International Journal of Core Engineering & Management
Volume-6, Issue-11, 2021 ISSN No: 2348-9510

COMPREHENSIVE GUIDE ON CHANGE DATA CAPTURE USING SNOWFLAKE AND AWS S3 WITH RETAIL TRAFFIC AND CUSTOMER INSIGHT DATA

Ravi Kiran Koppichetti
koppichettiravikiran@gmail.com

Abstract

Change Data Capture (abbreviated as CDC) is a crucial process in Data Management that enables businesses to track and synchronize real-time data changes. It enables advanced data management, timely insights, and operational efficiencies. CDC is a way of tracking and collecting only the changes made to data in the source. So, instead of capturing all the data from the source table and refreshing the entire target table, it enables us to capture the changes that occurred since the last refresh. CDC is beneficial if data needs to be transferred from source to target tables/ databases in real-time or near-real time. It helps synchronize databases, improve data analytics, and keep the data warehouse up-to-date. It enables providing the most current data to downstream systems without reprocessing everything [4].

This comprehensive guide offers a detailed implementation of Change Data Capture using Snowflake and AWS S3, using retail traffic analysis data. This paper describes how to implement CDC using Snowflake data objects such as staging, target schema layers, tables, snowpipe, stream, and tasks. This resource aims to help data engineers, data architects, and IT professionals implement effective CDC strategies to remain competitive in the evolving retail landscape.

Keywords: Snowflake, AWS S3, CDC, Change Data Capture, Retail Traffic Analysis, Customer Insights, Real time data processing, Cloud, Retail Analytics, Continuous Data Flow, Incremental Data Updates, CDC Strategy, Transform and Load

I. INTRODUCTION

Most enterprise applications store their operational data in a relational database. These systems efficiently manage large volumes of transactions, including data insertion, deletion, and updates. Since most of these applications handle high transaction volumes and ensure data integrity (ACID properties: Atomicity, Consistency, Isolation, Durability), developers build analytical processing systems to support easy and efficient data exploration, reporting, and analytical querying over large datasets. Moving data from the relational database to an analytical database helps organizations maintain operational efficiency, improve analytical capabilities, and help businesses make fast data-driven decisions without affecting the enterprise application [5].

Many modern applications typically build on microservices architecture, decoupling source systems (publishers) from analytical systems (consumers) using an asynchronous communication pattern [6]. This communication pattern is impossible in most legacy application databases, so loading the entire dataset from the source system(s) to the analytical system(s) is necessary to develop the required business metrics and dashboards. This approach increases overall batch execution time and load on source infrastructure.

Change Data Capture process designed to transfer data from a legacy source system to an analytical system can prevent the need to move the entire dataset each time the source data changes. This approach allows for delivering near-real-time data to analytical systems or downstream applications, thus helping businesses make data-driven decisions quickly.

II. ASSUMPTIONS

- Retail customer traffic data is captured by sensors in the store and stored in the application system's Customer Tracker Database.
- The Customer Tracker Database application system is a legacy system without built-in Pub/Sub functionality.
- The user has created an AWS S3 bucket/folder to store Customer Tracker Data Files.
- A data pipeline is built between the Customer Tracker Database and already AWS S3 using API
- The designated AWS S3 bucket receives the data from the Customer Tracker Database.
- The file format of Customer Tracker data files loaded into AWS S3 is CSV (Comma-separated values)
- The AWS S3 bucket designated to load Retail customer traffic data files is URL='s3://load/files/'.
- The user has built a Snowflake storage integration between Snowflake and the designated AWS S3 bucket.
- The user will be using the Snowflake default user 'sysadmin.'
- The user must load Customer Tracker data into the 'db_CustomerTrack' database.
- The user has built the EDW, Stage, and DM schemas in the 'db_CustomerTrack' database.
- Customer Tracker data loaded in AWS S3 consists of 5 columns (StoreID, data_code, enters, exits, traffic_startTime).
- The user needs the most updated data from the customer tracker database every minute.
- The data pipeline between the customer tracker database and the designated AWS S3 bucket will load a new data file at a one-minute interval (near-real time).
- The customer tracker database has new records inserted in real time and can not be updated.
- Each new data file created for loading into the AWS S3 bucket includes data from the previous 5 minutes.
- The change data capture (CDC), which refreshes the target table in the EDW Schema, is scheduled to run at a one-minute interval.
- The user does not want to use third-party applications to enable pub-sub functionality.
- The CDC is enabled after a full data transfer from the customer tracker database to the target table in the EDW schema of the 'db_CustomerTrack' database.

III. INTRODUCTION TO SNOWFLAKE

Snowflake is a fully managed cloud data platform explicitly designed to handle the complex demands of modern data warehousing, data lake, and data engineering applications. Developed with a cloud-native architecture, Snowflake provides a scalable, performant, and cost-effective solution for storing and analysing large-scale datasets across various cloud environments, including AWS, Google Cloud Platform, and Microsoft Azure [1].

Unlike traditional on-premises data warehouses, Snowflake utilizes a unique multi-cluster, shared-data architecture that separates compute and storage layers. This separation allows for elastic scaling of compute resources independently of data storage, which enables cost savings and performance optimization based on workload demands [2]. The platform natively supports structured and semi-structured data formats, such as JSON, Avro, and Parquet, allowing seamless integration and analysis of diverse data types within a single environment [3].

One of Snowflake's key features is its data sharing and collaboration capabilities, which allow for secure, near-instantaneous data sharing across different accounts without creating redundant copies of data. This functionality is valuable for organizations sharing data across business units, partners, or customers in real-time, enhancing decision-making and operational agility [1].

Snowflake automates many tasks, such as indexing, partitioning, and performance tuning. These automations allow organizations to concentrate on generating insights rather than managing database infrastructure, leading to more efficient use of resources and expertise. Snowflake also offers built-in security features and holds compliance certifications like SOC 2, ISO 27001, and HIPAA, making it an excellent choice for industries with strict data governance requirements.[3].

Due to its flexibility, ease of use, and scalability, Snowflake has garnered broad adoption across industries for use cases spanning data warehousing, data lakes, machine learning, data engineering, and real-time analytics. It represents a significant evolution in data platform design, aligning with the needs of data-driven organizations seeking cloud-native solutions for the modern data ecosystem [2].

IV. SNOWFLAKE ARCHITECTURE

Snowflake’s architecture is unique because it was designed specifically for the cloud. It separates the compute, and storage layers and allows highly flexible, scalable, and cost-effective data management. Traditional data warehouses typically combine computing and storage, which can lead to resource constraints and inefficient scaling. Snowflake, however, employs a multi-cluster, shared-data architecture where storage and computing operate independently, allowing each to scale independently and be optimized for different types of workloads [1].

At the core of Snowflake’s architecture are three main layers. Fig 1. visualizes the three main layers in Snowflake.

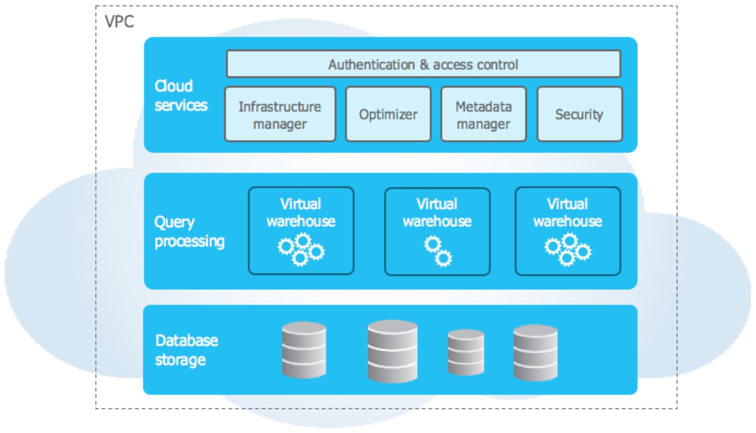


Fig.1. Snowflake Core Architecture. [3]

1. **Database Storage Layer:** Snowflake stores all data in a centralized storage layer decoupled from the compute resources. Data is automatically managed, compressed, and organized in a proprietary format, which supports structured, semi-structured, and unstructured data formats such as JSON, Avro, and Parquet. This centralized storage layer allows all users to work from a single source of truth without creating redundant data copies.

2. **Virtual Warehouse (Compute) Layer:** Snowflake refers to its compute resources as "virtual warehouses," which users can scale up, down, or pause based on computing demands. These warehouses consist of independent clusters of compute nodes. Each virtual warehouse can operate concurrently on the same data, enabling high concurrency for multiple workloads without resource contention or performance degradation. This design is especially advantageous for businesses with varying data processing needs throughout the day [3].

3. **Cloud Services Layer:** The cloud services layer manages functions such as metadata storage, authentication, access control, and query optimization. Snowflake automatically handles metadata management, performance tuning, and other operational tasks, allowing users to focus on data analysis rather than infrastructure management. This layer also includes Snowflake's innovative features like time travel, zero-copy cloning, and secure data sharing, making Snowflake a powerful choice for collaborative and multi-tenant data ecosystems [2,3]

In addition to the three main layers, Snowflake also provides the following features.

1. **Data Sharing and Collaboration**: Snowflake's architecture also supports data sharing and secure data exchange across organizations through its "Data Marketplace" and "Data Sharing" features. This capability allows users to share live data with internal or external stakeholders in real time without having to copy or move the data, significantly enhancing collaboration while maintaining data governance [2,3].

2. **Concurrency and Performance Management:** Unlike traditional architectures, Snowflake's design inherently supports high concurrency. Multiple virtual warehouses can work on the same dataset without interference, allowing different teams to perform operations simultaneously without affecting performance [2].

Snowflake's architecture allows organizations to use a single platform for multiple data-related tasks, including data warehousing, data lake, data engineering, and advanced analytics, thereby reducing data silos and creating a unified, scalable data environment that adapts to business demands.

V. SPECIFY DATABASE, SCHEMA AND ROLE FOR THE USER SESSION

```
use role sysadmin;
use database db_CustomerTrack;
use schema stage;
```

5.1 Snowflake File Format

A Snowflake file format is a defined database object containing essential information about a data file, including its format type (CSV or JSON), configuration settings, and compression method.

Snowflake file formats simplify data loading and unloading between Snowflake tables and external files. When loading data into a table, you can specify the file format, which informs Snowflake how to read and accurately import the data.

Here are the supported Snowflake file formats:

- CSV (Comma-separated values): A commonly used format for structured data imports.
- JSON (JavaScript Object Notation): A flexible, lightweight format for semi-structured data.
- Avro: A binary format ideal for efficiently handling large datasets.
- ORC (Optimized Row Columnar): A columnar format optimized for analytical workloads.
- Parquet: A columnar format similar to ORC but with broader compatibility across systems.
- XML (Extensible Markup Language): A text-based format frequently used for structured data storage.

These file formats allow Snowflake to interpret and process diverse data types effectively.

5.2 Create a File Format for Customer Tracker data

```
CREATE OR REPLACE FILE FORMAT CSV_ST_LOAD
TYPE = CSV
COMMENT = 'CSV File Format for Customer Tracker API Data'
COMPRESSION = AUTO
RECORD_DELIMITER = '\\n'
FIELD_DELIMITER = ','
SKIP_HEADER = 0
SKIP_BLANK_LINES = TRUE
TIMESTAMP_FORMAT = AUTO
TIMESTAMP_FORMAT = AUTO
DATE_FORMAT = AUTO
TIME_FORMAT = AUTO
TIMESTAMP_FORMAT = AUTO
ESCAPE = NONE
ESCAPE_UNENCLOSED_FIELD = NONE
TRIM_SPACE = TRUE
NULL_IF = ('\\N', 'NULL', 'NUL', '')
ERROR_ON_COLUMN_COUNT_MISMATCH = FALSE
REPLACE_INVALID_CHARACTERS = FALSE
VALIDATE_UTF8 = FALSE
EMPTY_FIELD_AS_NULL = TRUE
SKIP_BYTE_ORDER_MARK = TRUE
ENCODING = UTF8 ;
```

5.3 Snowflake Stage

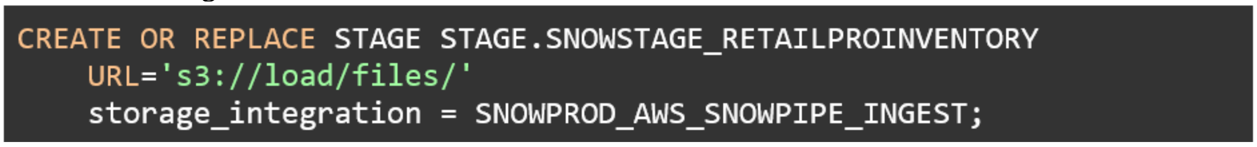
Snowflake stages are storage locations within Snowflake used to temporarily store data files before loading them into Snowflake tables or exporting them elsewhere. Each stage acts as a reference to specific data files, enabling Snowflake to load or unload these files without duplicating or moving them directly.

Stages offer a solution to the challenges of directly loading files into tables by streamlining data transfers between Snowflake and other platforms. Common uses include:

- Loading external data into Snowflake tables for analysis or reporting.
- Exporting data from Snowflake tables to external storage for backup or sharing.
- Importing internal data into Snowflake for processing and transformation.
- Exporting data to internal locations for temporary storage or further staging.

Snowflake stages can work with a range of sources, including local files and cloud storage services like AWS S3, Google Cloud Storage, and Azure Containers, making them a flexible tool for efficient data transfer.

5.4 Create a Stage for Customer Tracker data

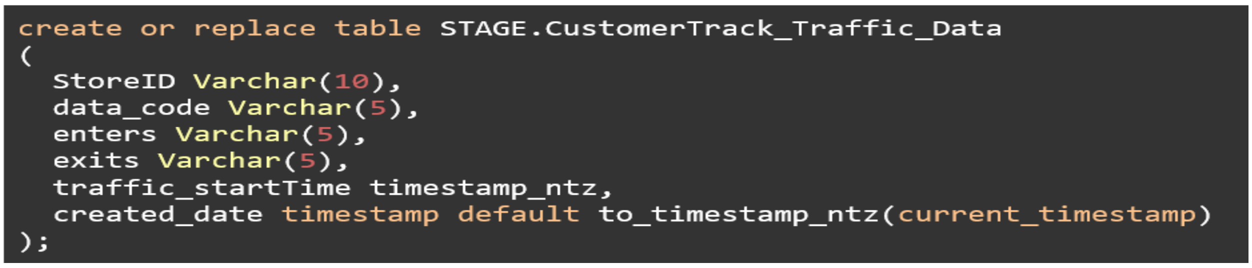


```
CREATE OR REPLACE STAGE STAGE.SNOWSTAGE_RETAILPROINVENTORY
    URL='s3://load/files/'
    storage_integration = SNOWPROD_AWS_SNOWPIPE_INGEST;
```

5.5 Snowflake Database Objects

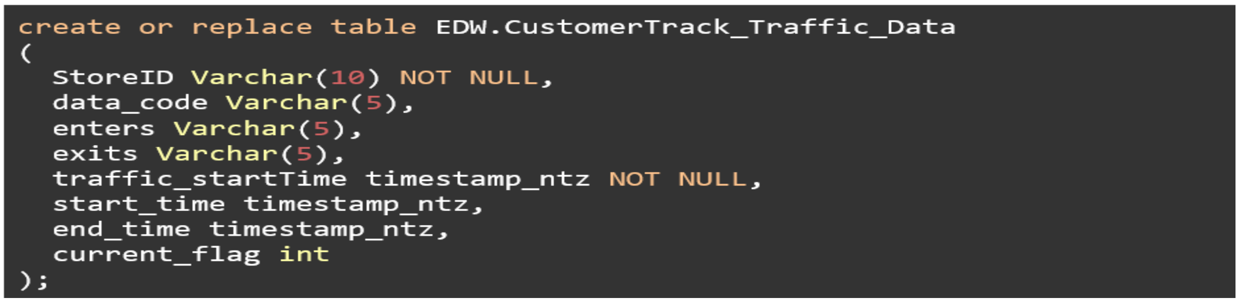
For this example, let's create a data staging table in 'Stage' Schema and a target table in 'EDW' Schema

5.5.1 Create a table for Customer Tracker data in STAGE Schema



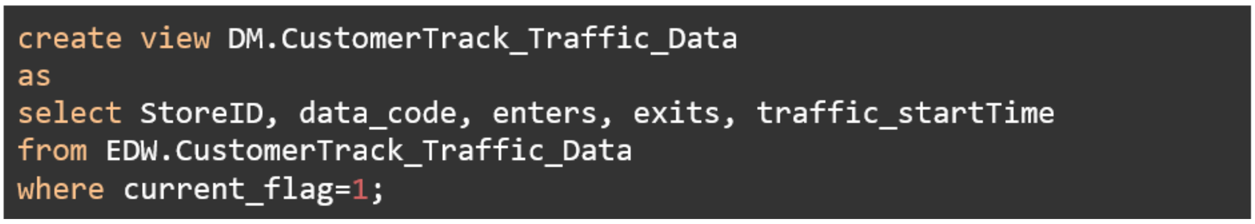
```
create or replace table STAGE.CustomerTrack_Traffic_Data
(
  StoreID Varchar(10),
  data_code Varchar(5),
  enters Varchar(5),
  exits Varchar(5),
  traffic_startTime timestamp_ntz,
  created_date timestamp default to_timestamp_ntz(current_timestamp)
);
```

5.5.2 Create a table for Customer Tracker data in EDW Schema



```
create or replace table EDW.CustomerTrack_Traffic_Data
(
  StoreID Varchar(10) NOT NULL,
  data_code Varchar(5),
  enters Varchar(5),
  exits Varchar(5),
  traffic_startTime timestamp_ntz NOT NULL,
  start_time timestamp_ntz,
  end_time timestamp_ntz,
  current_flag int
);
```

5.5.3 Create a view for Customer Tracker data in DM Schema

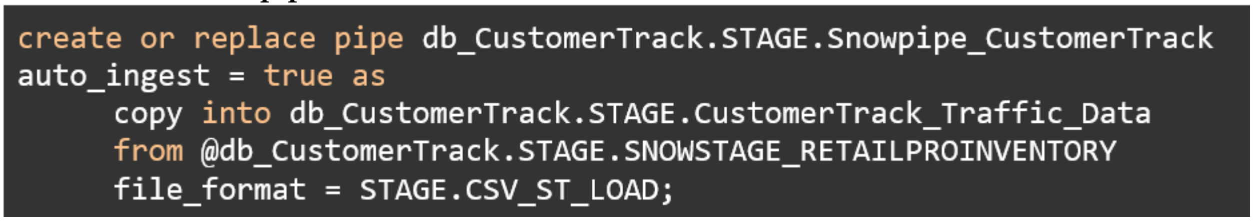


```
create view DM.CustomerTrack_Traffic_Data
as
select StoreID, data_code, enters, exits, traffic_startTime
from EDW.CustomerTrack_Traffic_Data
where current_flag=1;
```

5.6 Snowflake SnowPipe

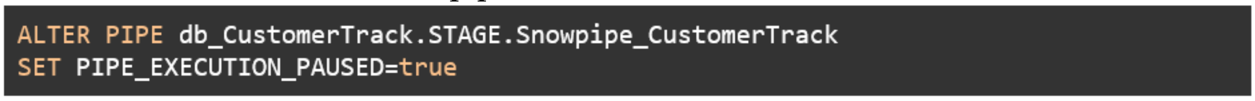
Snowpipe is a serverless data ingestion service from Snowflake that enables fast, automated data loading into Snowflake tables. It continuously ingests data from files when available in a specified stage, supporting micro-batch loading and providing users near-real-time access to new data. This approach eliminates the need for manually running scheduled `COPY` commands for extensive batch loading, enabling a more efficient and timely data pipeline.

5.6.1 Create a Snowpipe for Customer Tracker data in STAGE Schema



```
create or replace pipe db_CustomerTrack.STAGE.Snowpipe_CustomerTrack
auto_ingest = true as
     copy into db_CustomerTrack.STAGE.CustomerTrack_Traffic_Data
     from @db_CustomerTrack.STAGE.SNOWSTAGE_RETAILPROINVENTORY
     file_format = STAGE.CSV_ST_LOAD;
```

5.6.2 Pause the execution of Snowpipe created for Customer Tracker Data



```
ALTER PIPE db_CustomerTrack.STAGE.Snowpipe_CustomerTrack
SET PIPE_EXECUTION_PAUSED=true
```

5.7 Snowflake Stream

Snowflake streams are objects that monitor and track all DML (Data Manipulation Language) changes—such as inserts, updates, and deletes—on a specified source table. When we create a stream, it adds three metadata columns to capture these changes: “METADATA$ACTION”, “METADATA$ISUPDATE”, and “METADATA$ROW_ID”. These columns allow the stream to log modification information without duplicating the entire table’s data.

A stream essentially functions as a pointer or offset that bookmarks a specific point in the source table’s timeline. When queried, the stream leverages the table’s version history to retrieve only the rows that changed after this offset, along with relevant metadata. It enables Snowflake to reconstruct the minimal set of changes in the table since the last time the stream was accessed, efficiently managing data updates and preserving table history. Snowpipe is a serverless data ingestion service from Snowflake that enables fast, automated data loading into Snowflake tables. It continuously ingests data from files when available in a specified stage, supporting micro-batch loading and providing users near-real-time access to new data. This approach eliminates the need for manually running scheduled `COPY` commands for extensive batch loading, enabling a more efficient and timely data pipeline.

5.7.1 Create a Snowflake Stream for Customer Tracker data in STAGE Schema on STAGE.CustomerTrack_Traffic_Data Table

```
create or replace stream STAGE.CustomerTrack_traffic_data_changes on
table STAGE.CustomerTrack_Traffic_Data
```

5.8 Change Data Capture

Change Data Capture (CDC) is an advanced data integration technique that identifies and processes changes made to data in a source system. This technique enables organizations to synchronize this data across multiple environments with minimal latency [3]. This process is beneficial for enabling real-time analytics, maintaining up-to-date data warehouses, and supporting zero-downtime migrations and business continuity.

How CDC Works: At its core, the CDC captures, inserts, updates, and deletes events in source tables, transmitting only the modified data to target systems. This selective approach reduces the need for complete data reloads, making CDC an efficient alternative to batch processing.

The CDC implements various mechanisms, including:

1. **Log-Based CDC:** Tracks changes directly from transaction logs. This method, often supported by relational databases, is highly efficient because it does not interfere with source data operations [7].
2. **Trigger-Based CDC:** This approach utilizes database triggers to capture events in real time. While this approach can add some overhead, it offers a reliable means of tracking changes [7].
3. **Timestamp-Based CDC**: This method uses timestamp fields to detect changes made within a specific time. Though less immediate, it is effective for lower-frequency updates [7].
4. **Snapshot-Based CDC:** Periodically compares database snapshots, identifying changes as discrepancies between the snapshots. This method is less real-time but valuable in some analytics contexts [8].

5.8.1 Key Benefits of CDC

1. **Real-Time Data Synchronization**: CDC enables the continuous and near real-time replication of data, which is critical for applications requiring timely information, such as real-time analytics, fraud detection, and supply chain management.
2. **Efficient Data Migration and Integration:** CDC simplifies data management by focusing on changes instead of reloading all data. This approach reduces bandwidth usage and leads to smoother, more cost-effective cloud migrations and hybrid data integrations, making the process efficient for all involved.
3. **Scalability for Distributed Systems**: CDC efficiently synchronizes data over wide-area networks to support large-scale, distributed data architectures, including multi-cloud and hybrid environments.
4. **Applications in Data Warehousing and Analytics:** Change Data Capture (CDC) commonly works with cloud data warehousing solutions like Snowflake and Amazon Redshift, as well as with data lakes. This approach keeps analytics systems updated with the latest operational data, facilitating timely insights and enhancing data-driven decision-making [9].

5.8.2 Snowflake Change Data Capture

Snowflake Change Data Capture (CDC) is a method for tracking and capturing changes made to data within a Snowflake data warehouse, enabling real-time synchronization with other systems, databases, or applications. This technique leverages change streams, which are log-based systems that capture data modifications such as inserts, updates, and deletes [3].

A stream in Snowflake takes logical snapshots of the source objects, such as external tables, underlying tables, or views. It continuously logs changes to the source data and stores the captured changes in the stream. This log of data changes is updated in real-time, ensuring that any modifications to the data are tracked and available for downstream systems [3].

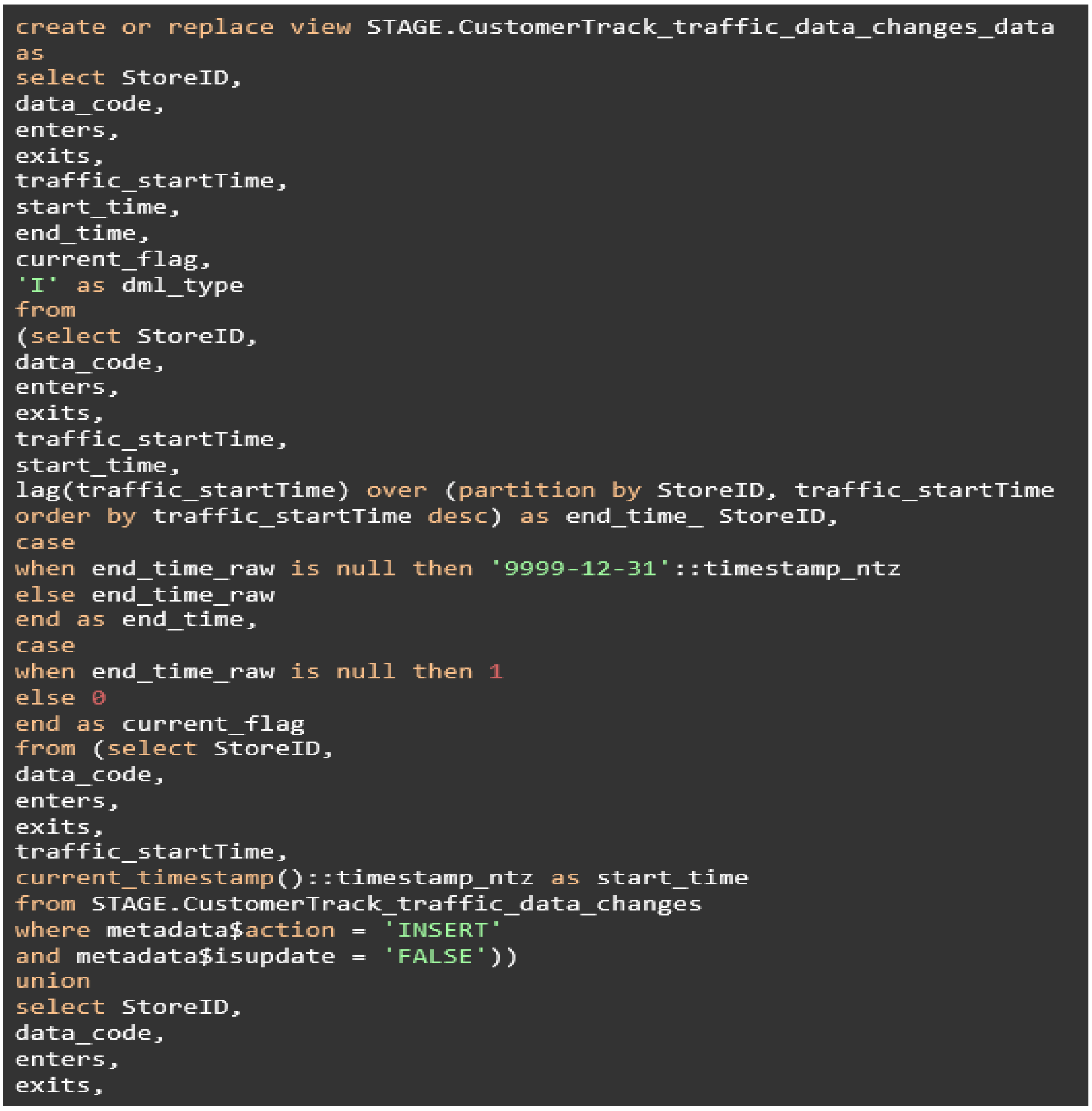
The process starts by creating a CDC-enabled Snowflake table that captures data changes. This table automatically tracks modifications, including metadata about the changes, and sends this data to a target system for replication.

Once we create a stream, add metadata columns to the source table to track data modifications. These columns store detailed information about each change, ensuring we capture the data accurately. We must consume or move tracked data changes to permanent storage within a retention period. If we do not, those changes will become inaccessible, and we will need to create a new stream to continue tracking modifications.

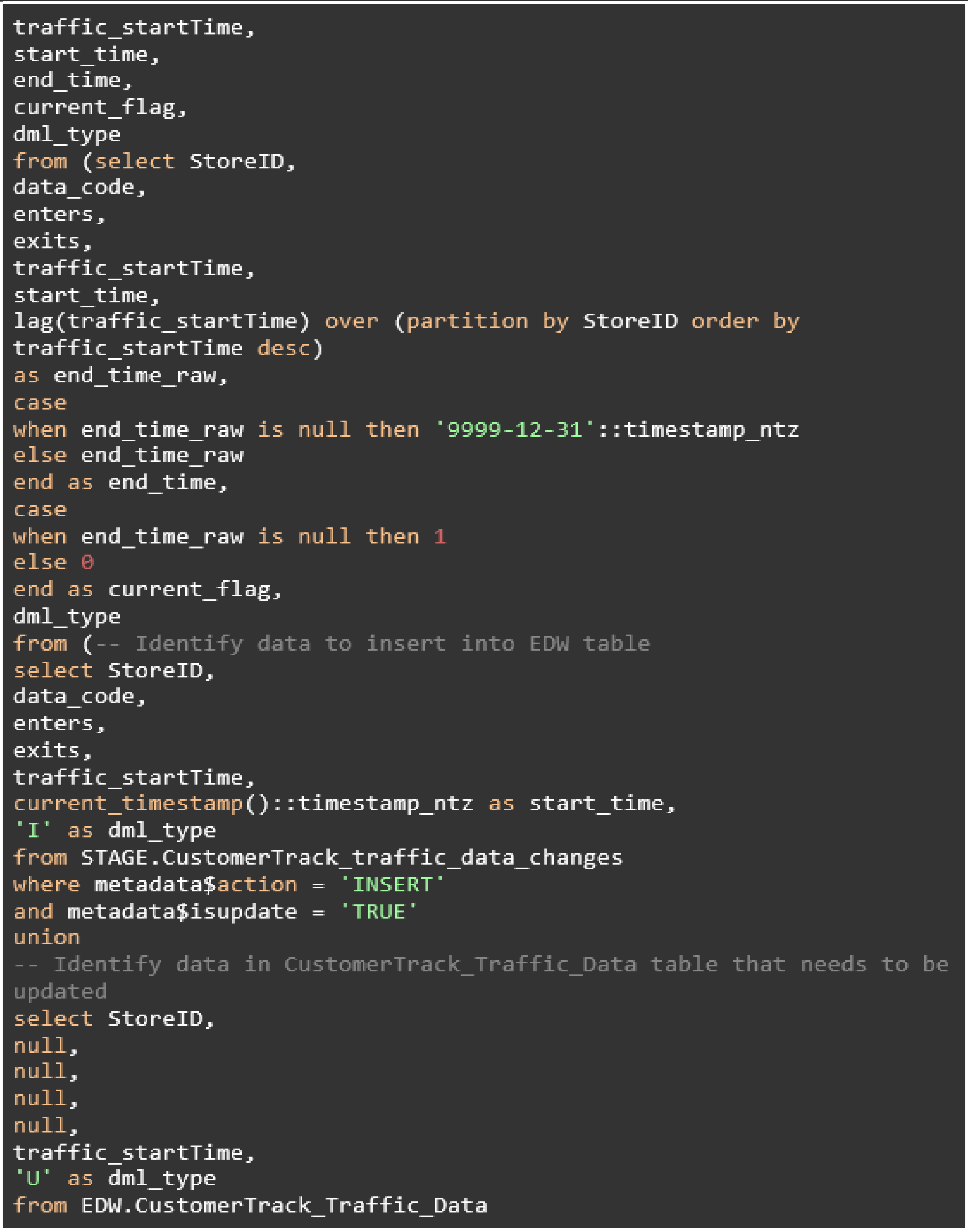
5.8.3 The key benefits of Change Data Capture (CDC) in Snowflake

1. **Real-Time Data Synchronization**: CDC in Snowflake enables real-time data synchronization across various systems. This ensures that data is consistently updated and available, allowing businesses to make time-sensitive decisions with the most current data. It is particularly beneficial in high-velocity data environments like e-commerce, banking, and fraud detection [3].
2. **Efficient Data Replication**: Snowflake CDC captures and replicates only the changes made to data (inserts, updates, and deletes) rather than the entire dataset, which minimizes data movement. This efficiency makes it ideal for cloud-based architectures, where transferring large volumes of data over networks can be expensive and time-consuming [3].
3. **Zero-Downtime Data Migrations**: CDC helps migrate databases with zero downtime, facilitating smooth transitions between systems or during database upgrades. Snowflake's CDC capabilities enable continuous tracking of data changes without interrupting the system's ongoing operations [3].
4. **Scalability & Cost-Effectiveness:** Snowflake's cloud-native architecture makes CDC highly scalable. It can efficiently handle large volumes of change data, which is crucial for enterprises dealing with large datasets. Moreover, it reduces the need for bulk data operations and batch processing, reducing costs associated with these tasks [3].
5. **Simplified Data Warehousing**: With Snowflake's CDC, organizations ensure that their data warehouse continuously updates with the most recent changes. It improves data consistency and ensures that analytical systems always reflect the latest information without requiring complex ETL pipelines [3].

5.8.4 Create view to capture the changes and avoid duplicates in target table of Customer Tracker

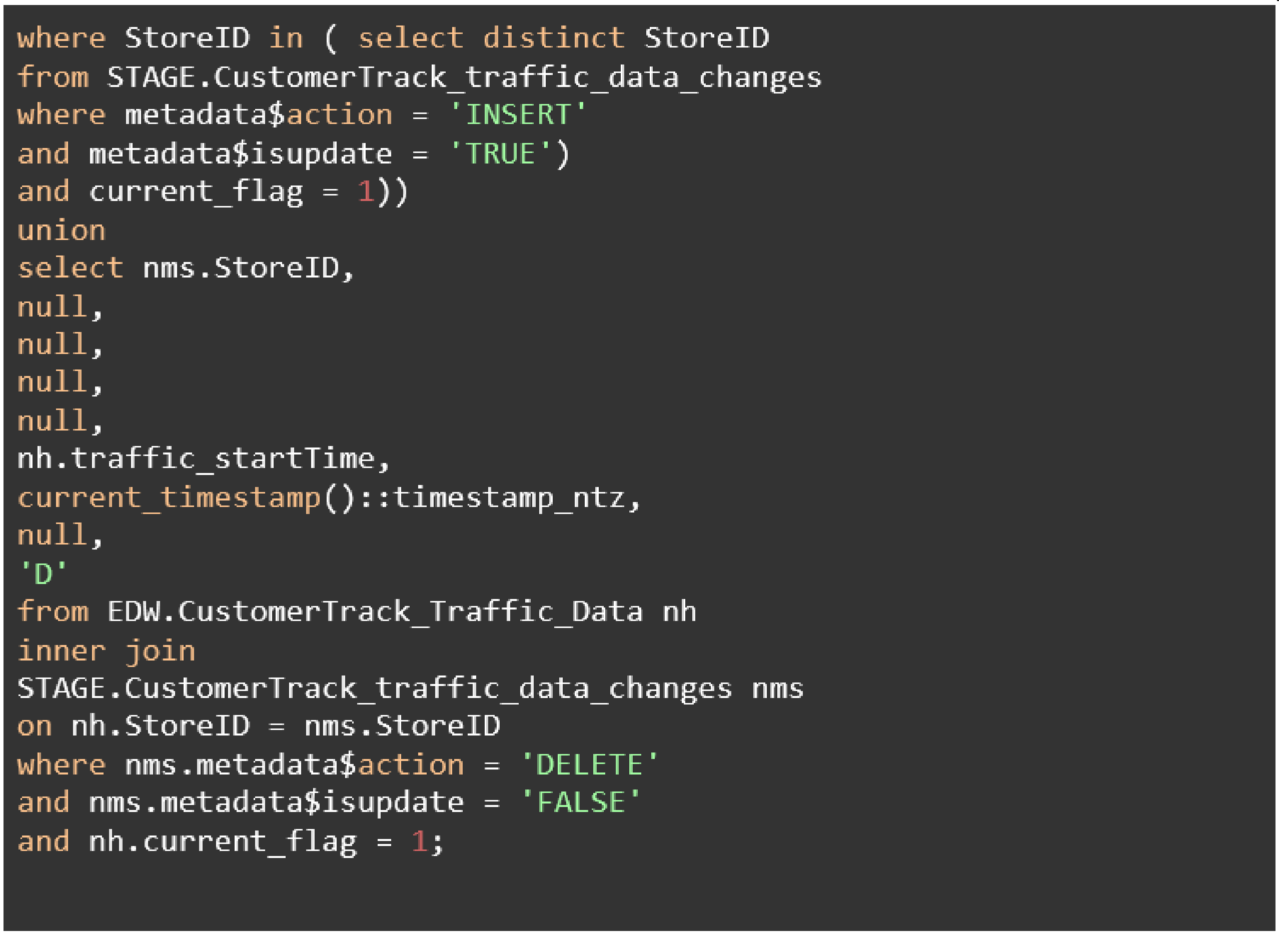


```
create or replace view STAGE.CustomerTrack_traffic_data_changes_data
as
select StoreID,
data_code,
enters,
exits,
traffic_startTime,
start_time,
end_time,
current_flag,
'I' as dml_type
from
(select StoreID,
data_code,
enters,
exits,
traffic_startTime,
start_time,
lag(traffic_startTime) over (partition by StoreID, traffic_startTime
order by traffic_startTime desc) as end_time_ StoreID,
case
when end_time_raw is null then '9999-12-31'::timestamp_ntz
else end_time_raw
end as end_time,
case
when end_time_raw is null then 1
else 0
end as current_flag
from (select StoreID,
data_code,
enters,
exits,
traffic_startTime,
current_timestamp()::timestamp_ntz as start_time
from STAGE.CustomerTrack_traffic_data_changes
where metadata$action = 'INSERT'
and metadata$isupdate = 'FALSE'))
union
select StoreID,
data_code,
enters,
exits,
```



```
traffic_startTime,
start_time,
end_time,
current_flag,
dml_type
from (select StoreID,
data_code,
enters,
exits,
traffic_startTime,
start_time,
lag(traffic_startTime) over (partition by StoreID order by
traffic_startTime desc)
as end_time_raw,
case
when end_time_raw is null then '9999-12-31'::timestamp_ntz
else end_time_raw
end as end_time,
case
when end_time_raw is null then 1
else 0
end as current_flag,
dml_type
from (-- Identify data to insert into EDW table
select StoreID,
data_code,
enters,
exits,
traffic_startTime,
current_timestamp()::timestamp_ntz as start_time,
'I' as dml_type
from STAGE.CustomerTrack_traffic_data_changes
where metadata$action = 'INSERT'
and metadata$isupdate = 'TRUE'
union
-- Identify data in CustomerTrack_Traffic_Data table that needs to be
updated
select StoreID,
null,
null,
null,
null,
traffic_startTime,
'U' as dml_type
from EDW.CustomerTrack_Traffic_Data
```

International Journal of Core Engineering & Management
Volume-6, Issue-11, 2021 ISSN No: 2348-9510



```
where StoreID in ( select distinct StoreID
from STAGE.CustomerTrack_traffic_data_changes
where metadata$action = 'INSERT'
and metadata$isupdate = 'TRUE')
and current_flag = 1))
union
select nms.StoreID,
null,
null,
null,
null,
nh.traffic_startTime,
current_timestamp()::timestamp_ntz,
null,
'D'
from EDW.CustomerTrack_Traffic_Data nh
inner join
STAGE.CustomerTrack_traffic_data_changes nms
on nh.StoreID = nms.StoreID
where nms.metadata$action = 'DELETE'
and nms.metadata$isupdate = 'FALSE'
and nh.current_flag = 1;
```

5.9 Snowflake Tasks

In Snowflake, a task is a feature that allows us to schedule and automate SQL statements to run at specified intervals or in response to certain conditions. Tasks can be beneficial for automating recurring data processing workflows, like data loading, transformation, and preparation for analytics. Tasks are executed according to a set schedule or a dependency chain, beginning only after the completion of the preceding task [3]

Tasks in Snowflake are commonly used to:

1. Automate ETL/ELT Processes: Tasks can load data from various sources, transform it, and save it into Snowflake tables, enabling streamlined data integration and preparation for analytics.
2. Manage Data Pipelines: Tasks support the creation of complex workflows by defining chains of tasks, where one task can trigger another, allowing a series of operations to occur in a specific sequence.
3. Trigger Data Ingestion with Snowpipe: Snowflake tasks can be used alongside Snowpipe to load data into tables as new files arrive in external stages (e.g., AWS S3).
4. Run SQL Statements on a Schedule: With scheduled execution, tasks can perform periodic maintenance, data transformation, or reporting tasks.

5.10 Snowflake Merge

In Snowflake, the MERGE command is a DML (Data Manipulation Language) operation that enables users to perform conditional updates, inserts, and deletes on a target table based on the results of a join with a source table [3]. The MERGE statement is often used for data synchronization tasks, such as keeping a target table updated with new or modified data from a source.

In Snowflake, Tasks and the MERGE command can automate data synchronization or update processes, especially in ETL (Extract, Transform, Load) workflows.

5.10.1 Query to update target Customer Tracker table without any duplicates using Task and Merge

```
create or replace task Populate_CustomerTrack_Traffic_Data
warehouse = CustomerTrack_ETL
schedule = '1 minute'
when
system$stream_has_data('STAGE.CustomerTrack_traffic_data_changes')
as
merge into EDW.CustomerTrack_Traffic_Data nh
using STAGE.CustomerTrack_traffic_data_changes_data m
   on nh.StoreID = m.StoreID
   and nh.traffic_startTime = m.traffic_startTime
when matched and m.dml_type = 'U' then update
      set nh.end_time = m.end_time,
      nh.current_flag = 0
when matched and m.dml_type = 'D' then update
      set nh.end_time = m.end_time,
      nh.current_flag = 0
when not matched and m.dml_type = 'I' then insert
           (StoreID, data_code, enters, exits,
traffic_startTime, start_time, end_time, current_flag)
      values (m.StoreID, m.data_code, m.enters, m.exits,
m.traffic_startTime,
           m.start_time, m.end_time, m.current_flag);
```

Pause the execution of Snowpipe created for Customer Tracker Data

```
Alter Task Populate_CustomerTrack_Traffic_Data Resume
```

VI. LIMITATIONS AND CHALLENGES

1. Data Latency: This guide on Change Data Capture focuses on near-real-time processing, but users might encounter some latency during data transfer and loading stages. This latency makes it less suitable for data solutions that demand ultra-low latency.
2. Data Quality: The change data capture process depends heavily on the consistency of the source system data. If the data from the source system is inconsistent, additional validation and cleansing processes will be necessary.
3. Scalability Issues: As data volumes grow, large file sizes in S3 and high-frequency updates could strain Snowpipe and Streams, impacting performance.
4. Cost Implications: Scaling Snowflake and Amazon S3 leads to higher data storage and processing costs when handling high data volumes.
5. Error Handling: Developers may need to intervene manually to resolve errors or failures in the data pipeline, especially in real-time data flows.
6. Compliance and Security: Ensuring data privacy and governance in cloud environments (e.g., GDPR, HIPAA) adds complexity and requires continuous monitoring.

These challenges necessitate meticulous planning, monitoring, and optimization to uphold system reliability, scalability, and efficiency.

VII. CONCLUSION

Snowflake's Change Data Capture (CDC) has modernized data change management, making traditional CDC methods less relevant. Known as a fast, cloud-native data platform with adaptable storage and processing options, Snowflake further elevates its capabilities with CDC. It is especially effective for high-transaction environments where millions of records are modified daily.

Change Data Capture (CDC) efficiently identifies the data changes and eliminates the need to reload entire datasets and conserving computational, storage resources. By using the MERGE command, it effectively updates target tables with just the delta, significantly improves data accuracy, and ensures more reliable insights.

REFERENCES

1. M. Sacco, F. Malomo, and H. Wang, Snowflake Cloud Data Platform: A Step-by-Step Guide to Modern Cloud Analytics. Sebastopol, CA: O'Reilly Media, 2020.
2. D. Abadi, "Data Management in the Cloud: Limitations and Opportunities," Communications of the ACM, vol. 52, no. 6, pp. 45-51, 2019.
3. Snowflake Inc., Snowflake Documentation. Accessed: Jan. 7, 2021. [Online]. Available: https://docs.snowflake.com
4. Giri, V., & Gupta, S. (2018). Capture Streaming Data with Change-Data-Capture. In Practical Enterprise Data Lake Insights (pp. 87–123). Apress L. P. https://doi.org/10.1007/978-1-4842-3522-5_3
5. Medjahed, B., Ouzzani, M., & Elmagarmid, A. (2009). Generalization of acid properties.
6. Baldoni, R., Contenti, M., Virgillito, A. (2003). The Evolution of Publish/Subscribe Communication Systems. In: Schiper, A., Shvartsman, A.A., Weatherspoon, H., Zhao, B.Y. (eds) Future Directions in Distributed Computing. Lecture Notes in Computer Science, vol

2584. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-37795-6_25

7. Eccles, Mitchell (2013). Pragmatic development of service based real-time change data capture. PHD thesis, Aston University.
8. Wei Du and X. Zou, "Differential snapshot algorithms based on Hadoop MapReduce," 2015 12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD), Zhangjiajie, 2015, pp. 1203-1208, doi: 10.1109/FSKD.2015.7382113
9. Devarasetty, N. (2017). Scalable Data Engineering Platforms for AI-Powered Business Intelligence. International Journal of Machine Learning Research in Cybersecurity and Artificial Intelligence, 8(1), 1-27.