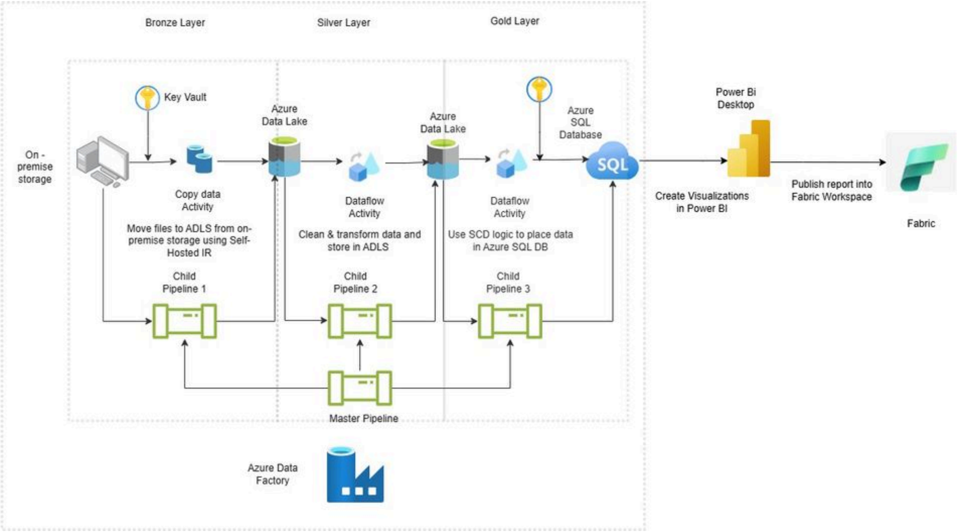
Project Implementation

Architecture Overview

The pipeline architecture has been carefully designed to balance modularity, security, scalability, and maintainability. The architecture aligns with industry standards for cloud-based data platforms and supports the full data lifecycle from ingestion to insight.



Architecture of the project

Data Ingestion Layer (Bronze):

This layer receives raw files from a backend team’s on-premise folder. The files include:

- accounts.csv
- customers.csv
- loan_payments.csv
- loans.csv
- transactions.csv

ADF’s copy activities move these files into the bronze container in ADLS Gen2 with minimal or no transformation. Schema enforcement ensures that only valid data is accepted.

Transformation Layer (Silver):

This layer processes the raw data into a clean, deduplicated, and enriched form. ADF Dataflows are used with:

- Aggregate transformations to detect duplicates
- Window transformations for ranking and ordering
- Filter transformations for validation rules (e.g., no null IDs)

The transformed data is written into the silver container in a CSV format.

Curated Layer (Gold):

This layer focuses on business-ready datasets. Using Dataflows again, SCD logic is applied:

- SCD Type 1 tables for current snapshot updates
- SCD Type 2 tables for historical tracking with "effective_start_date", "effective_end_date", and "is_active" flags

The results are upserted into Azure SQL Database, which is optimized for reporting.

Reporting Layer:

Power BI connects to the Azure SQL Database and uses tables from the gold layer to create insightful dashboards. Reports include trends in loan payments, customer segmentation, account status, and more. The final visuals are published into a Fabric workspace for sharing with stakeholders.

Orchestration:

A Master Pipeline in ADF uses Execute Pipeline activities to call three child pipelines in sequence:

- Bronze ingestion
- Silver transformation
- Gold SCD loading

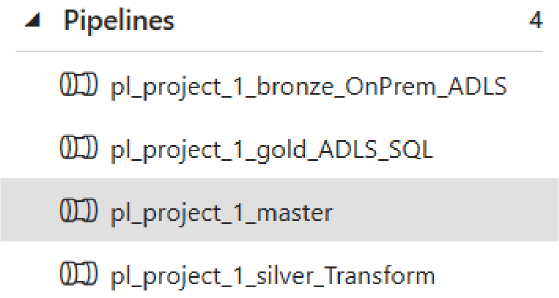
Security & Governance:

All sensitive information such as database credentials and storage keys are stored in Azure Key Vault. Access to these secrets is granted to ADF using managed identities, ensuring compliance and security.

Project Implementation

STEP 1 - Create the Master pipeline

The pipeline `pl_project_1_master` orchestrates a three-stage data processing flow by executing three sub-pipelines in a sequential manner. Each stage is responsible for a different level of data transformation following a **Bronze** → **Silver** → **Gold** architecture. This modular approach ensures a clean separation of concerns and efficient data processing.



Pipelines 4

pl_project_1_bronze_OnPrem_ADLS

pl_project_1_gold_ADLS_SQL

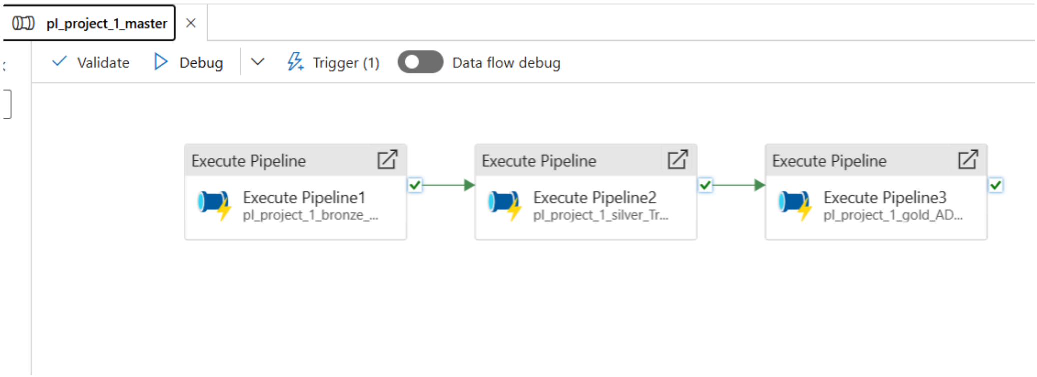
pl_project_1_master

pl_project_1_silver_Transform

Pipelines used for this project

Pipeline Objective

To coordinate the end-to-end data flow from on-premises SQL Server to ADLS Gen2, apply transformations, and load curated data into Azure SQL Database.



Master pipeline overview

Step 1.1: Bronze Layer Ingestion (Raw Data Load)

Activity Name: `Execute Pipeline1`
Referenced Pipeline: `pl_project_1_bronze_OnPrem_ADLS`
Purpose:

- This pipeline ingests raw data from the on-premises SQL Server using a Self-hosted Integration Runtime.
- Data is loaded into the Bronze layer (Raw zone) in Azure Data Lake Storage Gen2 (ADLS Gen2).
- The data is stored in its original format (CSV or Parquet), without any transformations.

Key Features:

- No dependencies, starts first.
- Waits until completion before next activity proceeds (`waitOnCompletion: true`).

Step 1.2: Silver Layer Transformation (Cleaned & Filtered)

Activity Name: `Execute Pipeline2`
Referenced Pipeline: `pl_project_1_silver_Transform`
Depends On: `Execute Pipeline1` (on Succeeded status)

Purpose:

- Performs data cleaning and transformation tasks such as:
    - Removing nulls
    - Deduplication
    - Data type standardization
    - Filtering based on business logic
- Data is moved from Bronze to Silver layer in ADLS Gen2.
- Output is a cleaner and more structured dataset ready for analytics.

Key Features:

- Executes only if `Execute Pipeline1` completes successfully.
- Ensures quality data moves to the next step.

Step 1.3: Gold Layer Load (Curated for Consumption)

Activity Name: `Execute Pipeline3`
Referenced Pipeline: `pl_project_1_gold_ADLS_SQL`
Depends On: `Execute Pipeline2` (on Succeeded status)

Purpose:

- Loads curated, transformed data from the Silver zone in ADLS Gen2 to Azure SQL Database.
- Suitable for reporting, dashboards, and downstream consumption.
- May include dimension and fact table population, slowly changing dimension (SCD) logic, or merge statements.

Key Features:

- Triggered only upon successful transformation in `Execute Pipeline2`.
- Ensures only valid and ready-to-consume data is loaded into the SQL DB.

Pipeline Architecture Summary

Stage	Pipeline Reference	Description	Output Location
Bronze	`pl_project_1_bronze_OnPrem_ADLS`	Raw ingestion from On-Prem SQL to ADLS	ADLS Gen2 (Bronze)

Silver	`pl_project_1_silver_Transform`	Data cleaning and transformation	ADLS Gen2 (Silver)
Gold	`pl_project_1_gold_ADLS_SQL`	Load into Azure SQL DB for reporting	Azure SQL Database

Best Practices Followed

- **Modular Pipelines**: Each layer is a separate pipeline, making it reusable and easier to maintain.
- **Sequential Execution**: Ensures upstream tasks succeed before proceeding, preventing data issues.
- **Parameterization (Implied)**: Likely used in sub-pipelines to make paths and table names dynamic (not shown in code but recommended).

STEP 2 - Create the pipeline for Bronze layer - Child pipeline 1

The pipeline `pl_project_1_bronze_OnPrem_ADLS` performs the ingestion of raw data from an **on-premises File Server** to **Azure Data Lake Storage Gen2 (ADLS Gen2)**. This forms the **Bronze Layer** in a multi-stage data lake architecture.

Pipeline Objective

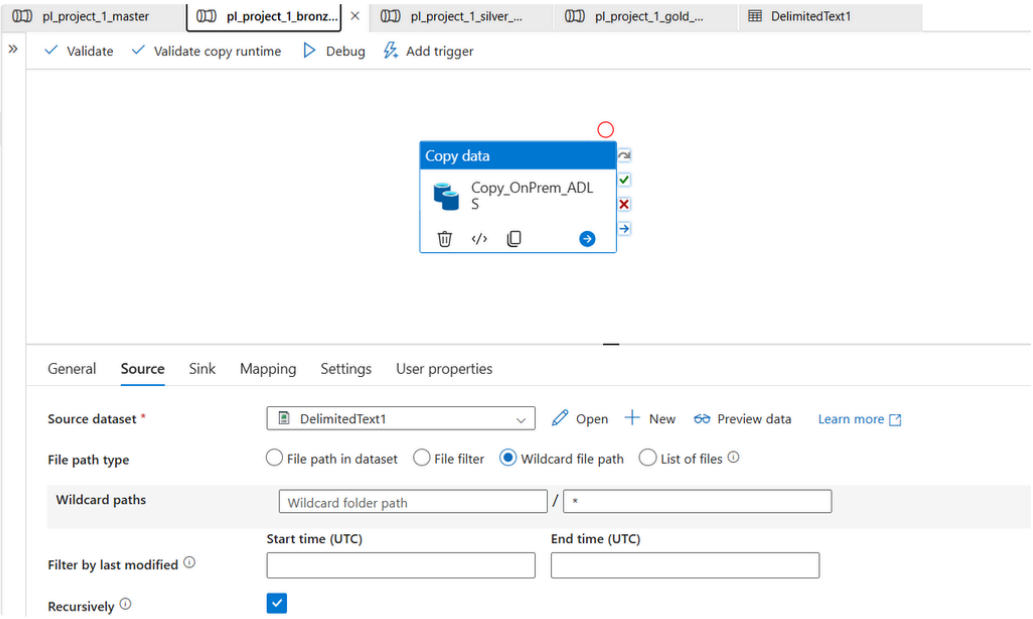
To ingest raw data files from an on-premise file system into the Bronze zone of ADLS Gen2 in CSV format using a robust and automated Copy activity.

Step 2.1: Copy Activity - OnPremise to ADLS Gen2

Activity Name: `Copy_OnPrem_ADLS`
Activity Type: `Copy Data`

Purpose:

- Transfers data from an on-prem file system (likely mounted using Self-hosted Integration Runtime) to a cloud destination (ADLS Gen2).
- Reads delimited text (CSV) files recursively and writes them in a structured format into the Bronze container.



Source dataset * DelimitedText1

File path type ○ File path in dataset ○ File filter ● Wildcard file path ○ List of files

Wildcard paths Wildcard folder path / *

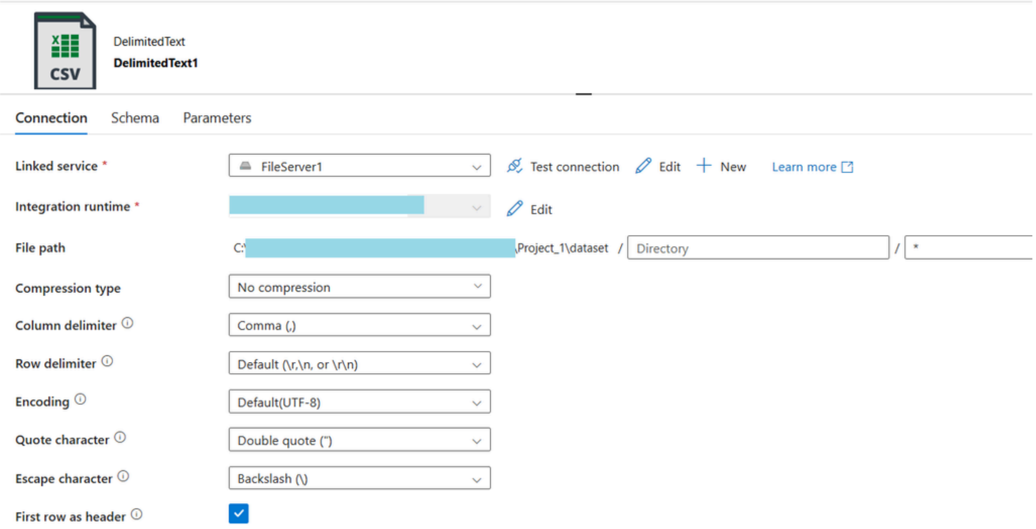
Filter by last modified Start time (UTC) End time (UTC)

Recursively ☑

Copy Activity - Source Configuration

Source Configuration

- **Source Type**: `DelimitedTextSource`



Linked service *: FileServer1

Integration runtime *: [redacted]

File path: C:\[redacted]\Project_1\dataset / Directory / *

Compression type: No compression

Column delimiter: Comma (,)

Row delimiter: Default (\r,\n, or \r\n)

Encoding: Default(UTF-8)

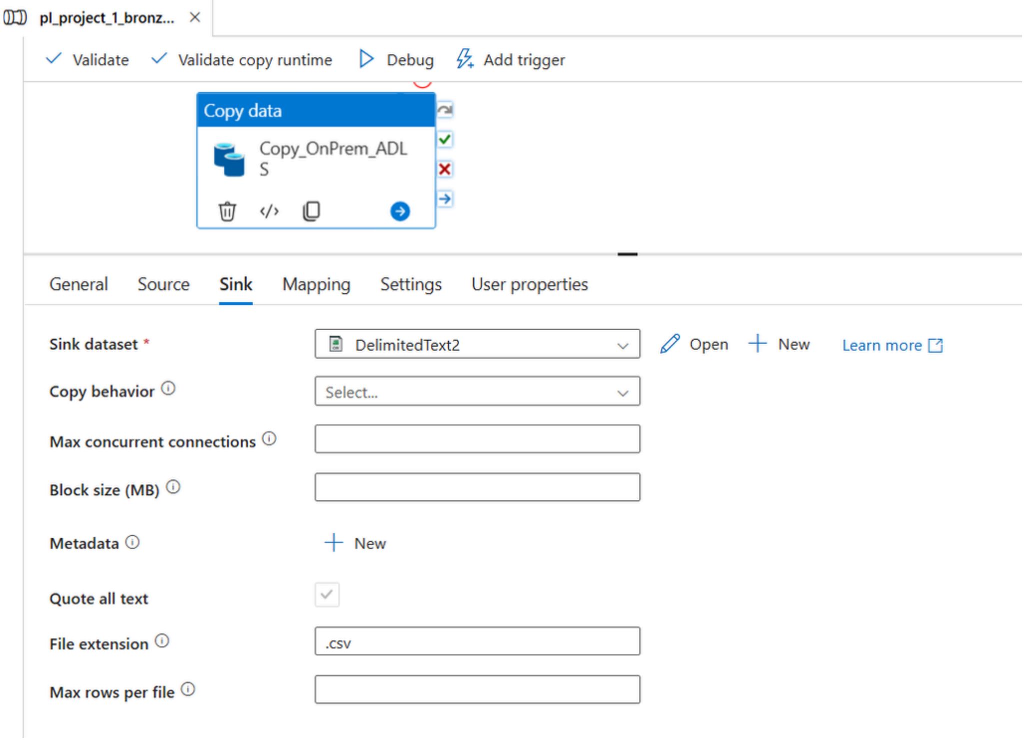
Quote character: Double quote (")

Escape character: Backslash (\)

First row as header: ☑

Source Dataset

- **Store Settings**:
  - **Type**: `FileServerReadSettings`
  - **Recursive**: `true` (reads files from all subfolders)
  - **Wildcard FileName**: `*` (reads all file names)
  - **Partition Discovery**: Disabled
- **Format Settings**:
  - Reads files as delimited text (default CSV handling)



Sink dataset *: DelimitedText2

Copy behavior: Select...

Max concurrent connections:

Block size (MB):

Metadata: + New

Quote all text: ☑

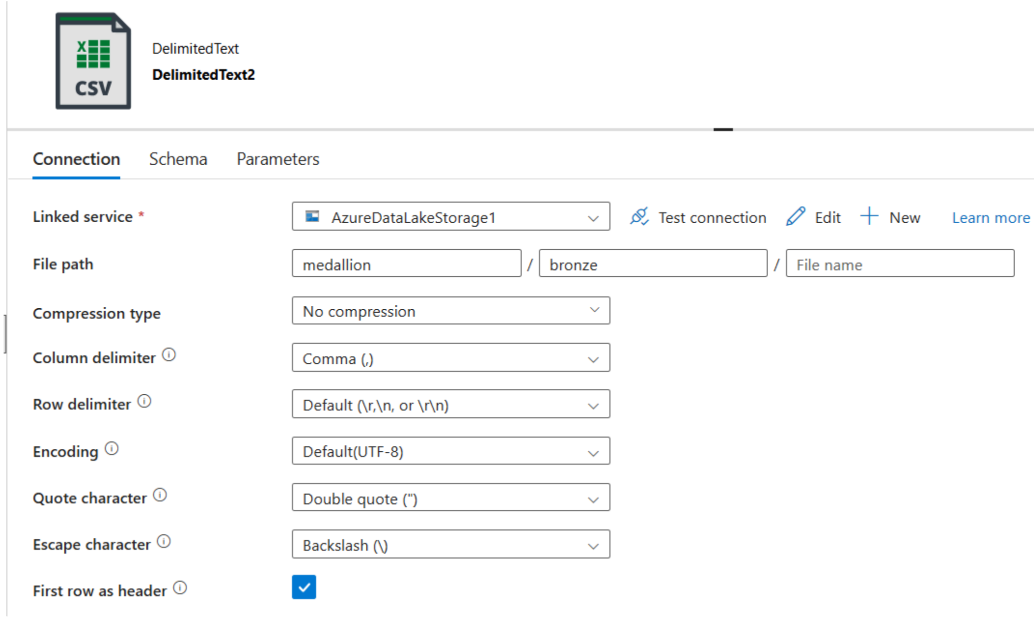
File extension: .csv

Max rows per file:

Copy Activity - Sink Configuration

Sink Configuration

- **Sink Type**: `DelimitedTextSink` (writes data as CSV to ADLS)


DelimitedText
DelimitedText2

Linked service * AzureDataLakeStorage1

File path: medallion / bronze / File name

Compression type: No compression

Column delimiter: Comma (,)

Row delimiter: Default (\r,\n, or \r\n)

Encoding: Default(UTF-8)

Quote character: Double quote (")

Escape character: Backslash (\)

First row as header: ☑

Sink Dataset

- **Store Settings**:
  - **Type**: `AzureBlobFSWriteSettings` (for writing to ADLS Gen2)
- **Format Settings**:
  - `quoteAllText` : `true` (quotes all fields for consistency)
  - `fileExtension` : `.csv` (output format)

Dataset References

- **Input Dataset**: `DelimitedText1` (point to on-prem file system)
- **Output Dataset**: `DelimitedText2` (point to ADLS Gen2 location)

Pipeline Architecture Summary

Stage	Activity Name	Description	Source Location	Destination Location
Bronze	`Copy_OnPrem_ADLS`	Copy raw delimited files to ADLS Gen2	On-Prem File Server	ADLS Gen2 (Bronze)

Best Practices Followed

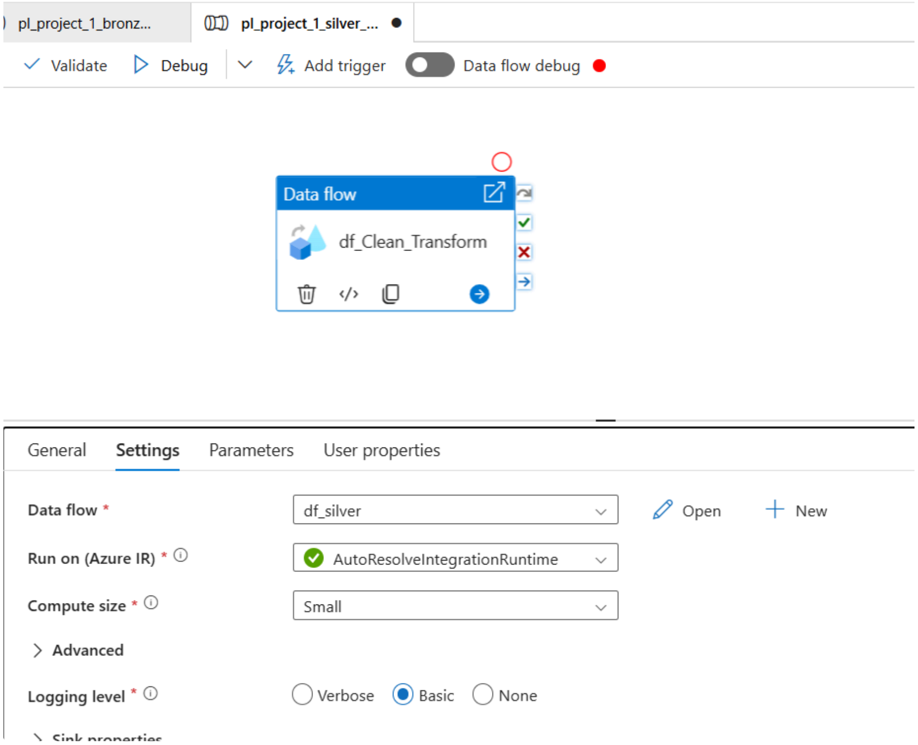
- **Recursive Read**: Ensures all relevant files across subfolders are ingested.
- **Wildcard File Name**: Makes ingestion dynamic and future-proof.
- **Format Consistency**: Quotes all fields and uses `.csv` extension.
- **Modular Design**: Can be reused or scaled to handle more file types or locations.

STEP 3 - Create the pipeline for Silver layer - Child pipeline 2

The `pl_project_1_silver_Transform` pipeline focuses on data cleaning and transformation. It takes raw data from the bronze layer (ADLS Gen2), removes duplicates and nulls, standardizes column formats, and prepares the data for advanced processing. This layer ensures that only validated, structured, and refined data flows into the next stage, improving quality and reliability.

Pipeline Objective:
This pipeline and its associated dataflow perform the transformation and cleaning of raw (bronze) data into a refined (silver) format, using Azure Data Factory Mapping Data Flows.

Pipeline Name: `pl_project_1_silver_Transform`
Dataflow Name: `df_silver`



Data flow * : df_silver

Run on (Azure IR) * : AutoResolveIntegrationRuntime

Compute size * : Small

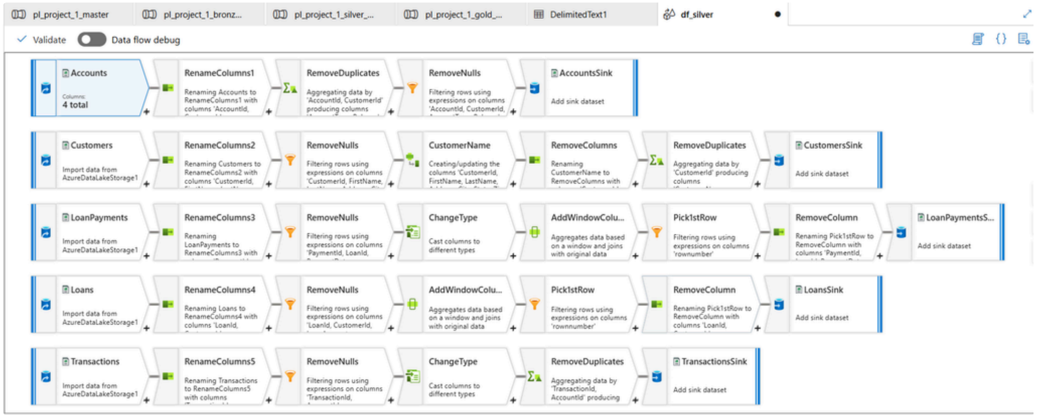
Logging level * : ○ Verbose ● Basic ○ None

Silver layer pipeline

Pipeline Activity: df_Clean_Transform

- **Type:** ExecuteDataFlow
- **Dataflow Reference:** df_silver
- **Compute:** General (8 cores)
- **Trace Level:** Coarse
- **Purpose:** Executes the `df_silver` dataflow to clean and transform data from multiple sources and write them to corresponding sinks in ADLS Gen2.

Dataflow Overview: df_silver

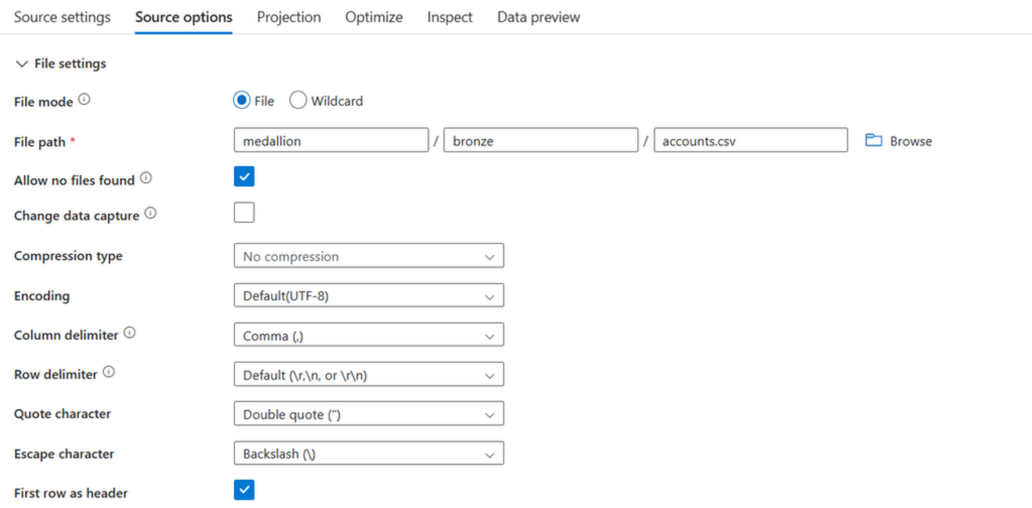

pl_project_1_master
pl_project_1_bronz...
pl_project_1_silver_...
pl_project_1_gold_...
DelimitedText1
df_silver
Validate
Data flow debug
Accounts
Columns:
4 total
RenameColumns1
Renaming Accounts to RenameColumns1 with columns 'AccountId,
RemoveDuplicates
Aggregating data by 'AccountId, CustomerId' producing columns
RemoveNulls
Filtering rows using expressions on columns 'AccountId, CustomerId,
AccountsSink
Add sink dataset
Customers
Import data from AzureDataLakeStorage1
RenameColumns2
Renaming Customers to RenameColumns2 with columns 'CustomerId,
RemoveNulls
Filtering rows using expressions on columns 'CustomerId, FirstName,
CustomerName
Creating/updating the columns 'CustomerId, FirstName, LastName,
RemoveColumns
Renaming CustomerName to RemoveColumns with
RemoveDuplicates
Aggregating data by 'CustomerId' producing columns
CustomersSink
Add sink dataset
LoanPayments
Import data from AzureDataLakeStorage1
RenameColumns3
Renaming LoanPayments to RenameColumns3 with
RemoveNulls
Filtering rows using expressions on columns 'PaymentId, LoanId,
ChangeType
Cast columns to different types
AddWindowColu...
Aggregates data based on a window and joins with original data
Pick1stRow
Filtering rows using expressions on columns 'rownumber'
RemoveColumn
Renaming Pick1stRow to RemoveColumn with columns 'PaymentId,
LoanPaymentsS...
Add sink dataset
Loans
Import data from AzureDataLakeStorage1
RenameColumns4
Renaming Loans to RenameColumns4 with columns 'LoanId,
RemoveNulls
Filtering rows using expressions on columns 'LoanId, CustomerId,
AddWindowColu...
Aggregates data based on a window and joins with original data
Pick1stRow
Filtering rows using expressions on columns 'rownumber'
RemoveColumn
Renaming Pick1stRow to RemoveColumn with columns 'LoanId,
LoansSink
Add sink dataset
Transactions
Import data from AzureDataLakeStorage1
RenameColumns5
Renaming Transactions to RenameColumns5 with columns
RemoveNulls
Filtering rows using expressions on columns 'TransactionId,
ChangeType
Cast columns to different types
RemoveDuplicates
Aggregating data by 'TransactionId, AccountId' producing
TransactionsSink
Add sink dataset

df_silver Dataflow

Step 3.1: Sources (Bronze Layer - ADLS Gen2)

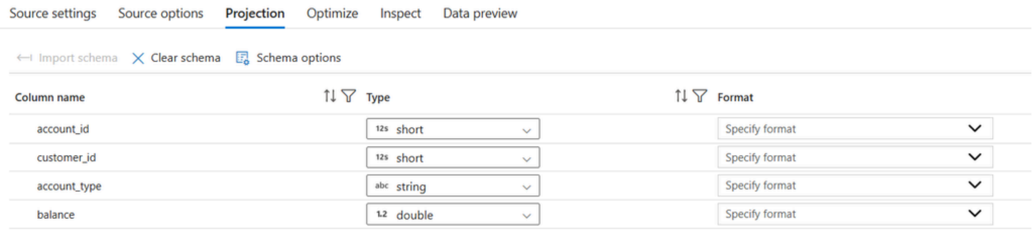
Each source reads raw CSV data with schema drift allowed and header-based column names. Schema is imported from the Projection tab.

Source	Source file	Key Columns	Description
`Accounts`	accounts.csv	account_id, customer_id, account_type, balance	Bank account information linked to customers
`Customers`	customers.csv	customer_id, first_name, last_name, address, city, state, zip	Personal information about customers
`LoanPayments`	loan_payments .csv	payment_id, loan_id, payment_date, payment_amount	Records of payments made on loans
`Loans`	loans.csv	loan_id, customer_id, loan_amount, interest_rate, loan_term	Loan details per customer
`Transactions`	transactions.cs v	transaction_id, account_id, transaction_date, transaction_amount, transaction_type	Individual transaction records for accounts



File mode	● File ○ Wildcard
File path *	medallion / bronze / accounts.csv
Allow no files found	☑
Change data capture	□
Compression type	No compression
Encoding	Default(UTF-8)
Column delimiter	Comma (,)
Row delimiter	Default (\r,\n, or \r\n)
Quote character	Double quote (")
Escape character	Backslash (\)
First row as header	☑

accounts.csv source for df_silver



Column name	Type	Format
account_id	short	Specify format
customer_id	short	Specify format
account_type	string	Specify format
balance	double	Specify format

Key Columns of account.csv

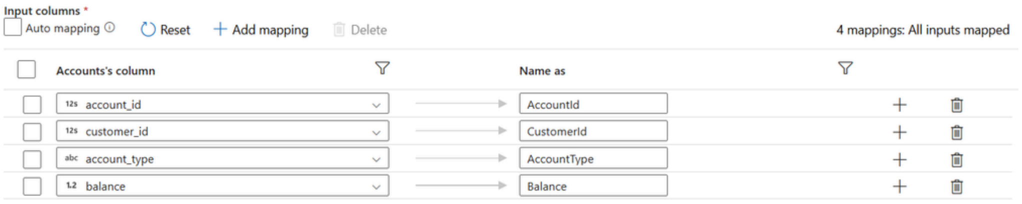
Transformations Applied

Step 3.2: Select Transformations (RenameColumns1 to RenameColumns5)

These map and rename columns to more meaningful and standardized names, which is a good practice for clarity and downstream usability.

Examples:

- `select1`: Maps `account_id` → `AccountId`, etc.
- `select2`: Maps customer fields and prepares them for further transformation.
- Similar renaming and field mapping is done for Loans, LoanPayments, and Transactions.



Input columns *

□ Auto mapping ⓘ   Reset   + Add mapping   Delete   4 mappings: All inputs mapped

□	Accounts's column		Name as
□	12s account_id	→	AccountId
□	12s customer_id	→	CustomerId
□	abc account_type	→	AccountType
□	1.2 balance	→	Balance

Select transformation - RenameColumns1 for Accounts source

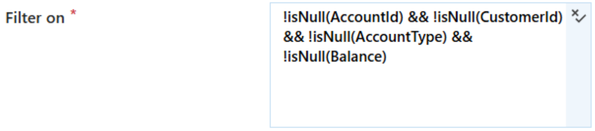
Step 3.3: Filtering Transformations (RemoveNulls1 to RemoveNulls5)

These are **data quality filters** to remove:

- Nulls in critical columns
- Invalid or zero values

Sample logic:

```
filter1 → Filters out rows with nulls in Account details
filter2 → Ensures all customer details are present
filter3 → Filters invalid LoanPayment records
filter4 → Validates Loans (must have positive amount, rate, term)
filter5 → Ensures Transactions have valid data
```



Filter on *

```
!isNull(AccountId) && !isNull(CustomerId)
&& !isNull(AccountType) &&
!isNull(Balance)
```

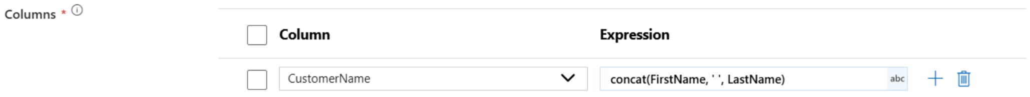
Filter Transformation - Remove Nulls in Account details

Step 3.4: Derived Column: `CustomerName`

Creates a new column:

```
CustomerName = concat(FirstName, ' ', LastName)
```

This is useful for generating a full customer name for analytics or reporting.



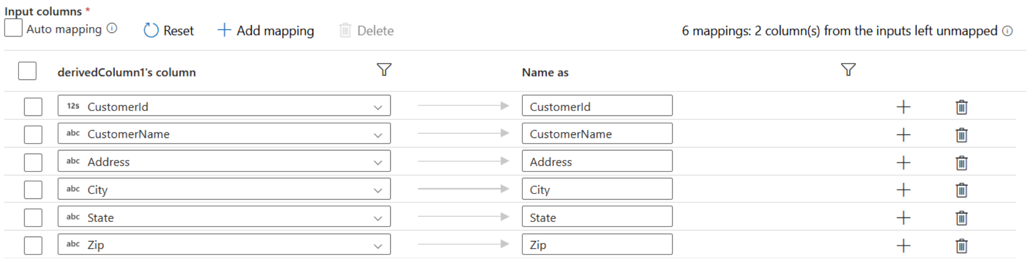
Columns * ⓘ

□	Column	Expression
□	CustomerName	concat(FirstName, ' ', LastName)

Derived column transformation - Concat FirstName and LastName of Customer to generate CostomerName column

Step 3.5: RemoveColumns1→ Final Cleaned Customer View

After deriving `CustomerName`, this select step reprojects only the required fields: `CustomerId`, `CustomerName`, Address info, etc. and remove the FirstName and LastName columns.



Input columns *

□ Auto mapping ⓘ Reset + Add mapping Delete 6 mappings: 2 column(s) from the inputs left unmapped ⓘ

□	derivedColumn1's column	Name as
□	12s CustomerId	CustomerId
□	abc CustomerName	CustomerName
□	abc Address	Address
□	abc City	City
□	abc State	State
□	abc Zip	Zip

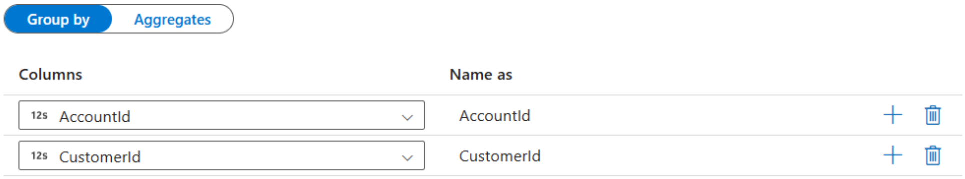
Select transformation - Removes unwanted columns and rearranges the order of the columns

Step 3.6: Aggregates (RemoveDuplicates1 to RemoveDuplicates3)

Aggregations group data for summarization and deduplication.

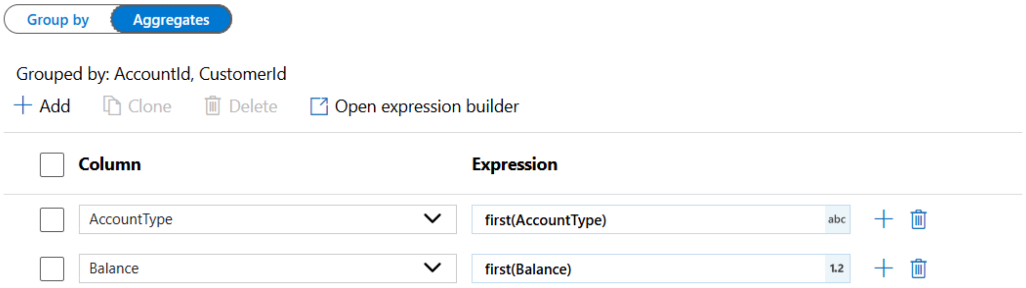
Examples:

- `RemoveDuplicates1`: Groups by `AccountId`, `CustomerId` to get one record per account
- `RemoveDuplicates2`: Groups by `CustomerId` to keep only one address set per customer



Columns	Name as
12s AccountId	AccountId
12s CustomerId	CustomerId

Aggregate transformation - Group by IDs for Accounts source



Grouped by: AccountId, CustomerId

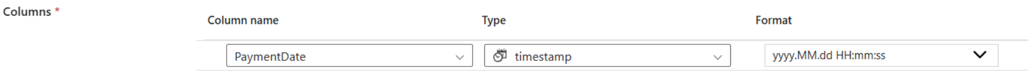
□	Column	Expression
□	AccountType	first(AccountType) abc
□	Balance	first(Balance) 1.2

Aggregate transformation - get the first row only from the list of duplicates for Accounts source

Step 3.7: Cast (ChangeType1, ChangeType2)

Converts data types, especially for:

- Date conversions
- Ensuring numeric precision This step is important for type consistency before writing to the sink.



Columns *

Column name	Type	Format
PaymentDate	timestamp	yyyy.MM.dd HH:mm:ss

Cast transformation - Change the data type of date columns, if any

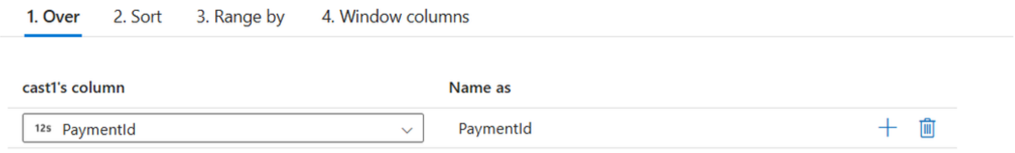
Step 3.8: Window Functions (AddWindowColumn1, AddWindowColumn2)

Used for row-wise comparisons, ranking, or detecting change over time.

Example:

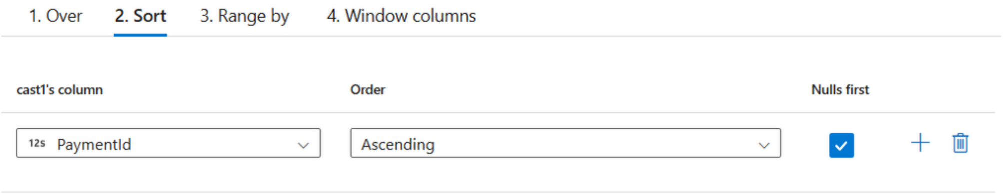
```
window1 → Might assign row numbers or rank based on PaymentId
```

This window function is used to group rows by the ID column and assign row numbers for all the rows in that particular group. This is useful when you want to remove duplicates by just selecting the first row.



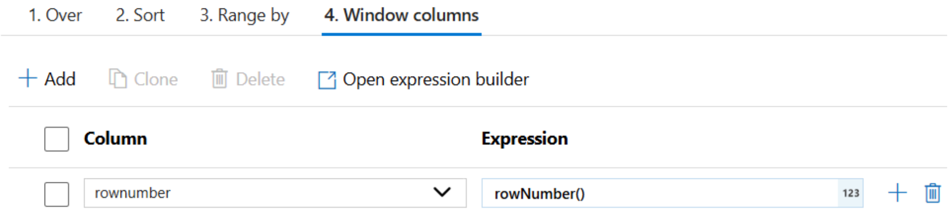
cast1's column	Name as
12S PaymentId	PaymentId

Window Transformation - over PaymentId column for LoanPayments source



cast1's column	Order	Nulls first
12S PaymentId	Ascending	☑

Window Transformation - Sort PaymentIds in Ascending order for LoanPayments source



	Column	Expression
□	rownumber	rowNumber()

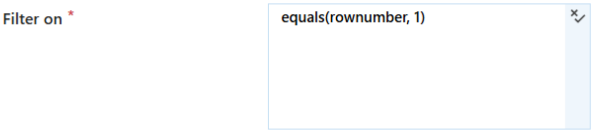
Window Transformation - get the row number for the corresponding PaymentId group for LoanPayments source

Step 3.9: Filtering with Window Output (Pick1stRow1, Pick1stRow2)

These filters remove unwanted rows after window logic.

Use cases:

- Retain only the **most recent payment**
- Keep only the **latest transaction per account**

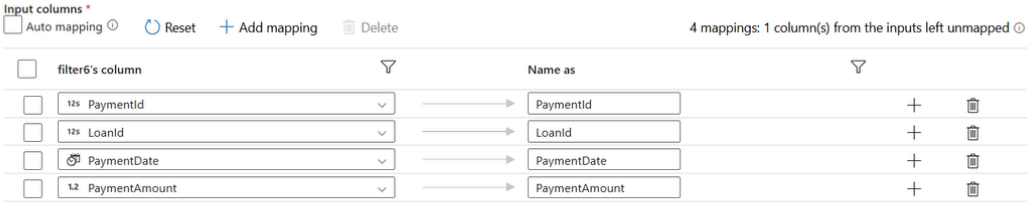


Filter on *: equals(rownumber, 1)

Filter Transformation - Remove duplicate rows for LoanPayments source

Step 3.10: Final Selects (RemoveColumns2, RemoveColumns3)

Extract only the final desired fields after all transformations are done, i.e, remove the rownumber column added in window transformation.



Input columns *

□ Auto mapping ⓘ · Reset · + Add mapping · Delete

4 mappings: 1 column(s) from the inputs left unmapped ⓘ

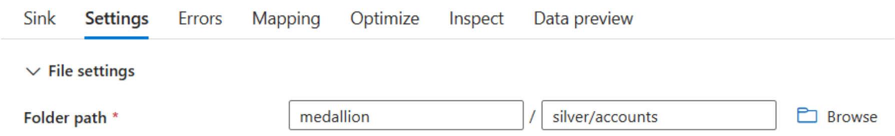
□	filter6's column		Name as
□	12s PaymentId	→	PaymentId
□	12s LoanId	→	LoanId
□	PaymentDate	→	PaymentDate
□	1.2 PaymentAmount	→	PaymentAmount

Select transformation - remove unwanted columns for LoanPayments source

Step 3.11: Sinks (Silver Layer - ADLS Gen2)

Refined outputs are written to ADLS Gen2 using the following sinks and store the cleaned files in the silver directory in the corresponding folders:

1. AccountsSink
2. CustomersSink
3. LoanPaymentsSinks
4. LoansSink
5. TransactionsSink



Sink · **Settings** · Errors · Mapping · Optimize · Inspect · Data preview

File settings

Folder path *: medallion / silver/accounts · Browse

Sink Settings for Accounts source

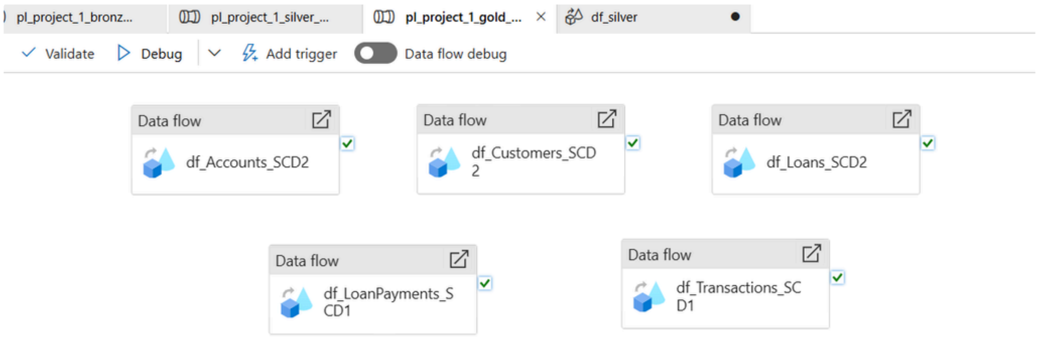
Summary of Cleaning and Transformation Objectives

- Standardizes column names and formats
- Removes incomplete or invalid records
- Adds derived columns (like full customer names)
- Aggregates and filters based on business logic
- Prepares clean, consistent data for reporting or analytical purposes

STEP 4 - Create the pipeline for Gold layer - Child pipeline 3

The `pl_project_1_gold_ADLS_SQL` pipeline handles business-ready data modeling and historical tracking. It applies SCD Type 1 and Type 2 logic to maintain either the latest state or full history of changes, depending on the data type. The output is written to Azure SQL Database, where the data is analytics-ready, supporting dashboards, reports, and insights.

Purpose:
This pipeline executes five dataflows that load and transform data from the bronze/silver layer into the gold layer (Azure SQL Database) using Slowly Changing Dimension (SCD) logic.



Data flow: df_Accounts_SCD2

Data flow: df_Customers_SCD2

Data flow: df_Loans_SCD2

Data flow: df_LoanPayments_SCD1

Data flow: df_Transactions_SCD1

Pipeline Name: `pl_project_1_gold_ADLS_SQL`

Activity 1: `df_Accounts_SCD2`

- **Type**: ExecuteDataFlow
- **Dataflow Reference**: `df_accounts_SCD2`
- **Purpose**:

  Loads and processes account data using **SCD Type 2 logic** to capture historical changes (e.g., keeping old versions of changed records).
- **Notes**:

  This transformation maintains a history of changes using start and end dates, and ensures the new record is marked as current.

Activity 2: `df_Customers_SCD2_`

- **Type**: ExecuteDataFlow
- **Dataflow Reference**: `df_customers_SCD2`
- **Purpose**:

  Applies **SCD Type 2 logic** to customer records to track history over time.
- **Notes**:

  Ensures the gold layer for customer data retains historical versions of any changes in personal details or status.

Activity 3: `df_Loans_SCD2`

- **Type**: ExecuteDataFlow
- **Dataflow Reference**: `df_loans_SCD2`
- **Purpose**:

  Uses **SCD Type 2 logic** to capture changes in loan information, such as interest rate changes, terms, or status.
- **Notes**:

  Maintains a complete loan history, supporting regulatory reporting or historical trend analysis.

Activity 4: `df_LoanPayments_SCD1`

- **Type**: ExecuteDataFlow
- **Dataflow Reference**: `df_loanpayments_SCD1`
- **Purpose**:

  Loads loan payment records using **SCD Type 1 logic**, which updates existing records without preserving historical data.
- **Notes**:

  Since payment records are typically transactional and not expected to maintain history, SCD1 is suitable here.

Activity 5: `df_Transactions_SCD1`

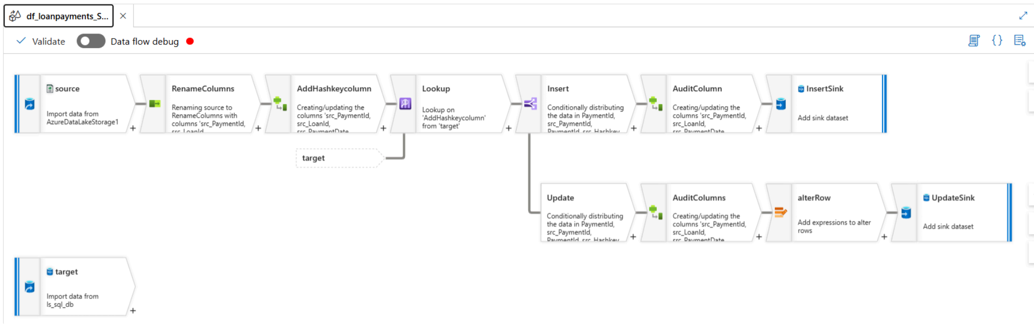
- **Type**: ExecuteDataFlow
- **Dataflow Reference**: `df_transactions_SCD1`
- **Purpose**:

  Applies **SCD Type 1 logic** for transaction data—overwriting old values with the latest ones.
- **Notes**:

  Focused on keeping the transaction table current; no need to track historical changes.

Implementation of SCD type 1 logic for LoanPayments

Here's a step-by-step explanation of implementing SCD Type 1 logic using the `df_loanpayments_SCD1` dataflow.

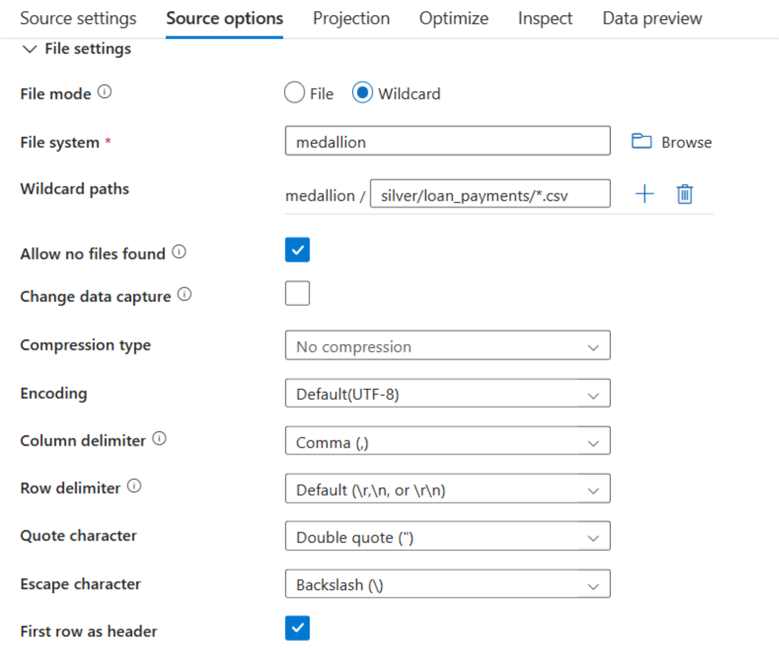

df_loanpayments_S...
Validate
Data flow debug
source
Import data from AzureDataLakeStorage1
RenameColumns
Renaming source to RenameColumns with columns 'src_PaymentId, src_LoanId,
AddHashkeycolumn
Creating/updating the columns 'src_PaymentId, src_LoanId, src_PaymentDate
Lookup
Lookup on 'AddHashkeycolumn' from 'target'
target
Insert
Conditionally distributing the data in PaymentId, src_PaymentId, PaymentId, src_Hashkey
AuditColumn
Creating/updating the columns 'src_PaymentId, src_LoanId, src_PaymentDate
InsertSink
Add sink dataset
Update
Conditionally distributing the data in PaymentId, src_PaymentId, PaymentId, src_Hashkey
AuditColumns
Creating/updating the columns 'src_PaymentId, src_LoanId, src_PaymentDate
alterRow
Add expressions to alter rows
UpdateSink
Add sink dataset
target
Import data from ls_sql_db

df_loanpayments_SCD1 - SCD type 1 logic on silver/loan_payments.csv

Step 4.1.1: Add Source Transformation – Source (ADLS Gen2)

Add a **Source** transformation and rename it to `source`.

- **Source Type**: Inline
- **Linked Service**: AzureDataLakeStorage1
- **File Path**: `silver/loan_payments/*.csv`
- **Dataset Format**: Delimited Text
- **First Row as Header**: Enabled



File settings

File mode ○ File ● Wildcard

File system * medallion

Wildcard paths medallion / silver/loan_payments/*.csv

Allow no files found ☑

Change data capture ☐

Compression type No compression

Encoding Default(UTF-8)

Column delimiter Comma (,)

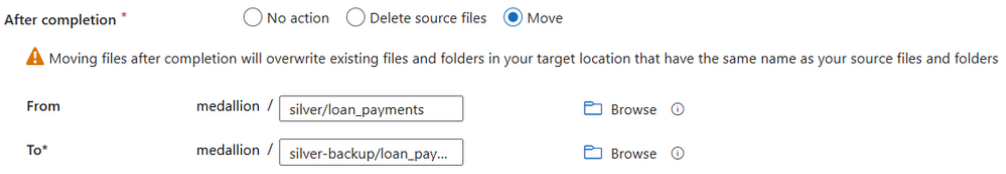
Row delimiter Default (\r,\n, or \r\n)

Quote character Double quote (")

Escape character Backslash (\)

First row as header ☑

Source options - loan_payments



After completion * ○ No action ○ Delete source files ● Move

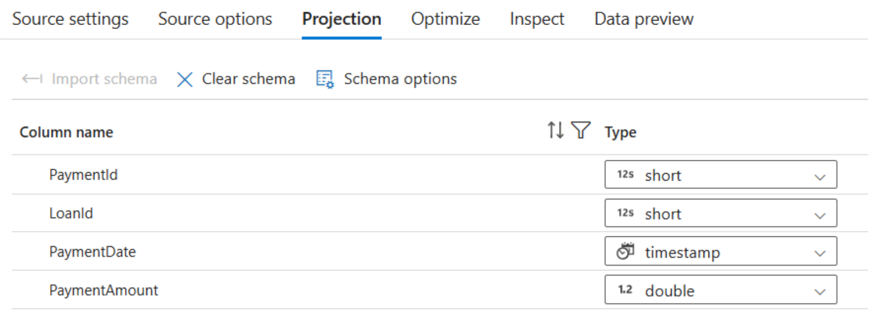
Moving files after completion will overwrite existing files and folders in your target location that have the same name as your source files and folders

From medallion / silver/loan_payments

To* medallion / silver-backup/loan_pay...

Source options - loan_payments

- **Import Schema**: Manually define columns:



Column name	Type
PaymentId	12S short
LoanId	12S short
PaymentDate	timestamp
PaymentAmount	1.2 double

Projection - - loan_payments

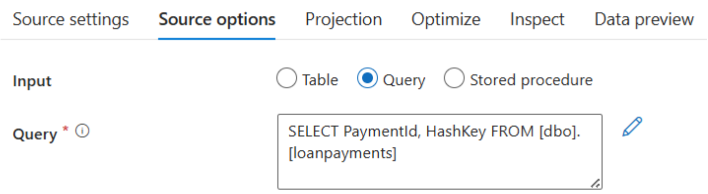
- **Options**:
  - Wildcard Path: Enabled

Step 4.1.2: Add Source Transformation – Target (Azure SQL Database)

Add another **Source** transformation and rename it to `target`.

- **Linked Service**: `ls_sql_db`
- **Source Type**: Inline
- **Dataset Format**: Query
- **Query**:

```
SELECT PaymentId, HashKey FROM [dbo].[loanpayments]
```

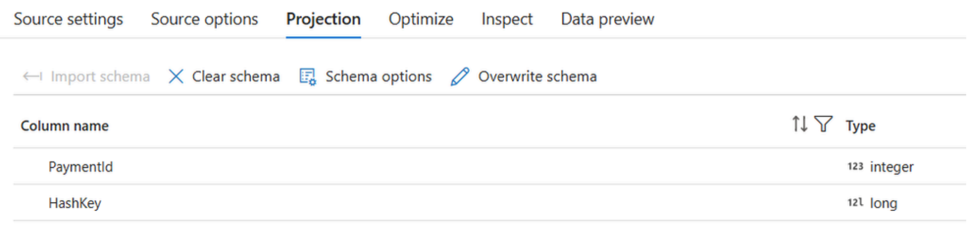


Input: ○ Table ● Query ○ Stored procedure

Query *: SELECT PaymentId, HashKey FROM [dbo].[loanpayments]

source options - loanpayments table

- **Projection**:



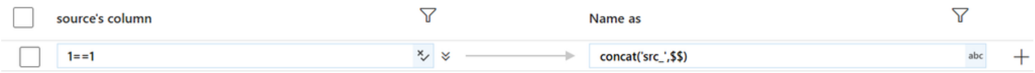
Column name	Type
PaymentId	123 integer
HashKey	12L long

  - projection - loanpayments table

Step 4.1.3: Rename Columns for Easy Differentiation

Add a **Select** transformation and rename it to `RenameColumns`.

- **Rule-based mapping**:
  - Prefix all columns from source with `src_`
  - Example:
    - `PaymentId` → `src_PaymentId`
    - `LoanId` → `src_LoanId`
    - `PaymentAmount` → `src_PaymentAmount`



□ source's column	Name as
□ 1==1	concat('src_',$$)

Select transformation for renaming the columns for source

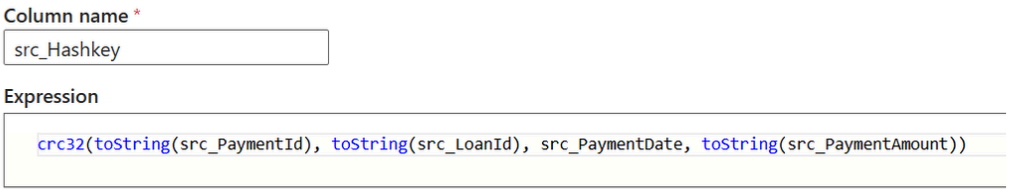
Step 4.1.4: Generate Hash Key to Detect Changes

Add a **Derived Column** transformation and rename it to `AddHashkeycolumn`.

- **Expression**:

```
src_Hashkey = crc32(toString(src_PaymentId), toString(src_LoanId), src_PaymentDate,
toString(src_PaymentAmount))
```

This generates a unique hash representing a row's current state.



Column name *
src_Hashkey

Expression
```
crc32(toString(src_PaymentId), toString(src_LoanId), src_PaymentDate, toString(src_PaymentAmount))
```

Derived column - generate src_hashkey for source

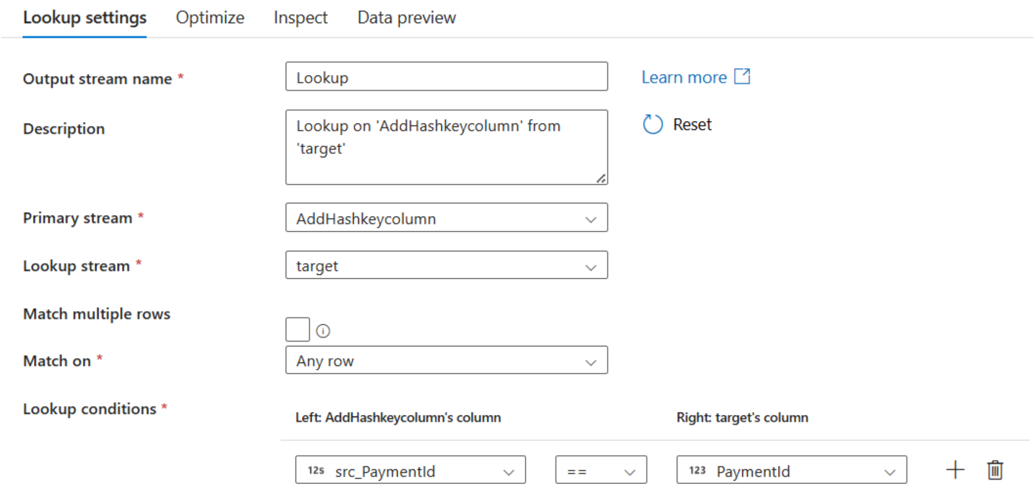
Step 4.1.5: Add Lookup to Compare with Target Table

Add a **Lookup** transformation and rename it to `Lookup`.

- **Left Stream**: `AddHashkeycolumn`
- **Right Stream**: `target`
- **Join Condition**:

```
src_PaymentId == PaymentId
```

Identifies if the source record already exists in the target.



Output stream name *	Lookup
Description	Lookup on 'AddHashkeycolumn' from 'target'
Primary stream *	AddHashkeycolumn
Lookup stream *	target
Match multiple rows	□
Match on *	Any row

Lookup conditions *

Left: AddHashkeycolumn's column		Right: target's column
src_PaymentId	==	PaymentId

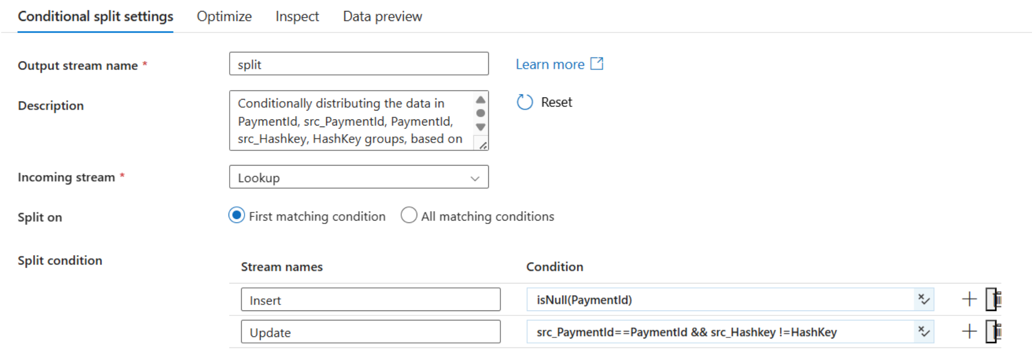
Lookup settings

Step 4.1.6: Conditional Split – Insert or Update

Add a **Conditional Split** transformation and rename it to `split`.

- **Conditions**:

- **Insert**: `isNull(PaymentId)`
- **Update**: `src_PaymentId == PaymentId && src_Hashkey != HashKey`



Output stream name *: split

Description: Conditionally distributing the data in PaymentId, src_PaymentId, PaymentId, src_Hashkey, HashKey groups, based on

Incoming stream *: Lookup

Split on: ● First matching condition ○ All matching conditions

Split condition

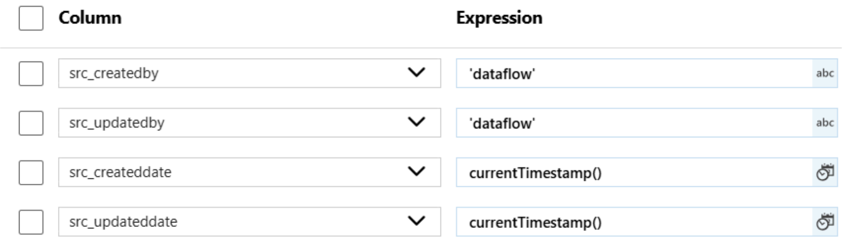
Stream names	Condition
Insert	isNull(PaymentId)
Update	src_PaymentId==PaymentId && src_Hashkey !=HashKey

Conditional split settings

Step 4.1.7: Handle Insert Records

On the **Insert** path from `split`:

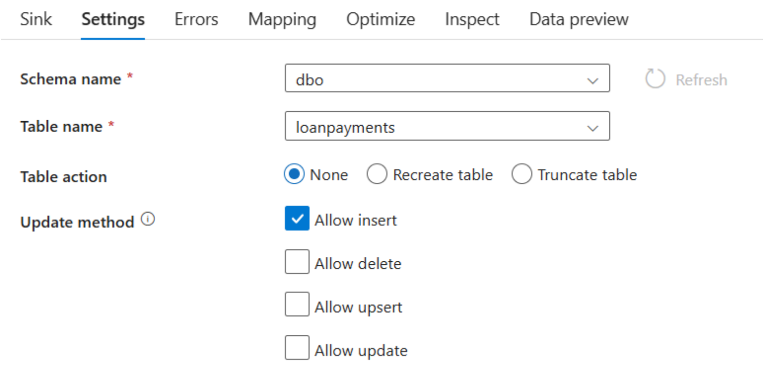
1. Add a **Derived Column** transformation and rename it to `AuditColumn`.
   - Add the following columns:
     - `src_createdby = 'dataflow'`
     - `src_updatedby = 'dataflow'`
     - `src_createddate = currentTimestamp()`
     - `src_updateddate = currentTimestamp()`



□ Column	Expression
□ src_createdby	'dataflow'
□ src_updatedby	'dataflow'
□ src_createddate	currentTimestamp()
□ src_updateddate	currentTimestamp()

Audit Columns for InsertSink

2. Add a **Sink** transformation and rename it to `InsertSink`.
   - **Linked Service**: `ls_sql_db`
   - **Table Name**: `dbo.loanpayments`
   - **Insertable**: true
   - **Mapping**:
     - Map input columns to target columns manually
     - Include audit columns and `Hashkey`



Sink | **Settings** | Errors | Mapping | Optimize | Inspect | Data preview

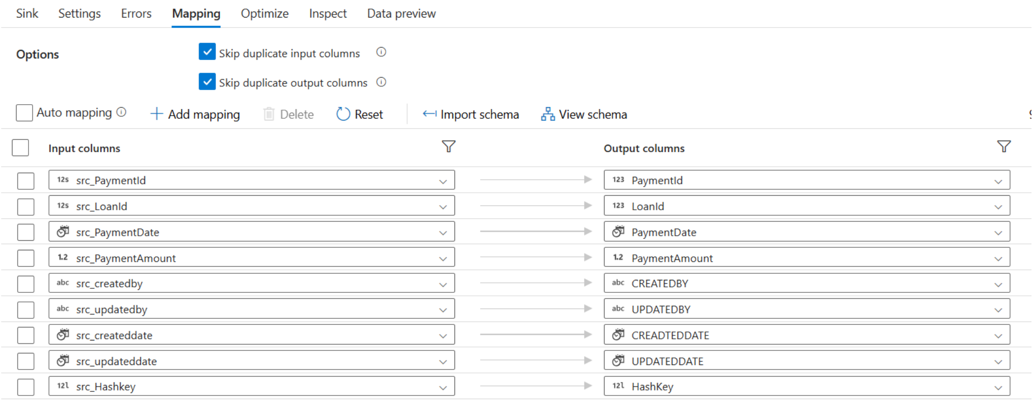
Schema name * dbo ⟳ Refresh

Table name * loanpayments

Table action ● None ○ Recreate table ○ Truncate table

Update method ⓘ
☑ Allow insert
☐ Allow delete
☐ Allow upsert
☐ Allow update

Sink Settings - InsertSink



Sink | Settings | Errors | **Mapping** | Optimize | Inspect | Data preview

Options ☑ Skip duplicate input columns ⓘ
☑ Skip duplicate output columns ⓘ

☐ Auto mapping ⓘ | + Add mapping | Delete | Reset | Import schema | View schema

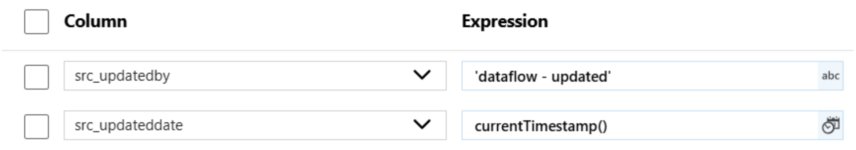
Input columns	Output columns
src_PaymentId	PaymentId
src_LoanId	LoanId
src_PaymentDate	PaymentDate
src_PaymentAmount	PaymentAmount
src_createdby	CREATEDBY
src_updatedby	UPDATEDBY
src_createddate	CREADTEDDATE
src_updateddate	UPDATEDDATE
src_Hashkey	HashKey

Mapping for InsertSink

Step 4.1.8: Handle Update Records

On the **Update** path from `split`:

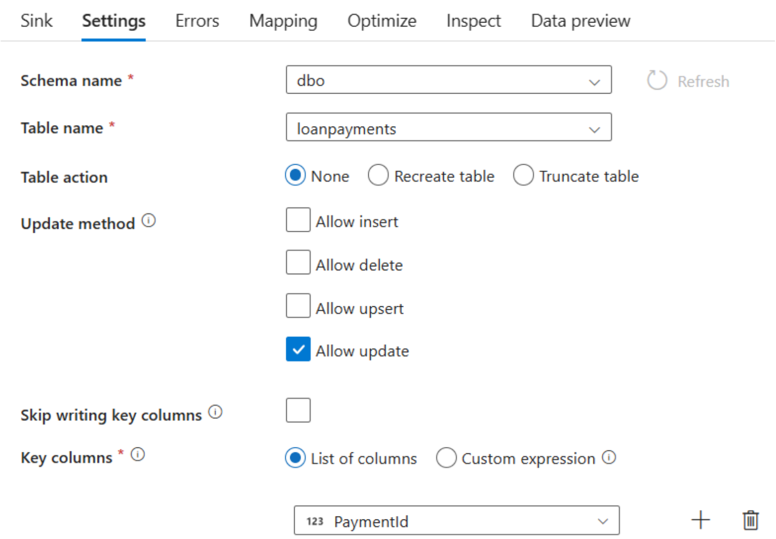
1. Add a **Derived Column** transformation and rename it to `AuditColumns`.
   - Add the following columns:
     - `src_updatedby = 'dataflow - updated'`
     - `src_updateddate = currentTimestamp()`



Column	Expression
src_updatedby	'dataflow - updated'
src_updateddate	currentTimestamp()

Audit Columns - Update Sink

2. Add an **Alter Row** transformation and rename it to `alterRow`.
   - **Update If** condition: 1 == 1
3. Add a **Sink** transformation and rename it to `UpdateSink`.
   - **Linked Service**: `ls_sql_db`
   - **Table Name**: `dbo.loanpayments`
   - **Updateable**: true
   - **Key Column**: `PaymentId`

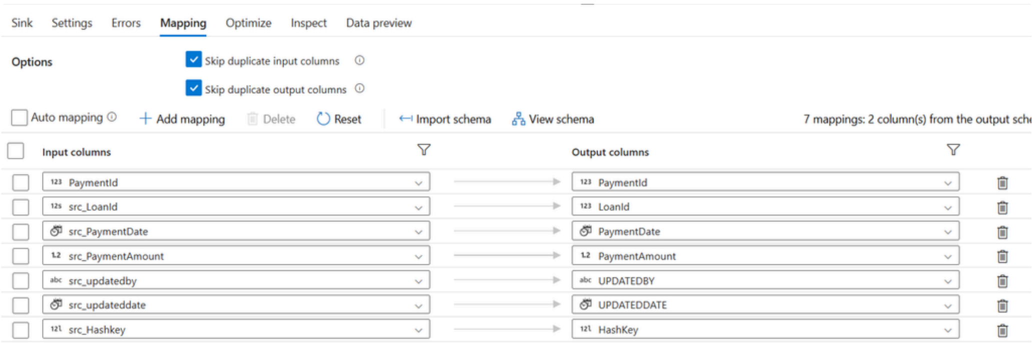


Sink | **Settings** | Errors | Mapping | Optimize | Inspect | Data preview

Schema name *	dbo ⌄ ⟳ Refresh
Table name *	loanpayments ⌄
Table action	● None ○ Recreate table ○ Truncate table
Update method ⓘ	□ Allow insert □ Allow delete □ Allow upsert ☑ Allow update
Skip writing key columns ⓘ	□
Key columns * ⓘ	● List of columns ○ Custom expression ⓘ
	123 PaymentId ⌄ + 🗑

- Sink Settings - UpdateSink

- **Mapping**:
  - Map input to target, exclude `createdby` and `createddate`
  - Include updated audit fields and `Hashkey`



Sink | Settings | Errors | **Mapping** | Optimize | Inspect | Data preview

Options: ☑ Skip duplicate input columns ⓘ ☑ Skip duplicate output columns ⓘ

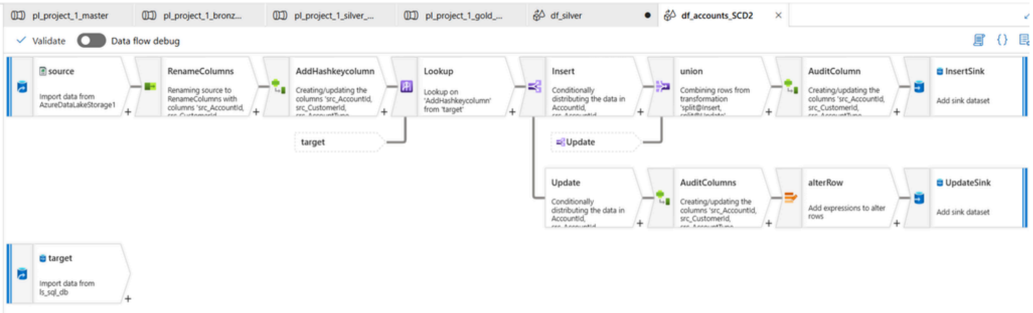
□ Auto mapping ⓘ | + Add mapping | Delete | Reset | Import schema | View schema | 7 mappings: 2 column(s) from the output sche

Input columns		Output columns
123 PaymentId	→	123 PaymentId
12s src_LoanId	→	123 LoanId
src_PaymentDate	→	PaymentDate
1.2 src_PaymentAmount	→	1.2 PaymentAmount
abc src_updatedby	→	abc UPDATEDBY
src_updateddate	→	UPDATEDDATE
121 src_Hashkey	→	121 HashKey

Mapping - UpdateSink

Implementation of SCD type 2 logic for Accounts

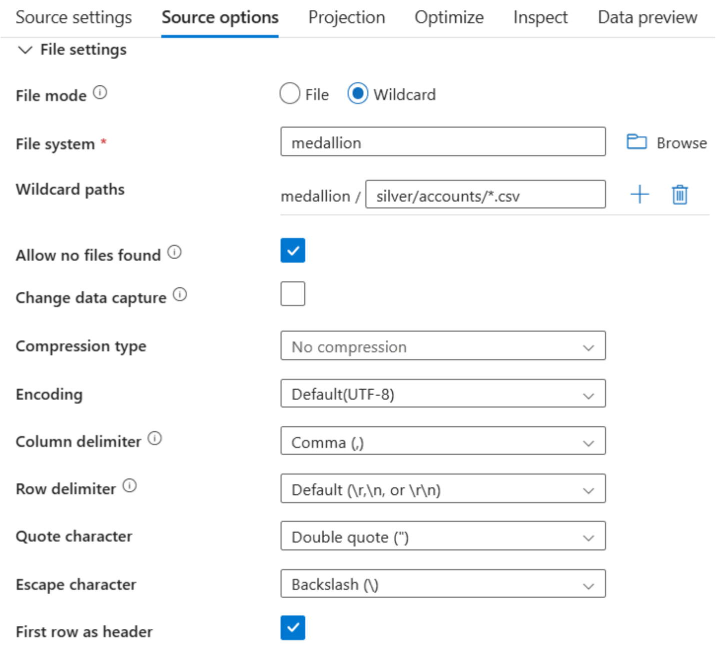
Here's a step-by-step explanation of implementing SCD Type 2 logic using the `df_accounts_SCD2` dataflow.


pl_project_1_master
pl_project_1_bronz...
pl_project_1_silver_...
pl_project_1_gold_...
df_silver
df_accounts_SCD2
Validate
Data flow debug
source
Import data from AzureDataLakeStorage1
RenameColumns
Renaming source to RenameColumns with columns 'src_AccountId,
AddHashkeycolumn
Creating/updating the columns 'src_AccountId, src_CustomerId,
Lookup
Lookup on 'AddHashkeycolumn' from 'target'
Insert
Conditionally distributing the data in AccountId,
union
Combining rows from transformation 'split@Insert,
AuditColumn
Creating/updating the columns 'src_AccountId, src_CustomerId,
InsertSink
Add sink dataset
target
Update
Update
Conditionally distributing the data in AccountId,
AuditColumns
Creating/updating the columns 'src_AccountId, src_CustomerId,
alterRow
Add expressions to alter rows
UpdateSink
Add sink dataset
target
Import data from ls_sql_db

df_accounts_SCD2 - SCD type 2 logic on silver/accounts.csv

Step 4.2.1: Add Source: Current Data from ADLS Gen2 (Silver Layer)

- Transformation Name: `source`
- Source Type: Delimited Text from Azure Data Lake Storage (`AzureDataLakeStorage1`)
- Path: `silver/accounts/*.csv`



File settings

File mode ○ File ● Wildcard

File system * medallion

Wildcard paths medallion / silver/accounts/*.csv

Allow no files found ☑

Change data capture □

Compression type: No compression

Encoding: Default(UTF-8)

Column delimiter: Comma (,)

Row delimiter: Default (\r,\n, or \r\n)

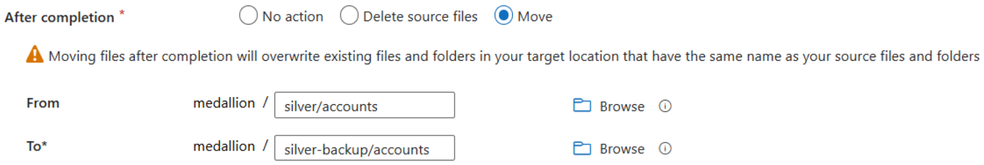
Quote character: Double quote (")

Escape character: Backslash (\)

First row as header ☑

Source options - accounts

- Options:
  - Header row enabled
  - File move after load: to `silver-backup/accounts`



After completion * ○ No action ○ Delete source files ● Move

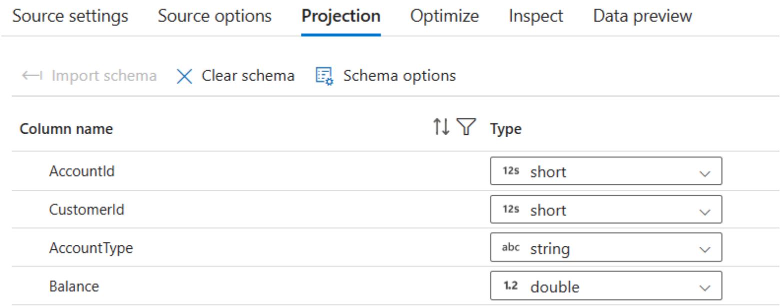
⚠ Moving files after completion will overwrite existing files and folders in your target location that have the same name as your source files and folders

From medallion / silver/accounts

To* medallion / silver-backup/accounts

  - Source options - accounts

- Projection: Import Schema Columns - `AccountId` , `CustomerId` , `AccountType` , `Balance`



Column name	Type
AccountId	12s short
CustomerId	12s short
AccountType	abc string
Balance	1.2 double

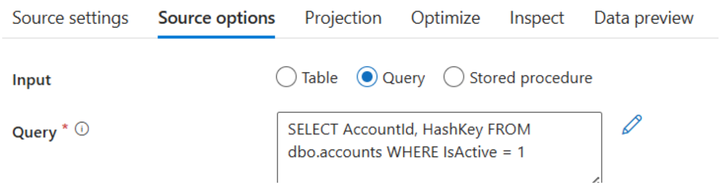
Import schema - accounts

Step 4.2.2: Add Source: Existing Records from Azure SQL Database

- Transformation Name: `target`
- Source Type: Azure SQL Database (`ls_sql_db`)
- Query:

```
SELECT AccountId, HashKey FROM dbo.accounts WHERE IsActive = 1
```

- Purpose: Loads current active records from the SQL target table for comparison.

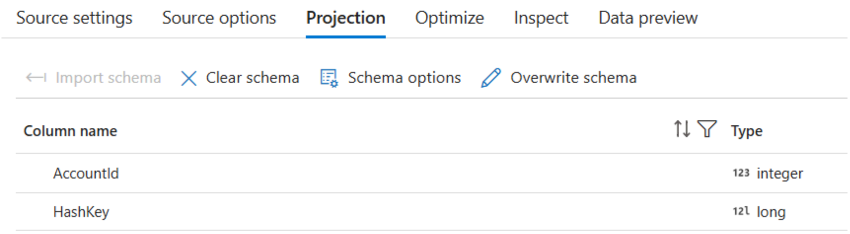


Input: ○ Table ● Query ○ Stored procedure

Query * ⓘ

```
SELECT AccountId, HashKey FROM
dbo.accounts WHERE IsActive = 1
```

Source options - accounts table

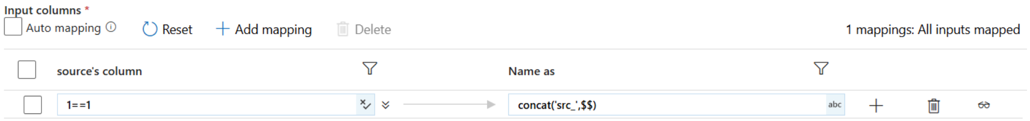


Column name	Type
AccountId	123 integer
HashKey	12L long

import schema - accounts table

Step 4.2.3: Rename Incoming Columns from ADLS

- Transformation Name: `RenameColumns`
- Rule-Based Mapping: Prefix all source columns with `src_` (e.g., `AccountId` becomes `src_AccountId`)



□	source's column	Name as
□	1==1	concat('src_',$$)

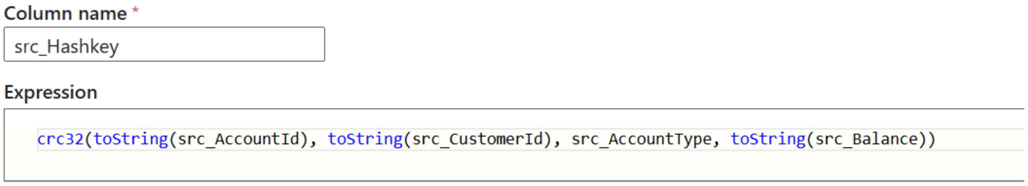
Select transformation - rename columns

Step 4.2.4: Generate HashKey for Change Detection

- Transformation Name: `AddHashkeycolumn`
- Derived Column: `src_Hashkey`
- Expression:

```
crc32(toString(src_AccountId), toString(src_CustomerId), src_AccountType, toString(src_Balance))
```

- Purpose: Creates a fingerprint of each record to detect changes.



Column name *

src_Hashkey

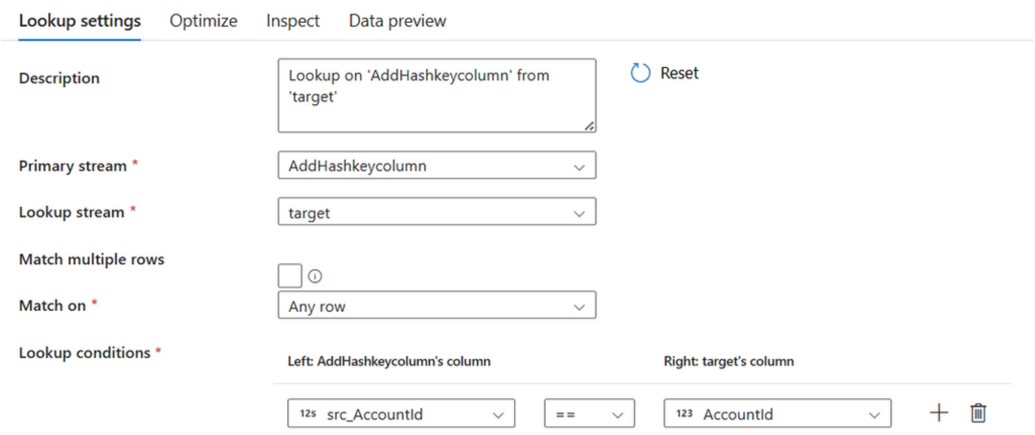
Expression

```
crc32(toString(src_AccountId), toString(src_CustomerId), src_AccountType, toString(src_Balance))
```

Creating src_Hashkey column

Step 4.2.5: Lookup Existing Records from SQL DB

- Transformation Name: `Lookup`
- Join Key: `src_AccountId == AccountId`
- Joins source data with the SQL data to find matches.
- Output includes matched `HashKey`.



Description: Lookup on 'AddHashkeycolumn' from 'target'

Primary stream *: AddHashkeycolumn

Lookup stream *: target

Match multiple rows: □

Match on *: Any row

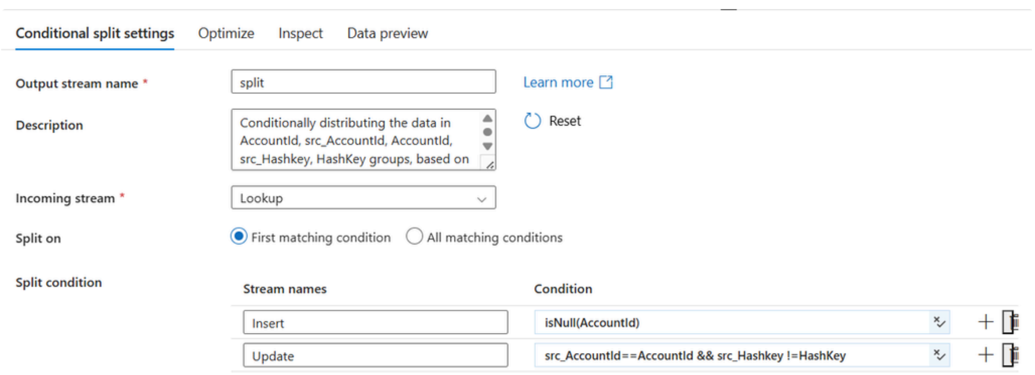
Lookup conditions *

Left: AddHashkeycolumn's column		Right: target's column
src_AccountId	==	AccountId

Lookup transformation - joining source and target

Step 4.2.6: Split Records into New and Updated

- Transformation Name: `split`
- Split Conditions:
  - **Insert (New)**: `isNull(AccountId)`
  - **Update (Changed)**: `src_AccountId == AccountId && src_Hashkey != HashKey`



Output stream name *: split

Description: Conditionally distributing the data in AccountId, src_AccountId, AccountId, src_Hashkey, HashKey groups, based on

Incoming stream *: Lookup

Split on: ● First matching condition ○ All matching conditions

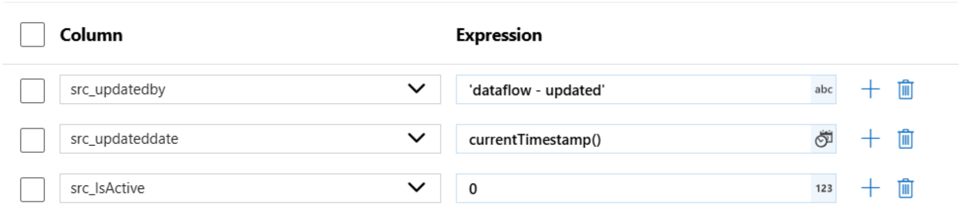
Split condition

Stream names	Condition
Insert	isNull(AccountId)
Update	src_AccountId==AccountId && src_Hashkey !=HashKey

Conditional split - insert and update conditions

Step 4.2.7: Handle Updated Records (SCD Type 2 Closure)

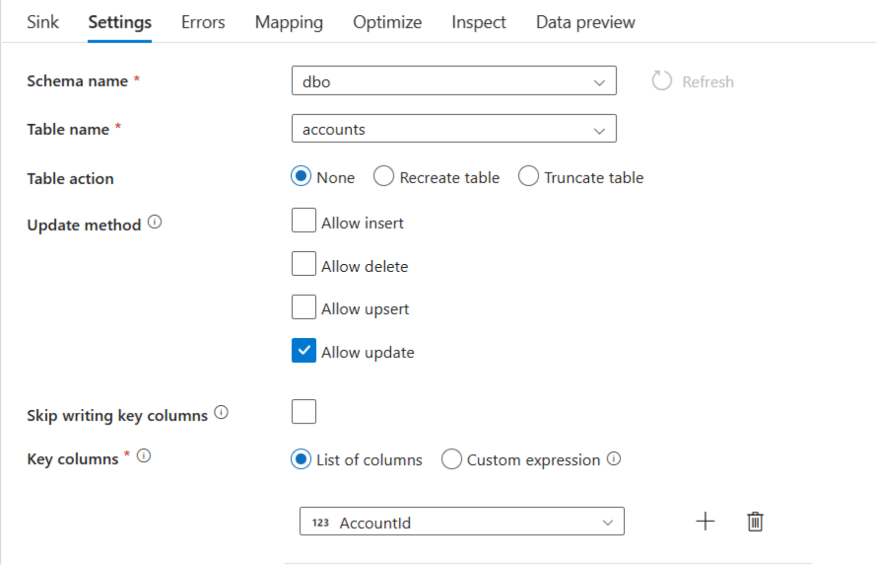
- From `split@Update`, add:
  - **Derived Columns** (`AuditColumns`):
    - `src_updatedby = 'dataflow - updated'`
    - `src_updateddate = currentTimestamp()`
    - `src_IsActive = 0` (mark old record inactive)



Column	Expression
src_updatedby	'dataflow - updated'
src_updateddate	currentTimestamp()
src_IsActive	0

    - AuditColumns for UpdateSink
  - **Alter Row**: `Upsert If (1 == 1)`
  - **Sink** (`UpdateSink`):
    - Type: Azure SQL Database (`ls_sql_db`)
    - Table: `dbo.accounts`
    - Method: **Update**

- Key Column: `AccountId`
- Mapping:
  - `UPDATEDBY = src_updatedby`
  - `UPDATEDDATE = src_updateddate`
  - `HashKey`
  - `IsActive = src_IsActive`



Schema name * dbo

Table name * accounts

Table action ● None ○ Recreate table ○ Truncate table

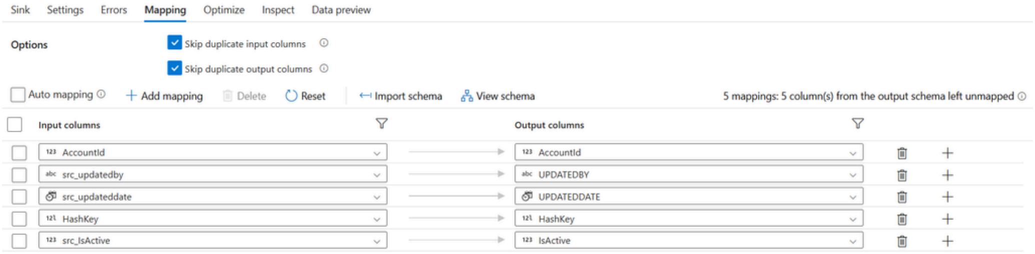
Update method
□ Allow insert
□ Allow delete
□ Allow upsert
☑ Allow update

Skip writing key columns □

Key columns * ● List of columns ○ Custom expression

123 AccountId

UpdateSink Settings



Options ☑ Skip duplicate input columns ☑ Skip duplicate output columns

□ Auto mapping

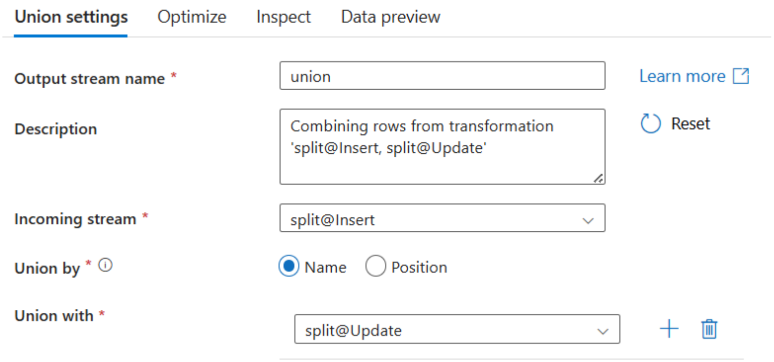
5 mappings: 5 column(s) from the output schema left unmapped

Input columns	Output columns
123 AccountId	123 AccountId
abc src_updatedby	abc UPDATEDBY
src_updateddate	UPDATEDDATE
12l HashKey	12l HashKey
123 src_IsActive	123 IsActive

UpdateSink Maaping

Step 4.2.8: Merge Paths for New Inserts

- **Union Transformation** (`union`) combines records from both `split@Insert` and `split@Update` .



Output stream name * union

Description Combining rows from transformation 'split@Insert, split@Update'

Incoming stream * split@Insert

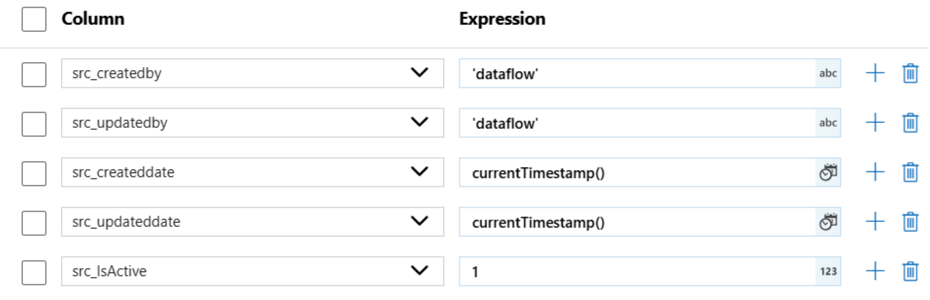
Union by * ● Name ○ Position

Union with * split@Update

Union Settings for Insert records

Step 4.2.9: Add Audit Columns for Inserted Records

- Transformation Name: `AuditColumn`
- Derived Columns:
  - `src_createdby = 'dataflow'`
  - `src_createddate = currentTimestamp()`
  - `src_updatedby = 'dataflow'`
  - `src_updateddate = currentTimestamp()`
  - `src_IsActive = 1`

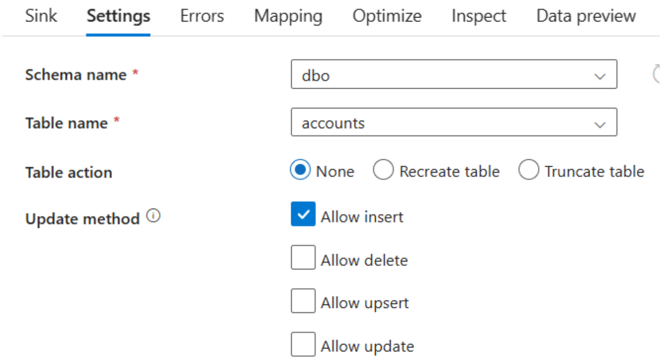


□ Column	Expression
□ src_createdby	'dataflow'
□ src_updatedby	'dataflow'
□ src_createddate	currentTimestamp()
□ src_updateddate	currentTimestamp()
□ src_IsActive	1

Audit Columns for Inserted Records

Step 4.2.10: Sink: Insert New Records into SQL

- Sink Name: `InsertSink`
- Type: Azure SQL Database (`ls_sql_db`)
- Table: `dbo.accounts`
- Method: **Insert**



Schema name * dbo

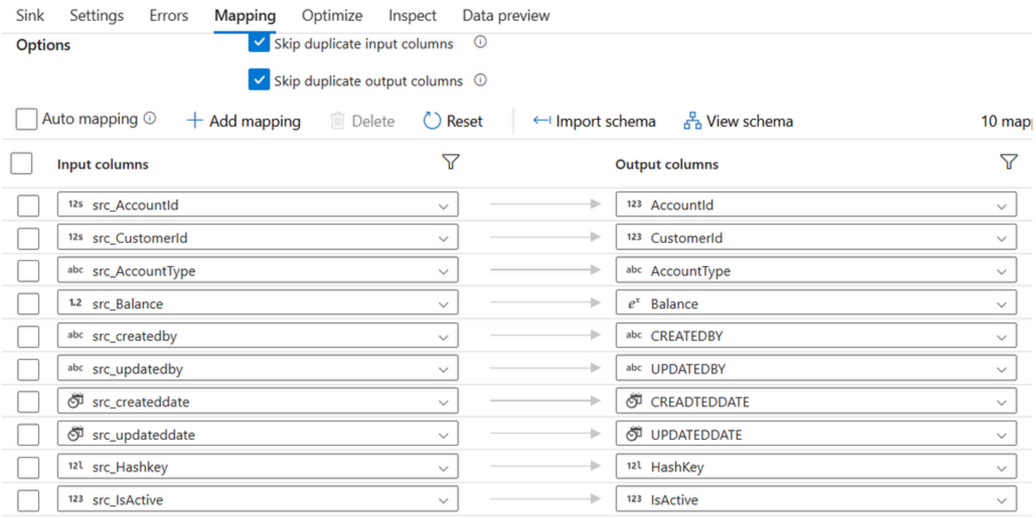
Table name * accounts

Table action ● None ○ Recreate table ○ Truncate table

Update method ☑ Allow insert
□ Allow delete
□ Allow upsert
□ Allow update

- Sink Settings for InsertSink

- Mapping:



Sink Settings Errors **Mapping** Optimize Inspect Data preview

Options

☑ Skip duplicate input columns

☑ Skip duplicate output columns

□ Auto mapping + Add mapping Delete Reset Import schema View schema 10 map

□	Input columns	Output columns
□	12s src_AccountId	123 AccountId
□	12s src_CustomerId	123 CustomerId
□	abc src_AccountType	abc AccountType
□	1.2 src_Balance	e^x Balance
□	abc src_createdby	abc CREATEDBY
□	abc src_updatedby	abc UPDATEDBY
□	src_createddate	CREADTEDDATE
□	src_updateddate	UPDATEDDATE
□	12l src_Hashkey	12l HashKey
□	123 src_IsActive	123 IsActive

Mapping for InsertSink

Summary

This pipeline consolidates and processes data from different domains:

- **SCD Type 2**: For Accounts, Customers, and Loans where historical tracking is necessary.
- **SCD Type 1**: For Loan Payments and Transactions where the latest state is sufficient.