American Journal of Engineering Research (AJER) ***2024***
American Journal of Engineering Research (AJER)
e-ISSN: 2320-0847 p-ISSN : 2320-0936
Volume-13, Issue-11, pp-01-10
www.ajer.org
Research Paper Open Access

Optimizing Delta Load Processes from SAP HANA to Microsoft Fabric (One Lake) Using Azure Data Factory (ADF) and Database Triggers

Govinda Rao Banothu
SAP Analytics Cloud Technical Specialist, NJ, USA.
[1]Corresponding Author: https://orcid.org/0009-0000-4728-7684

ABSTRACT : *This paper examines the optimization of delta load processes from SAP HANA to Microsoft Fabric utilizing Azure Data Factory (ADF) and database triggers. As organizations increasingly seek real-time data integration solutions, delta loading emerges as a critical technique for transferring only modified data, thereby reducing bandwidth and processing time. The study outlines the architecture and methodologies for implementing delta loads, focusing on the role of database triggers in SAP HANA to capture data changes efficiently. By leveraging ADF's capabilities for orchestration and transformation, the proposed framework enhances data movement to Microsoft Fabric, enabling organizations to gain timely insights and improve decision-making. This paper discusses the challenges associated with traditional data loading methods, including data consistency and latency, and present strategies to address these issues through automation and streamlined workflows. Real-world case studies illustrate the effectiveness of the approach, highlighting performance improvements and operational efficiencies. This research paper contributes to the field of data engineering by providing actionable insights for organizations aiming to optimize their data integration processes in cloud environments.*

KEYWORDS *Delta Load, SAP HANA, One Lake Storage, Azure Data Factory, Data Integration, Change Data Capture, Data Pipeline Optimization.*

Date of Submission: 26-10-2024 Date of acceptance: 07-11-2024

I. INTRODUCTION

SAP HANA (High-Performance Analytic Appliance) is a revolutionary in-memory database and application platform designed to handle both transactional (OLTP) and analytical (OLAP) workloads in real-time.

Online Transaction Processing (OLTP) systems are designed to handle many short online transaction requests. They focus on managing day-to-day operations and ensuring data integrity and consistency.

Online Analytical Processing (OLAP) systems are designed for complex queries and data analysis, providing insights, and supporting decision-making processes.

Key Features: -

- Columnar Storage: Data is stored in columns rather than rows, which optimizes read operations and improves data compression.
- Massive Parallel Processing: Utilizes multiple CPU cores to process large volumes of data simultaneously, enhancing performance.
- Data Compression: Techniques like dictionary compression reduce memory usage, allowing more data to be stored efficiently.
- Code Push Down: This paradigm allows data-intensive calculations to be executed directly in the database layer, minimizing data movement, and improving performance.

Azure Data Factory (ADF) is a cloud-based data integration service that enables organizations to design, schedule, and manage workflows driven by data, facilitating the orchestration of data movement and

transformation. It is designed to handle complex hybrid extract-transform-load (ETL) and extract-load-transform (ELT) processes.

Key Features: -

- Broad Connectivity: ADF offers extensive support for various data sources, including databases, file systems, and SaaS applications, enabling smooth data integration.
- Data Movement: The Copy Activity allows users to move data between different data stores efficiently.
- Data Transformation: Users can transform data using mapping data flows or by leveraging compute services like Azure Databricks and Azure HDInsight.
- Triggers and Scheduling: ADF supports various triggers (scheduled, event-based) to automate pipeline execution based on specific conditions.
- Monitoring and Management: Built-in monitoring tools provide insights into pipeline performance, allowing users to track success and failure rates.

II. LITERATURE REVIEW

The graph below shows that cloud adoption is expected to grow significantly in the next 12-18 months. The number of organizations running more than 50% of their workloads in the cloud is expected to double.

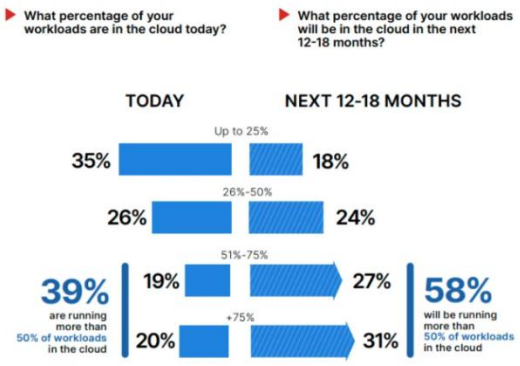


Fig. 1. Share of Workloads in the Cloud [1].

Full Load: -

A full load involves transferring an entire dataset from a source system to a target system. This process typically occurs during the initial data migration or when a comprehensive refresh of the dataset is needed. Full loads can be time-consuming and resource-intensive, especially for large datasets, as they require copying all records regardless of changes since the last transfer.

Delta Load: -

Delta load, on the other hand, refers to the process of transferring only the data that has changed since the last load. This includes new records, updates to existing records, and deletions. Delta loading is generally more efficient than full loading, as it reduces the amount of data moved and minimizes the load on both source and target systems.

Importance Of Delta Load: -

Delta load is crucial for several reasons:

- Efficiency: By transferring only changed data, delta loads significantly reduce the volume of data processed, saving time and resources.
- Timeliness: Delta loading allows for more frequent updates, enabling organizations to have access to near real-time data, which is essential for timely decision-making.
- Reduced Impact on Systems: Since delta loads require less processing power and bandwidth, they minimize the impact on source systems, ensuring they remain responsive for transactional operations.
- Cost-Effectiveness: With reduced data transfer and processing requirements, organizations can lower operational costs associated with data storage and movement.

In summary, delta loading is a vital practice for organizations looking to maintain current and accurate data while optimizing performance and resource utilization.

And we need Delta Loads as well to be loaded into the cloud because delta loading is vital for organizations as it enables efficient data integration by transferring only the changes made since the last load. This approach significantly reduces data volume, saving bandwidth and processing time. It supports near real-time insights, allowing businesses to make timely decisions based on the latest information. Additionally, delta loads lower storage and processing costs, enhance data quality, and facilitate scalability as data volumes grow. Overall, delta loading helps organizations optimize their data management processes and maintain a competitive edge in a data-driven environment.

III. CHALLENGES IN DELTA LOADING

Integrating SAP ERP systems and Microsoft One lake and extracting delta loads becomes more difficult cause as per the following [6] SAP Note 3255746: Unpermitted usage of Operational Delta Provisioning (ODP).

SAP restricts the use of CDC Connector with that the delta loads becomes too difficult, and the Organizations are forced to either stay with SAP Warehouse Solution or Use HANA Connectors only. That is where this paper comes in handy in pulling delta loads conveniently.

Delta loading, which involves loading only the changes (inserts, updates, deletes) since the last load, presents several challenges related to data consistency and integrity:

Data Integrity:

Partial updates: If updates are made to records during the delta loading process, there's a risk of loading outdated or incomplete data.

Simultaneous Modifications:

Multiple users or systems may update the same records concurrently, leading to conflicting changes that can result in data integrity issues.

Tracking Changes:

Change Detection: Accurately identifying which records have changed since the last load can be complex, especially if the source system lacks robust change tracking mechanisms.

Timestamps: Relying on timestamps can be problematic if system clocks are not synchronized, leading to missed or duplicated records.

Data Validation: Ensuring that the changes comply with business rules and integrity constraints can be challenging, particularly for complex data models.

IV. METHODOLOGY

Data extraction techniques from SAP HANA via ADF:

Requirements Gathering:

Identify Data Sources:
Determine the specific SAP HANA tables or views to be extracted.

Understand Business Needs*:*
Gather requirements regarding the frequency, volume, and purpose of the data extraction.

Environment Setup:

Azure Subscription*:*
Make sure you have an active Azure subscription that grants you access to Azure Data Factory.

SAP HANA Configuration:
Verify that SAP HANA is accessible, and appropriate users have permissions to extract data.

Azure Data Factory Configuration:

Create ADF Instance:

1) Log in to the Azure portal.

2) Create a new Azure Data Factory instance.

Integration Runtime:

Set up a self-hosted integration runtime (SHIR) if SAP HANA is on-premises or not directly accessible via ADF.

SHIR is a service that enables secure data integration between cloud services and on-premises data sources. It allows organizations to move and transform data across different environments, ensuring that data remains accessible and can be processed without extensive reconfiguration or risk.

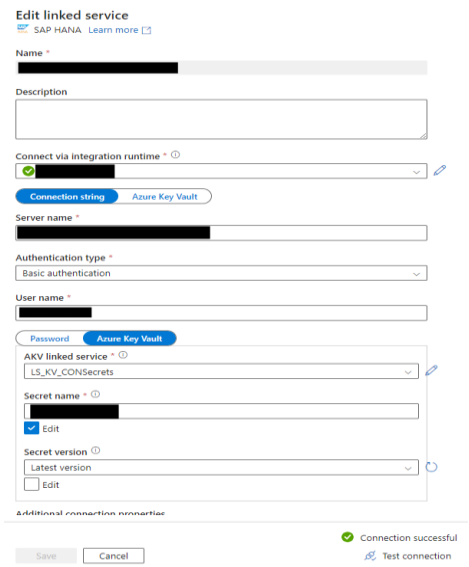
Establish Connectivity to SAP HANA:

Linked Service Configuration:

In ADF, create a linked service for SAP HANA. Specify connection details such as server name, port, database name, and authentication method.

Test Connectivity:

Validate the connection to ensure it is correctly set up.



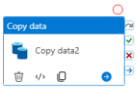
Data Pipeline Design:

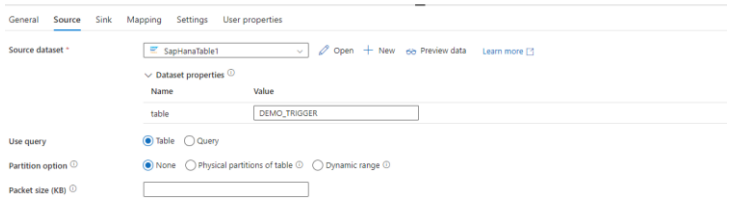
Create Data Pipeline:

Design a data pipeline in ADF to manage the data extraction process.

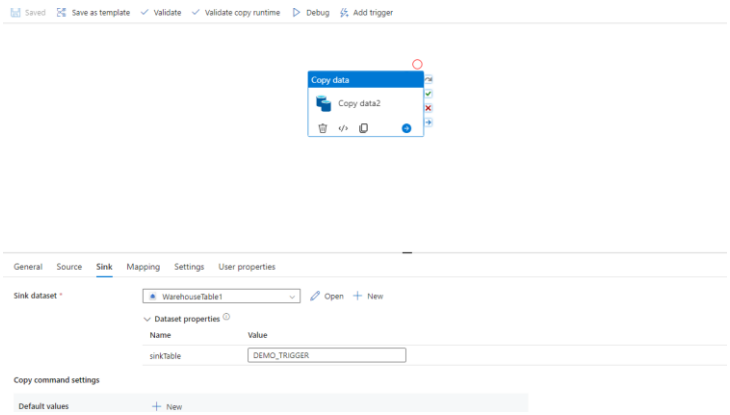
Source Dataset Configuration:

Define a source dataset pointing to the desired SAP HANA tables or views.





Sink Dataset Configuration:
Define a sink dataset, specifying the destination for the extracted data (e.g., Azure Blob Storage, Azure SQL Database).



Dataflow Development:

Mapping Data Flows:
Optionally, create data flows within ADF to transform the data during extraction if necessary (e.g., filtering, aggregating).

Set Up Data Movement Activities:
Use Copy Data activities to define how data will be transferred from SAP HANA to the destination.

Parameterization and Scheduling:

Parameterization*:*
Implement parameters in the pipeline to allow for dynamic filtering or control over the extraction process (e.g., date ranges).

Trigger Configuration:

Schedule the pipeline execution using triggers (e.g., time-based, event-based). Make sure that you first publish to the adf-publish branch.

And upon successful execution of the stored procedure will be merged into the original table.

Monitoring and Error handling

Set Up Monitoring:
Use ADF monitoring features to track the execution status of pipelines.

Implement Error Handling:
Configure retry policies and alerts for failures to ensure robust data extraction.

V. OPTIMIZED SOLUTION

Setting up HANA database triggers:

Firstly, we need a Loging Table which is going to be exactly the replica of the Source table but without any keys defined and 2 additional Columns for timestamp and DML operation.

Original Table: -

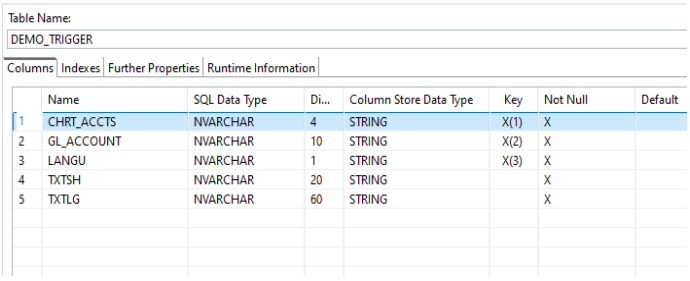
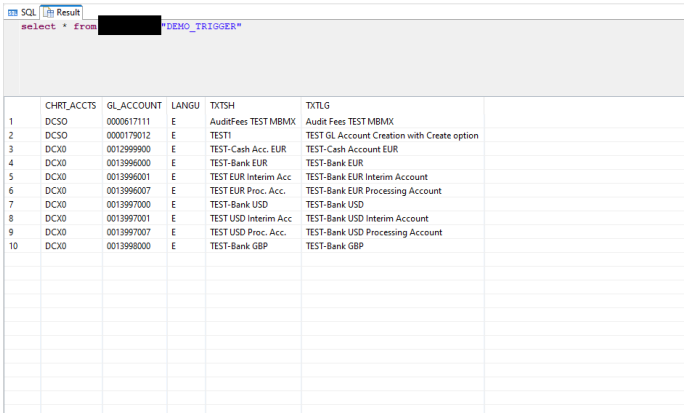


Table Name:
DEMO_TRIGGER

Columns | Indexes | Further Properties | Runtime Information

	Name	SQL Data Type	Di...	Column Store Data Type	Key	Not Null	Default
1	CHRT_ACCTS	NVARCHAR	4	STRING	X(1)	X	
2	GL_ACCOUNT	NVARCHAR	10	STRING	X(2)	X	
3	LANGU	NVARCHAR	1	STRING	X(3)	X	
4	TXTSH	NVARCHAR	20	STRING		X	
5	TXTLG	NVARCHAR	60	STRING		X	



SQL | Result

```
select * from [redacted]"DEMO_TRIGGER"
```

	CHRT_ACCTS	GL_ACCOUNT	LANGU	TXTSH	TXTLG
1	DCSO	0000617111	E	AuditFees TEST MBMX	Audit Fees TEST MBMX
2	DCSO	0000179012	E	TEST1	TEST GL Account Creation with Create option
3	DCX0	0012999900	E	TEST-Cash Acc. EUR	TEST-Cash Account EUR
4	DCX0	0013996000	E	TEST-Bank EUR	TEST-Bank EUR
5	DCX0	0013996001	E	TEST EUR Interim Acc	TEST-Bank EUR Interim Account
6	DCX0	0013996007	E	TEST EUR Proc. Acc.	TEST-Bank EUR Processing Account
7	DCX0	0013997000	E	TEST-Bank USD	TEST-Bank USD
8	DCX0	0013997001	E	TEST USD Interim Acc	TEST-Bank USD Interim Account
9	DCX0	0013997007	E	TEST USD Proc. Acc.	TEST-Bank USD Processing Account
10	DCX0	0013998000	E	TEST-Bank GBP	TEST-Bank GBP

Logging Table: -

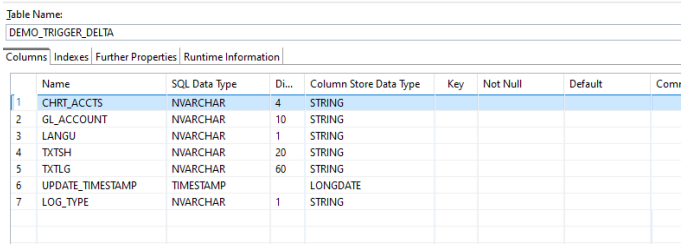


Table Name:
DEMO_TRIGGER_DELTA

Columns | Indexes | Further Properties | Runtime Information

	Name	SQL Data Type	Di...	Column Store Data Type	Key	Not Null	Default	Com
1	CHRT_ACCTS	NVARCHAR	4	STRING				
2	GL_ACCOUNT	NVARCHAR	10	STRING				
3	LANGU	NVARCHAR	1	STRING				
4	TXTSH	NVARCHAR	20	STRING				
5	TXTLG	NVARCHAR	60	STRING				
6	UPDATE_TIMESTAMP	TIMESTAMP		LONGDATE				
7	LOG_TYPE	NVARCHAR	1	STRING				

SQL Result

```
select * from [redacted]."DEMO_TRIGGER_DELTA"
```

	CHRT_ACCTS	GL_ACCOUNT	LANGU	TXTSH	TXTLG	UPDATE_TIMESTAMP	LOG_TYPE
1	DCSO	0001333111	E	INSERT Test	INSERT Test Long	Nov 1, 2024, 7:51:50.166 AM	I
2	DCX0	0013998000	E	UPSERT Test	UPSERT Test Long	Nov 1, 2024, 7:53:18.43 AM	U
3	DCSO	0000179012	E	TEST1	TEST GL Accoun...	Nov 1, 2024, 7:54:26.885 AM	D

After the creation of the Logging Table comes the part of Database Triggers. Triggers can be defined for the following events:

INSERT: Triggered when a new row is added [2].

Procedure Name: DEMO_TRIGGER_I_SP

Create Statement

```
CREATE TRIGGER [redacted]."DEMO_TRIGGER_I_SP" AFTER INSERT ON [redacted]."DEMO_TRIGGER" REFERENCING NEW ROW DEMO_TRIGGER_REF FOR EACH ROW
BEGIN INSERT
INTO [redacted]."DEMO_TRIGGER_DELTA" VALUES(:DEMO_TRIGGER_REF.CHRT_ACCTS,
    :DEMO_TRIGGER_REF.GL_ACCOUNT,
    :DEMO_TRIGGER_REF.LANGU,
    :DEMO_TRIGGER_REF.TXTSH,
    :DEMO_TRIGGER_REF.TXTLG,
    CURRENT_UTCTIMESTAMP,
    'I' )
;
END
;
```

UPDATE: Triggered when an existing row is modified.

Procedure Name: DEMO_TRIGGER_U_SP

Create Statement

```
CREATE TRIGGER [redacted]."DEMO_TRIGGER_U_SP" AFTER UPDATE ON [redacted]."DEMO_TRIGGER" REFERENCING NEW ROW DEMO_TRIGGER_REF FOR EACH ROW
BEGIN INSERT
INTO [redacted]."DEMO_TRIGGER_DELTA" VALUES(:DEMO_TRIGGER_REF.CHRT_ACCTS,
    :DEMO_TRIGGER_REF.GL_ACCOUNT,
    :DEMO_TRIGGER_REF.LANGU,
    :DEMO_TRIGGER_REF.TXTSH,
    :DEMO_TRIGGER_REF.TXTLG,
    CURRENT_UTCTIMESTAMP,
    'U' )
;
END
;
```

DELETE: Triggered when a row is removed.

Procedure Name: DEMO_TRIGGER_D_SP

Create Statement

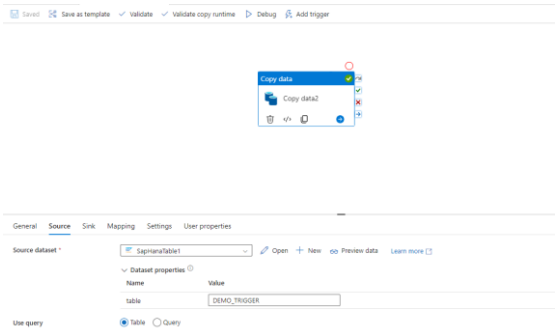
```
CREATE TRIGGER [redacted]."DEMO_TRIGGER_D_SP" AFTER DELETE ON [redacted]."DEMO_TRIGGER" REFERENCING OLD ROW DEMO_TRIGGER_REF FOR EACH ROW
BEGIN INSERT
INTO [redacted]."DEMO_TRIGGER_DELTA" VALUES(:DEMO_TRIGGER_REF.CHRT_ACCTS,
    :DEMO_TRIGGER_REF.GL_ACCOUNT,
    :DEMO_TRIGGER_REF.LANGU,
    :DEMO_TRIGGER_REF.TXTSH,
    :DEMO_TRIGGER_REF.TXTLG,
    CURRENT_UTCTIMESTAMP,
    'D' )
;
END
;
```

ADF pipeline design for delta loading:

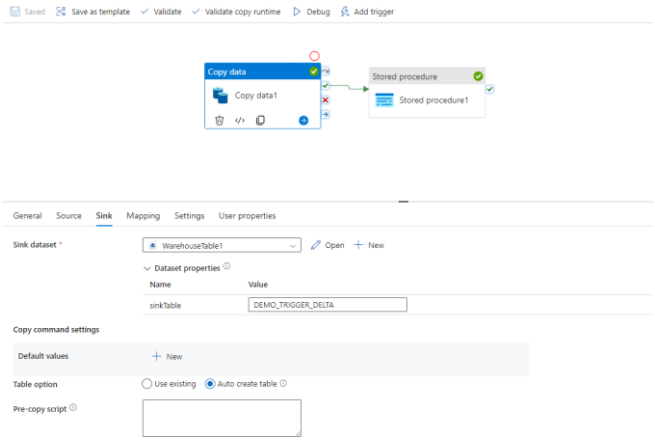
Create 2 Set of pipelines both with copy activity for Initial and Delta Loads.

Initial Pipeline: -

Copy activity with Source data set for Source table and sink data set for target table on Fabric Data Warehouse (One lake).



Delta Pipeline: -
Copy activity with source data for logging table and sink data for target table on Fabric data warehouse along with a stored procedure to process the logging table and then further updating the target table on fabric data warehouse.



Below is the SQL used for the stored procedure for processing the logging table: -

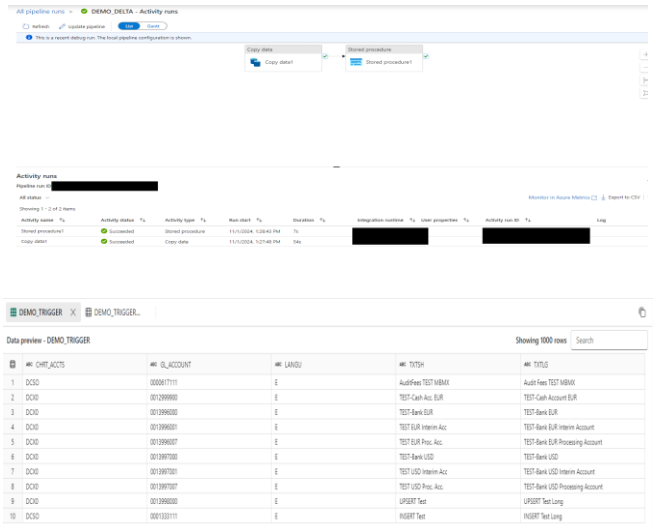
Preview Copilot uses AI. Mistakes can happen. Verify code suggestions before running them. Review terms

```
CREATE PROC [DEMO].[DELTA_MERGER]

AS
BEGIN

update tgt
set tgt.[TXTSH] = src.[TXTSH],
    tgt.[TXTLG] = src.[TXTLG]
from [DEMO_LOGGING].[DEMO].[DEMO_TRIGGER] AS tgt
inner join [DEMO_LOGGING].[DEMO].[DEMO_TRIGGER_DELTA] AS src
on tgt.[CHRT_ACCTS] = src.[CHRT_ACCTS]
and tgt.[GL_ACCOUNT] = src.[GL_ACCOUNT]
and tgt.[LANGU] = src.[LANGU]
and src.[LOG_TYPE] in ('U');


insert into [DEMO_LOGGING].[DEMO].[DEMO_TRIGGER] ([CHRT_ACCTS],
                [GL_ACCOUNT],
                [LANGU],
                [TXTSH],
                [TXTLG])
SELECT
                [CHRT_ACCTS],
                [GL_ACCOUNT],
                [LANGU],
                [TXTSH],
                [TXTLG]
        FROM
                [DEMO_LOGGING].[DEMO].[DEMO_TRIGGER_DELTA]
        WHERE
                [LOG_TYPE] in ('I') ;


delete [DEMO_LOGGING].[DEMO].[DEMO_TRIGGER]
from [DEMO_LOGGING].[DEMO].[DEMO_TRIGGER] as tgt
inner join [DEMO_LOGGING].[DEMO].[DEMO_TRIGGER_DELTA] as src
on tgt.[CHRT_ACCTS] = src.[CHRT_ACCTS] and
                tgt.[GL_ACCOUNT] = src.[GL_ACCOUNT] and
                tgt.[LANGU] = src.[LANGU] and
                [LOG_TYPE] = 'D';

END
```

Results:

And now after execution of Delta Pipeline the delta records will be first extracted from the HANA system.

And upon successful execution of the stored procedure will be merged into the original table.



VI. CONCLUSION

In conclusion, this research paper presents a comprehensive framework for optimizing delta load processes from SAP HANA to Microsoft Fabric utilizing Azure Data Factory (ADF) and database triggers. The study highlights the importance of delta loading in real-time data integration, enabling organizations to transfer only modified data, reduce bandwidth and processing time, and gain timely insights. The proposed approach leverages ADF's capabilities for orchestration and transformation, and database triggers in SAP HANA to capture data changes efficiently. The research addresses the challenges associated with traditional data loading methods, including data consistency and latency, and presents strategies to address these issues through automation and streamlined workflows. The real-world case studies demonstrate the effectiveness of the approach, showcasing performance improvements and operational efficiencies. This research contributes to the field of data engineering by providing actionable insights for organizations aiming to optimize their data integration processes in cloud environments. The findings of this study can be applied to various industries and organizations seeking to improve their data management processes, reduce costs, and enhance decision-making capabilities.

VII. FUTURE WORK

Future work can involve real time replication from Source tables, Automatic generation of pipelines leveraging the meta data which can be extracted from SAP HANA Systems, Cleanup of logging tables from source system.

REFERENCES

[1]. 101 Shocking Cloud Computing Statistics (UPDATED 2024) Reference. [Online]. Available at https://www.cloudzero.com/blog/cloud-computing-statistics/

[2]. CREATE TRIGGER Statement (Data Definition) | SAP Help Portal Reference. [Online]. Available at https://help.sap.com/docs/SAP_HANA_PLATFORM/4fe29514fd584807ac9f2a04f6754767/20d5a65575191014946db96aaedbef5b.html

[3]. Copy activity - Azure Data Factory & Azure Synapse | Microsoft Learn https://learn.microsoft.com/en-us/azure/data-factory/copy-activity-overview
[4]. SAP HANA Cloud, SAP HANA Database Deployment Infrastructure (HDI) Reference. [Online]. Available at: https://help.sap.com/docs/hana-cloud-database/sap-hana-cloud-sap-hana-database-deployment-infrastructure-hdi-reference/triggers-hdbtrigger
[5]. High Performance Analytical Application Reference. [Online]. Available at: https://www.jetir.org/papers/JETIRBD06049.pdf
[6]. Unpermitted usage of ODP Data Replication APIs. Reference. [Online]. Available at: https://me.sap.com/notesLatestChanges/0003255746/E/diff