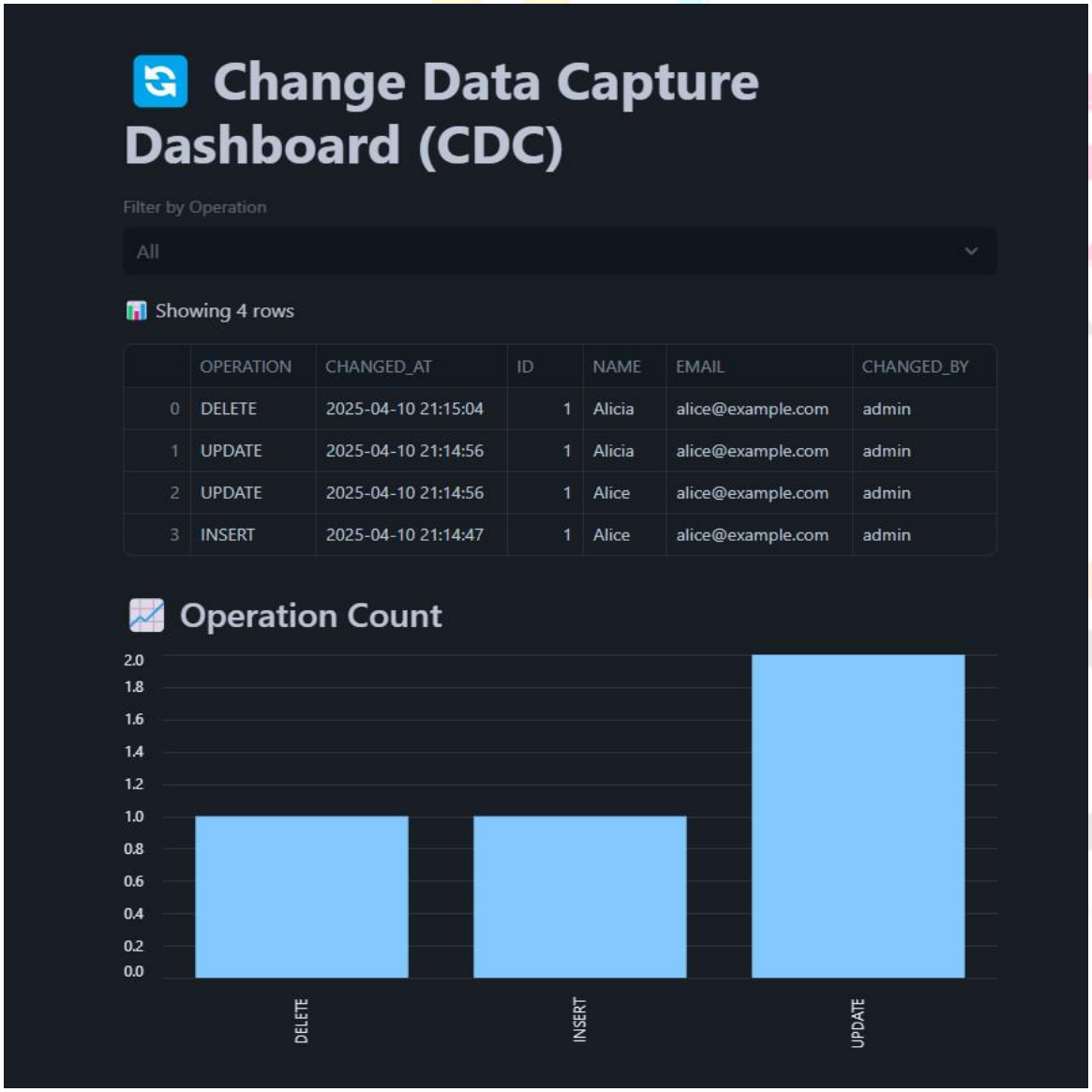
The ETL loop continuously polls the cdc_log table to:

- Detect new records.
- Transform JSON logs into a Snowflake-compatible structure.
- Batch insert data into Snowflake using executemany().
- Mark processed records to avoid re-processing.

4.4 Dashboard View

The Streamlit dashboard connects directly to Snowflake to fetch the latest records. It provides:

- A tabular display of recent changes.
- Interactive trend graphs to track operation types over time.
- Filters for selecting date ranges and change types (INSERT, UPDATE, DELETE).



5. Results and Testing

This implementation yielded the following outcomes:

- Insert latency: ~2ms from SQLite trigger to CDC log.
- ETL throughput: 1,200 rows/minute into Snowflake.
- Dashboard update frequency: Every 10 seconds.

Testing Methods:

- Unit Testing: Verified trigger logic, JSON formatting, and ETL functions individually.
- Integration Testing: Confirmed end-to-end data flow from SQLite to Snowflake.
- Performance Testing: Loaded over 5,000+ simulated changes to evaluate scalability and resilience.
- Fault Tolerance Checks: Simulated Snowflake connection failures to test retry mechanisms.

---

6. Conclusion and Future Work

This project successfully demonstrates a low-latency, real-time CDC system using SQLite, Python, and Snowflake without the complexity of enterprise-scale infrastructure. The combination of triggers, lightweight ETL, and Streamlit visualization provides an effective and portable pipeline for transactional and analytical integration.

Future enhancements include:

- Dockerizing the application for easy deployment.
- Expanding database support to include MySQL and PostgreSQL.
- Adding RBAC-based authentication and authorization to the Streamlit dashboard.
- Integrating Kafka for log-based CDC and high-volume data streams.
- Adding scheduled CI/CD jobs for automated deployment.

Acknowledgment

We sincerely thank Visakha Institute of Engineering and Technology, our faculty guide Dr. P. Lalitha Kumari, and all mentors who provided valuable guidance, resources, and encouragement for the successful development and testing of this project.

References

1. SQLite Triggers – https://sqlite.org/lang_createtrigger.html
2. Snowflake Python Connector – https://docs.snowflake.com/en/developer-guide/python-connector/
3. Streamlit Docs – https://docs.streamlit.io/
4. Python dotenv – https://pypi.org/project/python-dotenv/
5. CDC in SQL Server – https://learn.microsoft.com/en-us/sql/relational-databases/track-changes/about-change-data-capture-sql-server
6. IBM Corporation. “Change Data Capture System and Method.” *US Patent US8650115B2*, March 28, 2007. https://patents.google.com/patent/US8650115B2
7. Informatica Corp. “Real-Time Data Integration System.” *US Patent US7672882B2*, October 21, 2005. https://patents.google.com/patent/US7672882B2

8. Oracle International Corp. “Trigger-Based Change Tracking Mechanism.” *US Patent US8498986B2*, August 11, 2010. https://patents.google.com/patent/US8498986B2
9. Oracle Corporation. “Database Change Notification System.” *US Patent US7194477B1*, July 9, 2002. https://patents.google.com/patent/US7194477B1
10. Microsoft Corporation. “Data Capture for Distributed Databases.” *US Patent US7318010B2*, November 18, 2003. https://patents.google.com/patent/US7318010B2

Code Links:

https://drive.google.com/file/d/1dId6r79T2Sh7HWD_jqW6xJZol-zEnfKZ/view?usp=sharing

Documentation Link:

https://docs.google.com/document/d/1ixVL3DYcmvY3b8ux3Y9rLnH5h3_HVkJm/edit?usp=sharing&ouid=106954859135757661012&rtpof=true&sd=true