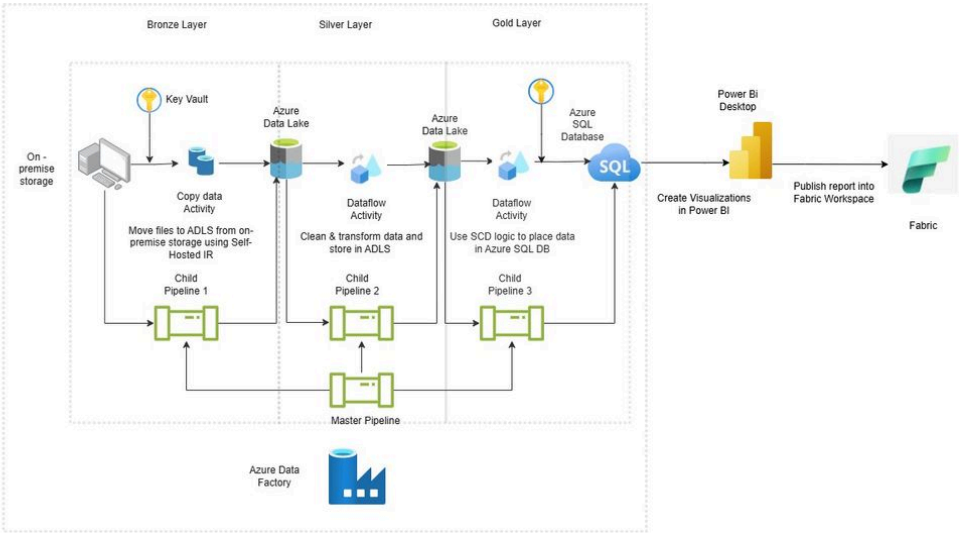
Project Implementation

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

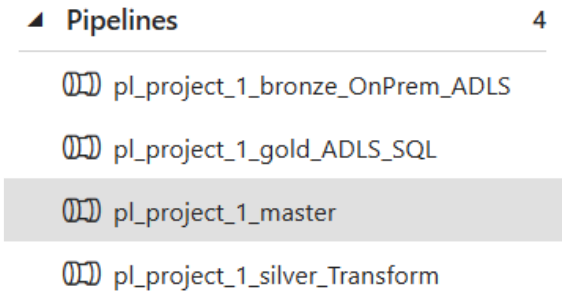
Security & Governance:

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

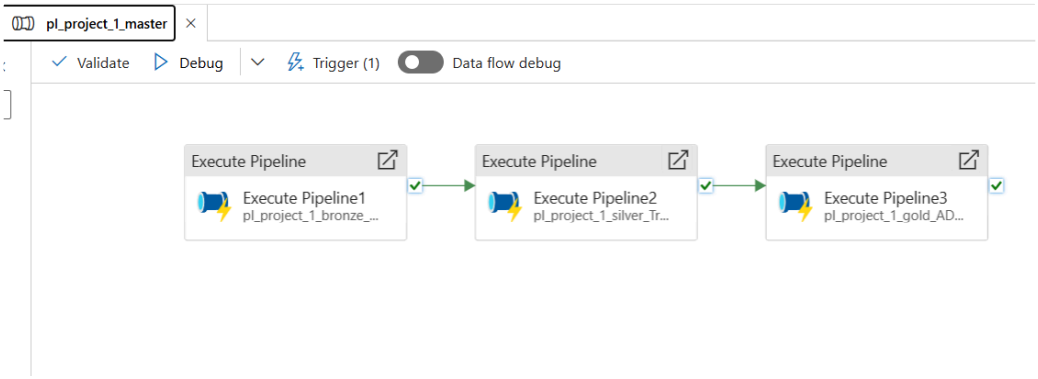
The pipeline `pl_project_1_master` orchestrates a three-stage data processing flow by executing three sub-pipelines in a sequential manner. Each stage is responsible for a different level of data transformation following a **Bronze** → **Silver** → **Gold** architecture. This modular approach ensures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



Pipelines 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

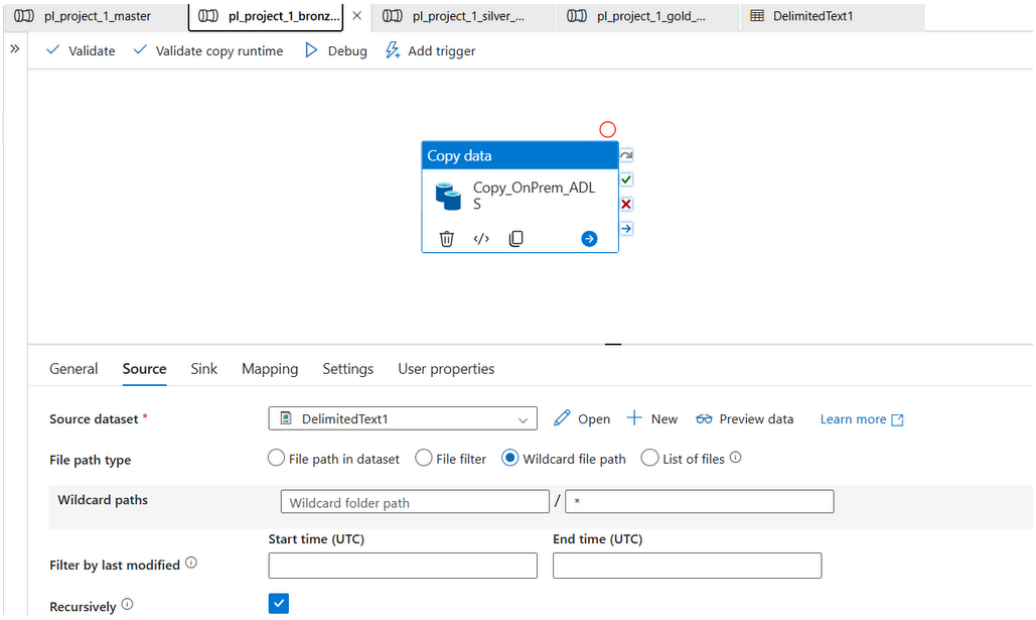
To ingest raw data files from an on-premise file system into the Bronze zone of ADLS Gen2 in CSV format using a robust and automated Copy activity.

Step 2.1: Copy Activity - OnPremise to ADLS Gen2

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

Purpose:

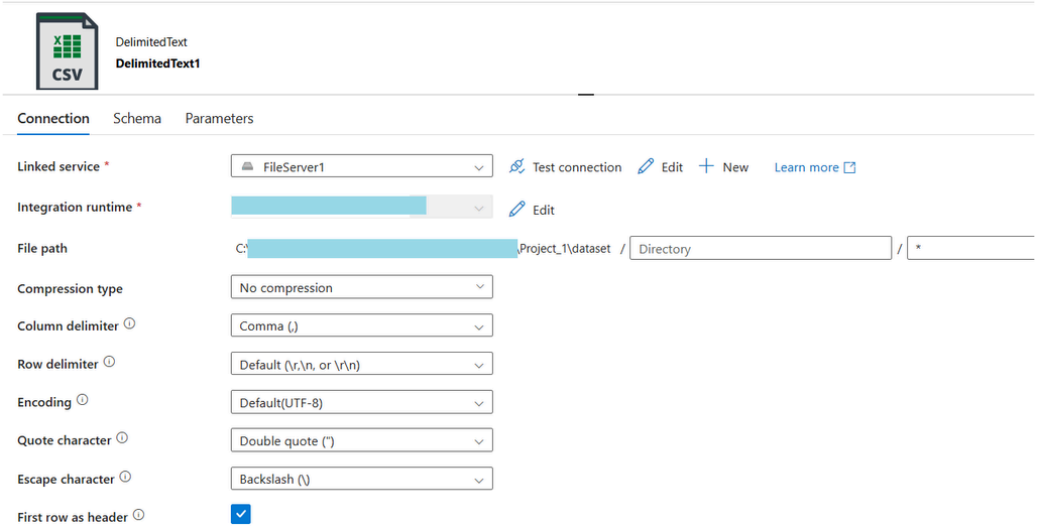
- Transfers data from an on-prem file system (likely mounted using Self-hosted Integration Runtime) to a cloud destination (ADLS Gen2).
- Reads delimited text (CSV) files recursively and writes them in a structured format into the Bronze container.



Copy Activity - Source Configuration

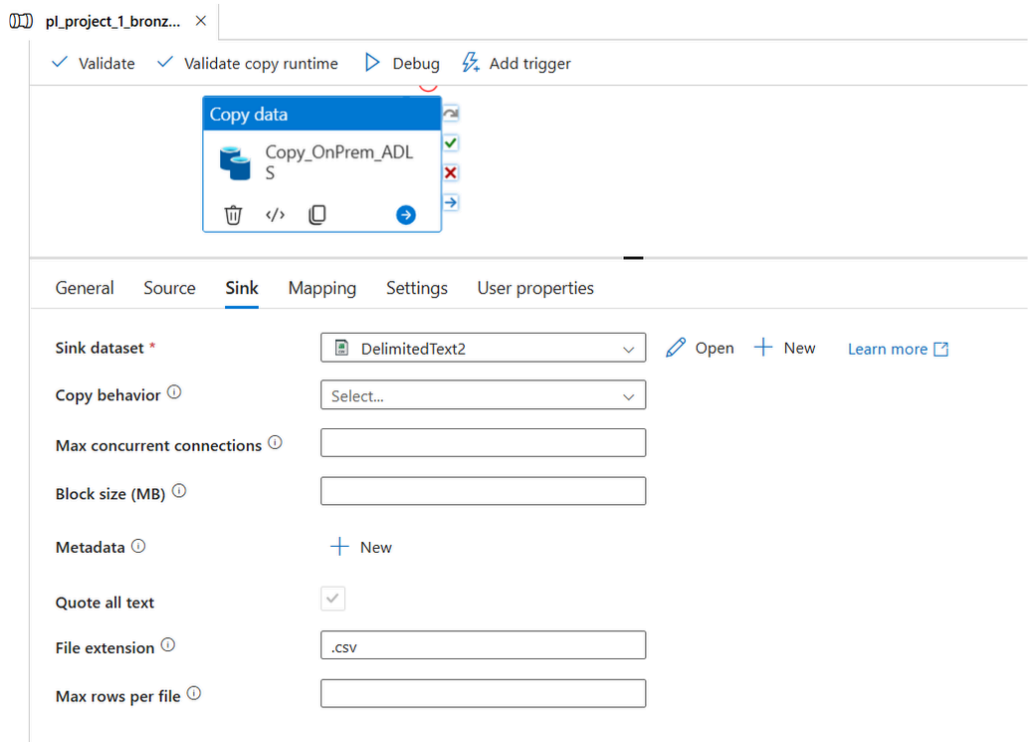
Source Configuration

- **Source Type**: `DelimitedTextSource`



Source Dataset

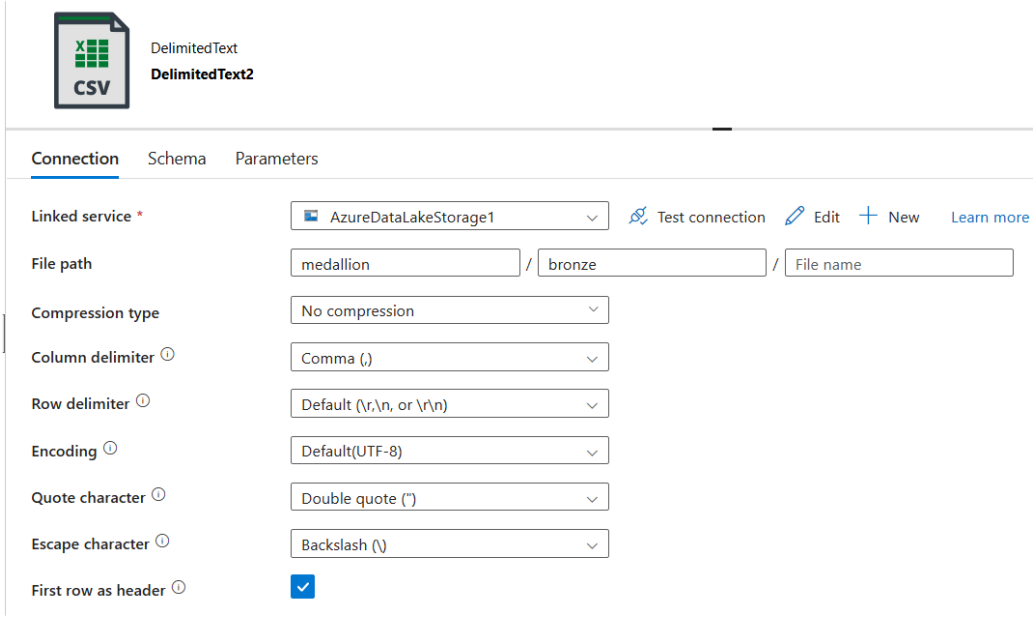
- **Store Settings**:
  - **Type**: `FileServerReadSettings`
  - **Recursive**: `true` (reads files from all subfolders)
  - **Wildcard FileName**: `*` (reads all file names)
  - **Partition Discovery**: Disabled
- **Format Settings**:
  - Reads files as delimited text (default CSV handling)



Copy Activity - Sink Configuration

Sink Configuration

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



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

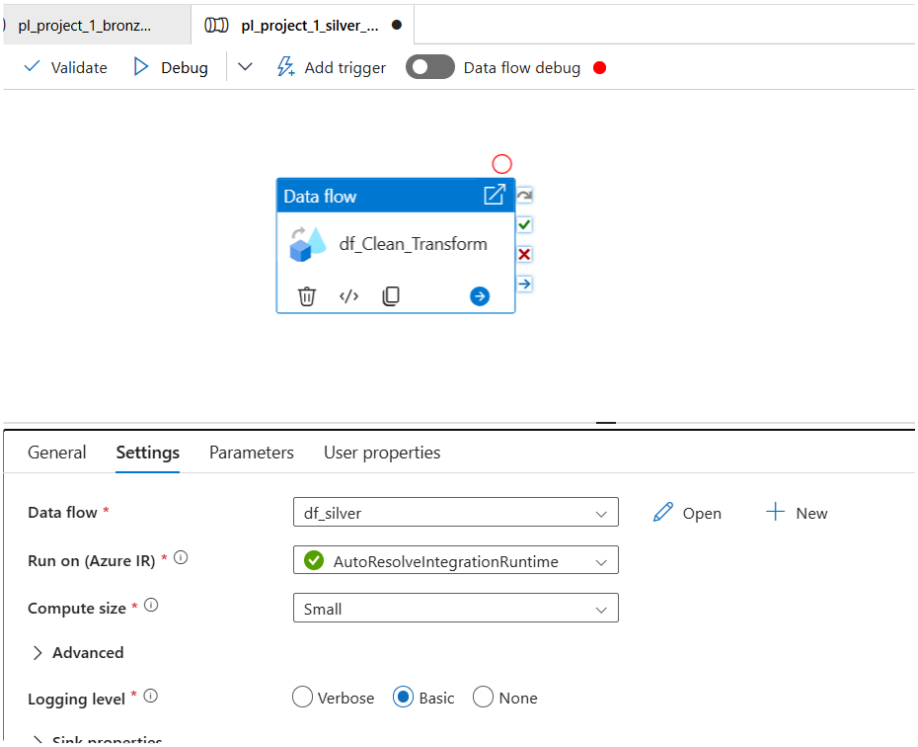
- **Recursive Read**: Ensures all relevant files across subfolders are ingested.
- **Wildcard File Name**: Makes ingestion dynamic and future-proof.
- **Format Consistency**: Quotes all fields and uses `.csv` extension.
- **Modular Design**: Can be reused or scaled to handle more file types or locations.

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`

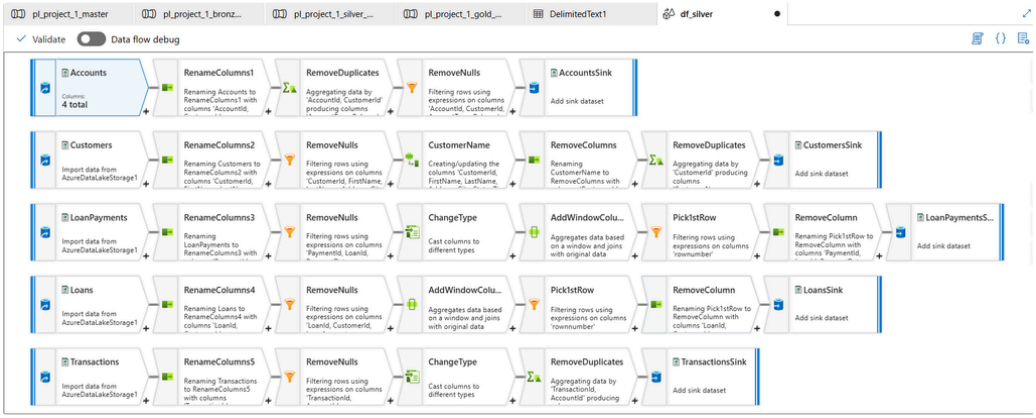


Silver layer pipeline

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

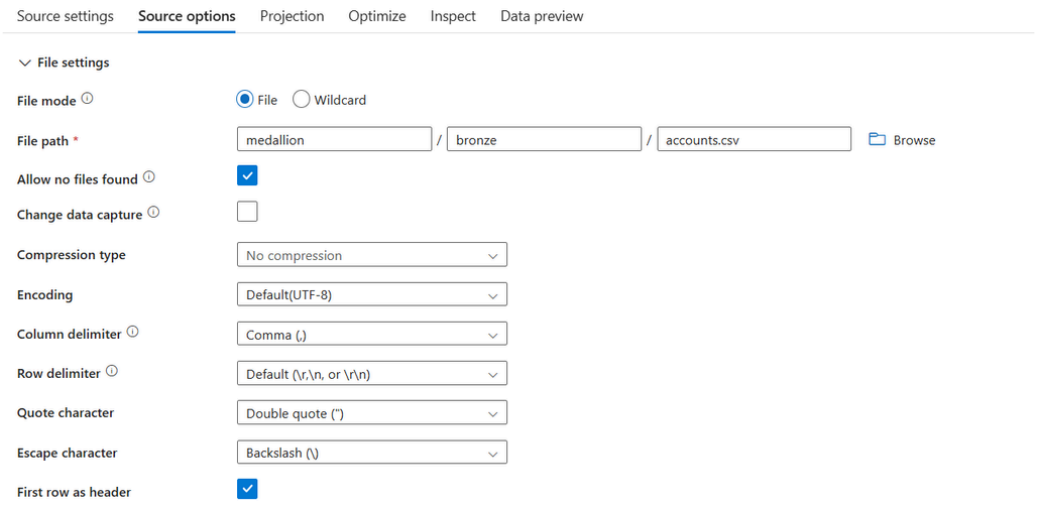


df_silver Dataflow

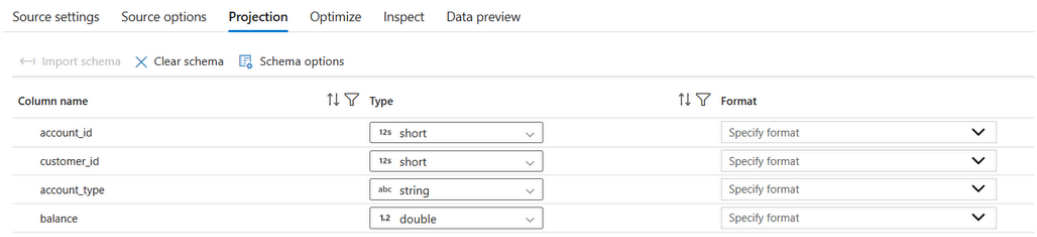
Step 3.1: Sources (Bronze Layer - ADLS Gen2)

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.csv	transaction_id, account_id, transaction_date, transaction_amount, transaction_type	Individual transaction records for accounts



accounts.csv source for df_silver



Key Columns of account.csv

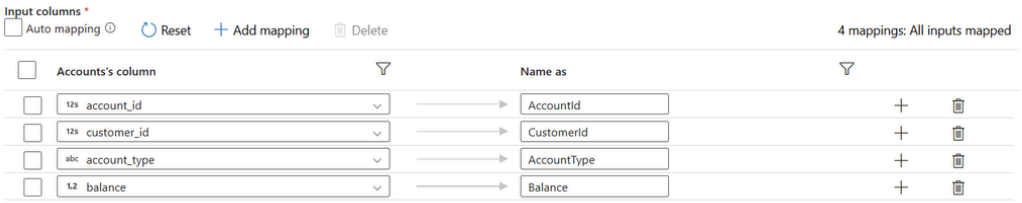
Transformations Applied

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.



Select transformation - RenameColumns1 for Accounts source

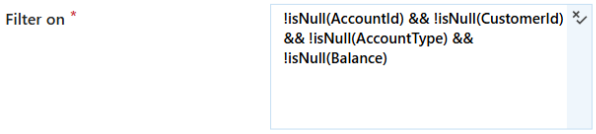
Step 3.3: Filtering Transformations (RemoveNulls1 to RemoveNulls5)

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 Transformation - Remove Nulls in Account details

Step 3.4: Derived Column: `CustomerName`

Creates a new column:

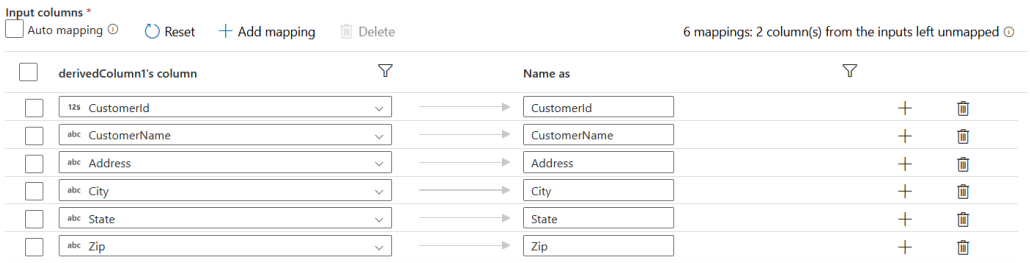
```
CustomerName = concat(FirstName, ' ', LastName)
```

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

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.



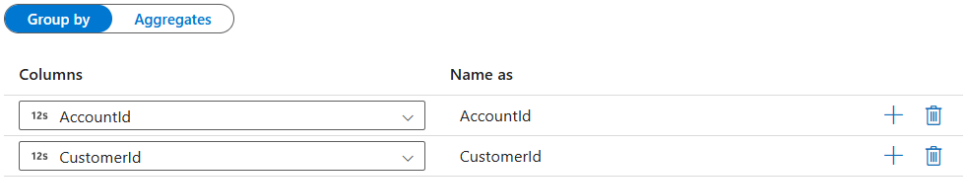
Select transformation - Removes unwanted columns and rearranges the order of the columns

Step 3.6: Aggregates (RemoveDuplicates1 to RemoveDuplicates3)

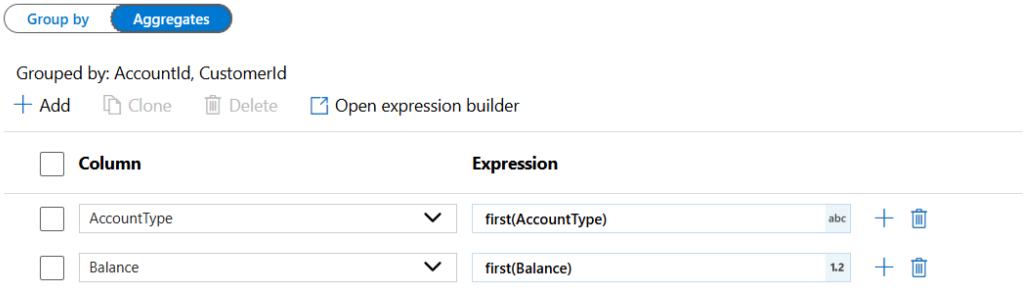
Aggregations group data for summarization and deduplication.

Examples:

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



Aggregate transformation - Group by IDs for Accounts source

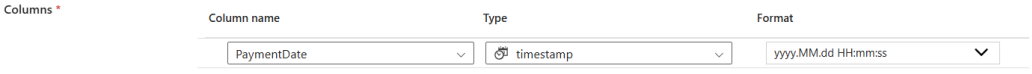


Aggregate transformation - get the first row only from the list of duplicates for Accounts source

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.



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

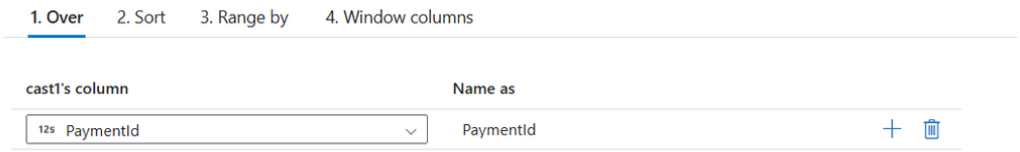
Step 3.8: Window Functions (AddWindowColumn1, AddWindowColumn2)

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

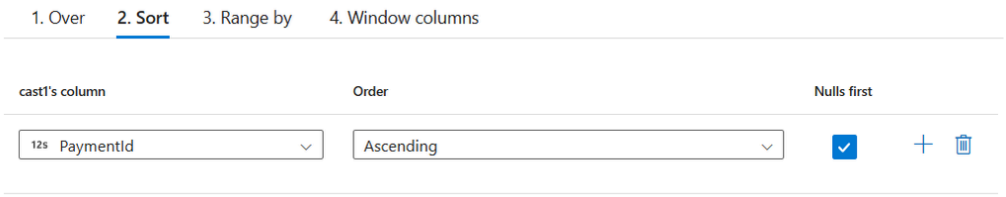
Example:

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

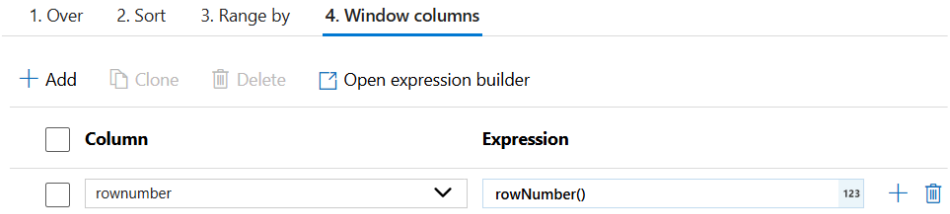
This window function is used to group rows by the ID column and assign row numbers for all the rows in that particular group. This is useful when you want to remove duplicates by just selecting the first row.



Window Transformation - over PaymentId column for LoanPayments source



Window Transformation - Sort PaymentIds in Ascending order for LoanPayments source



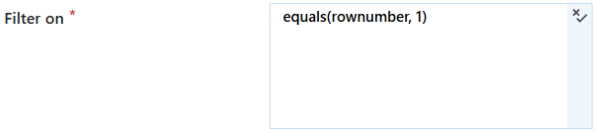
Window Transformation - get the row number for the corresponding PaymentId group for LoanPayments source

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

These filters remove unwanted rows after window logic.

Use cases:

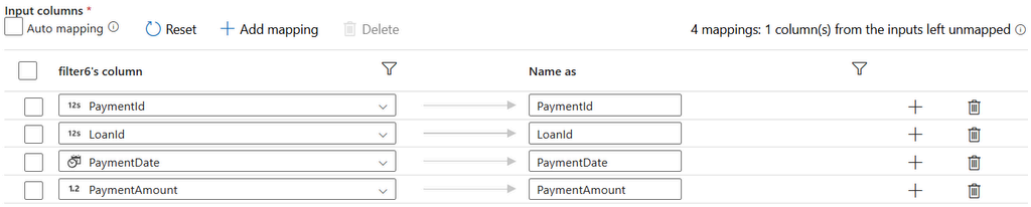
- Retain only the **most recent payment**
- Keep only the **latest transaction per account**



Filter 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.

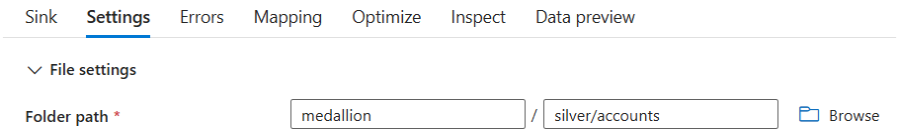


Select transformation - remove unwanted columns for LoanPayments source

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 for Accounts source

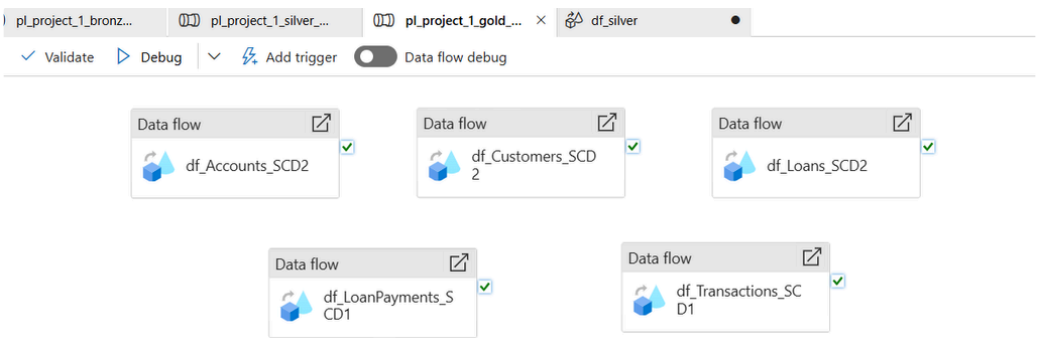
Summary of Cleaning and Transformation Objectives

- 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.



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`

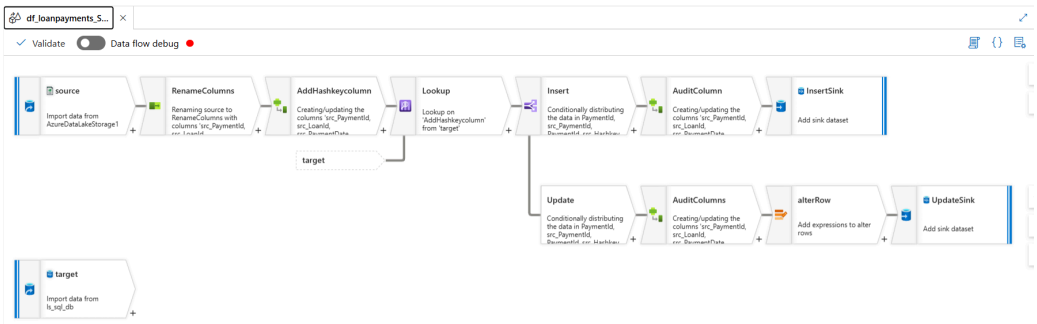
- **Type**: ExecuteDataFlow
- **Dataflow Reference**: `df_loanpayments_SCD1`
- **Purpose**:
  Loads loan payment records using **SCD Type 1 logic**, which updates existing records without preserving historical data.
- **Notes**:
  Since payment records are typically transactional and not expected to 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.

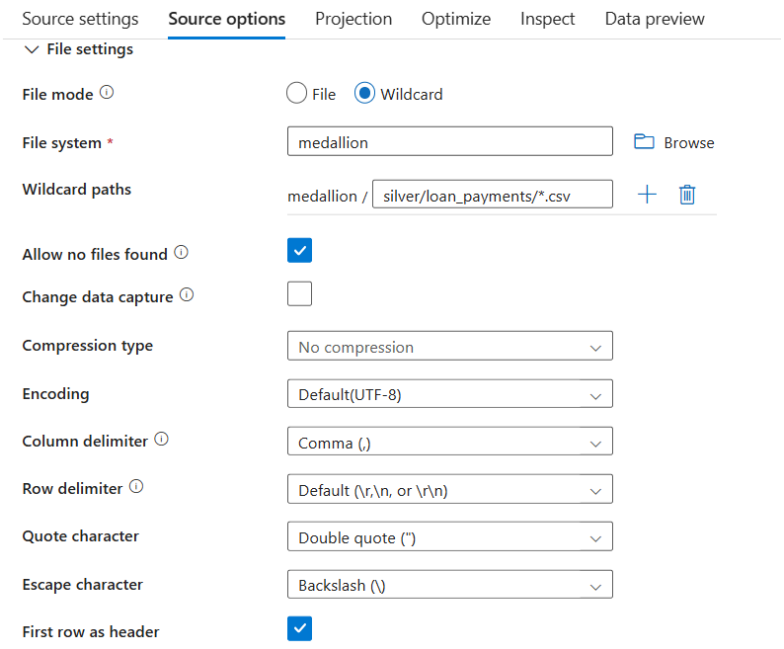


df_loanpayments_SCD1 - SCD type 1 logic on silver/loan_payments.csv

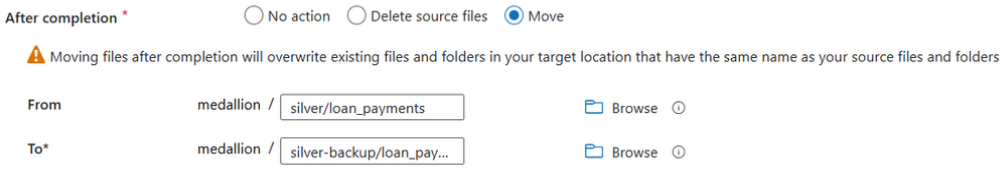
Step 4.1.1: Add Source Transformation – Source (ADLS Gen2)

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

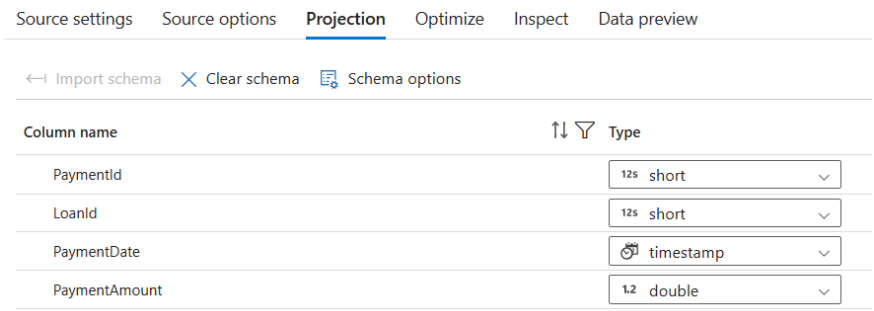


Source options - loan_payments



Source options - loan_payments

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



Projection - - loan_payments

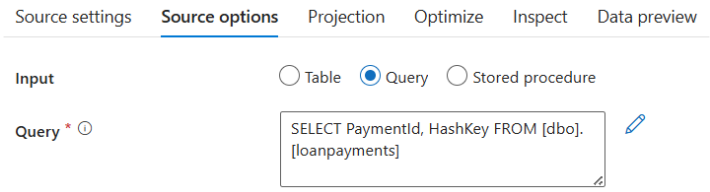
- **Options**:
  - Wildcard Path: Enabled

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

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

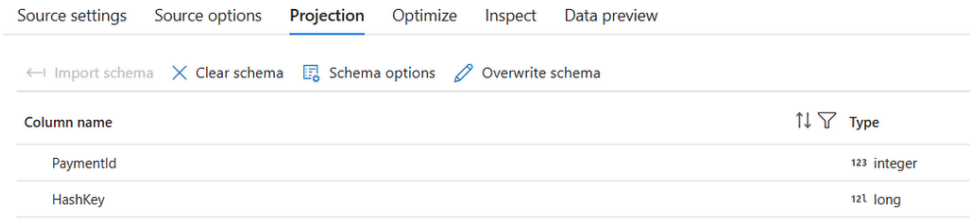
- **Linked Service**: `ls_sql_db`
- **Source Type**: Inline
- **Dataset Format**: Query
- **Query**:

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



source options - loanpayments table

- **Projection**:



  - 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`

Select transformation for renaming the columns for source

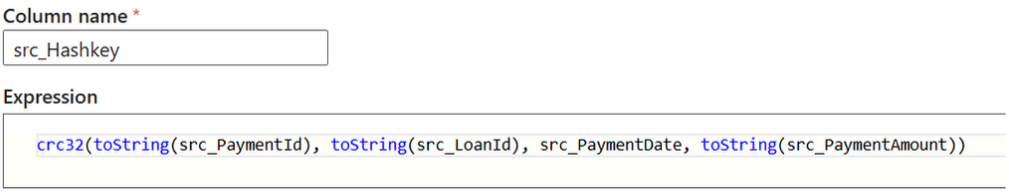
Step 4.1.4: Generate Hash Key to Detect Changes

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.



Derived column - generate src_hashkey for source

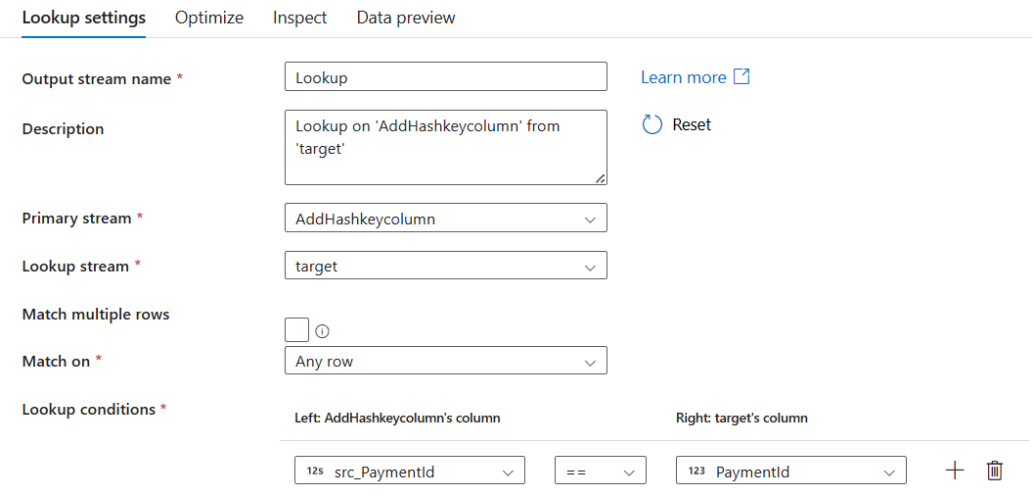
Step 4.1.5: Add Lookup to Compare with Target Table

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.



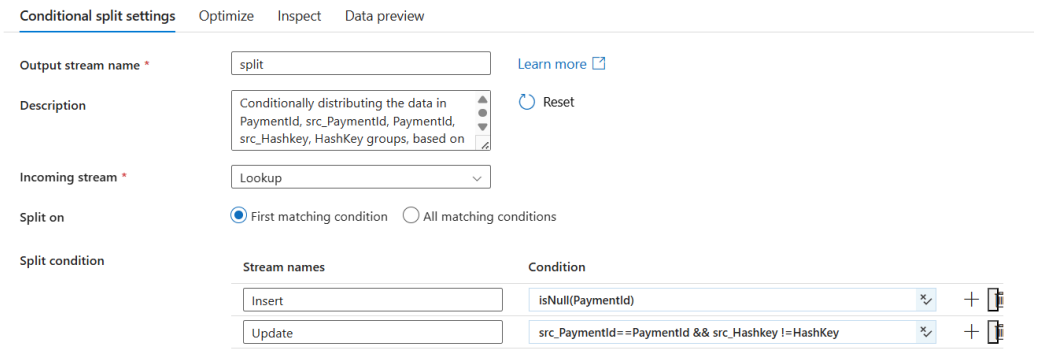
Lookup settings

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`

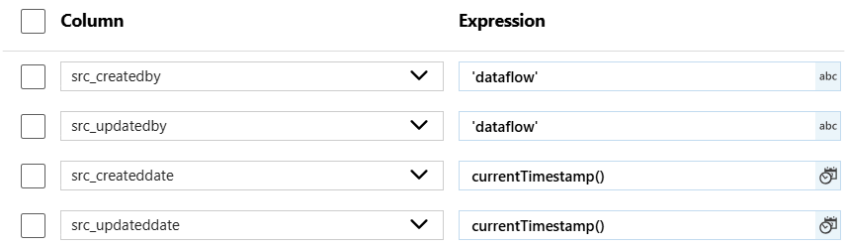


Conditional split settings

Step 4.1.7: Handle Insert Records

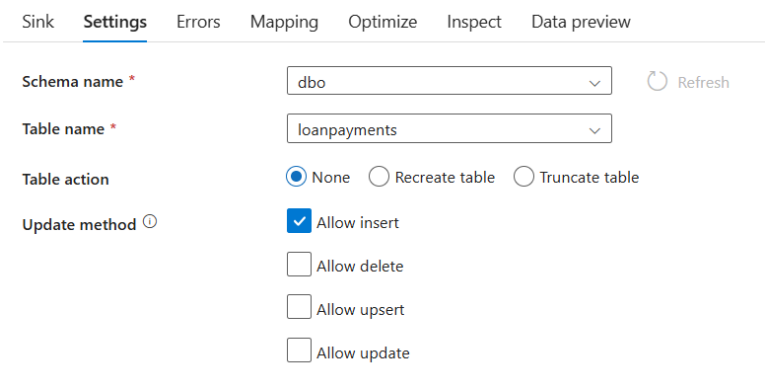
On the **Insert** path from `split`:

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()`

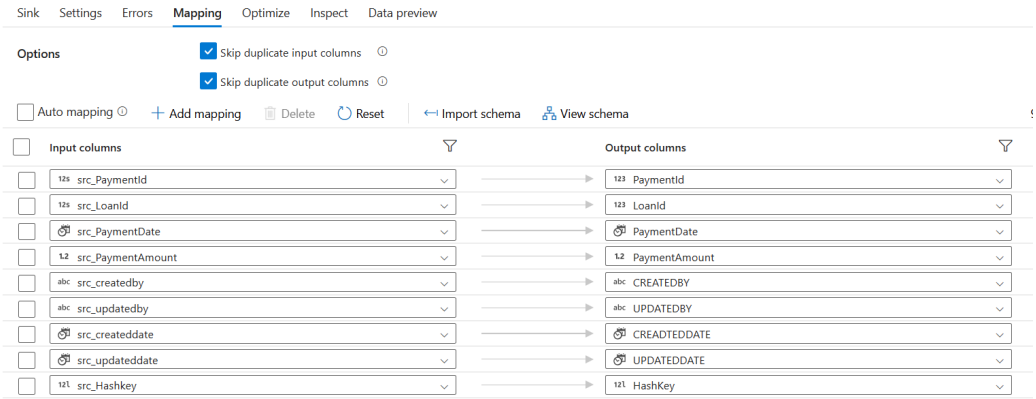


Audit Columns for InsertSink

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 - InsertSink

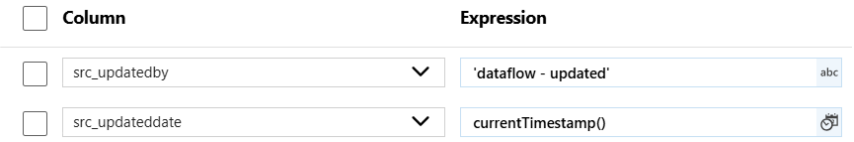


Mapping for InsertSink

Step 4.1.8: Handle Update Records

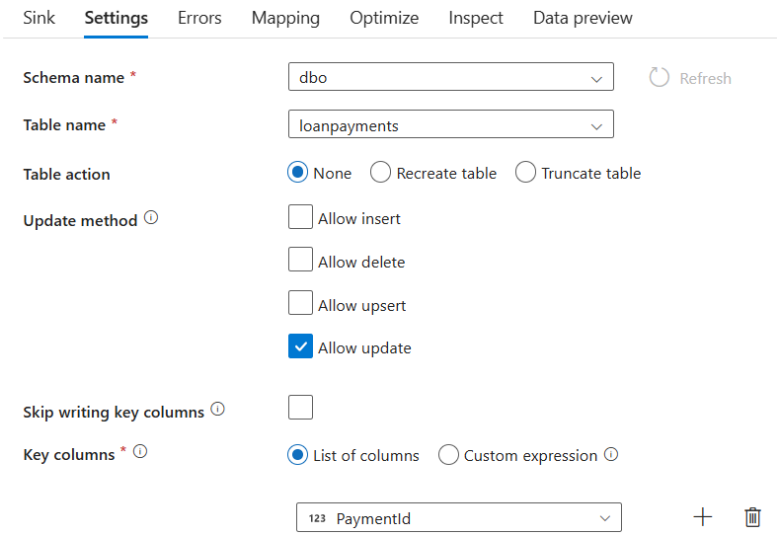
On the **Update** path from `split`:

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



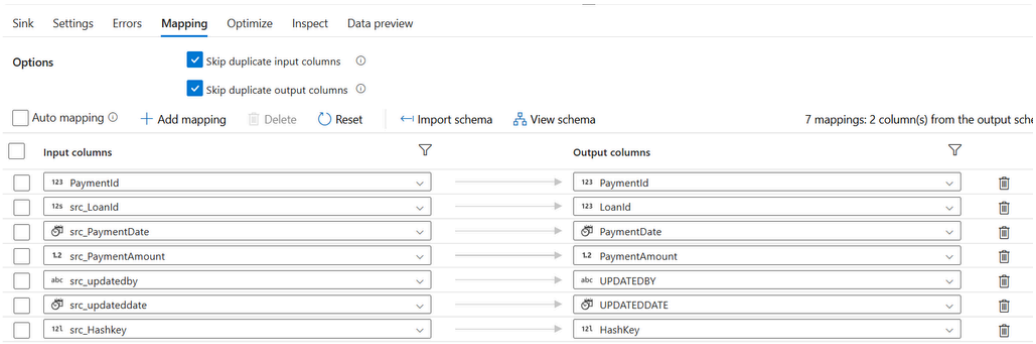
Audit Columns - Update Sink

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 - UpdateSink

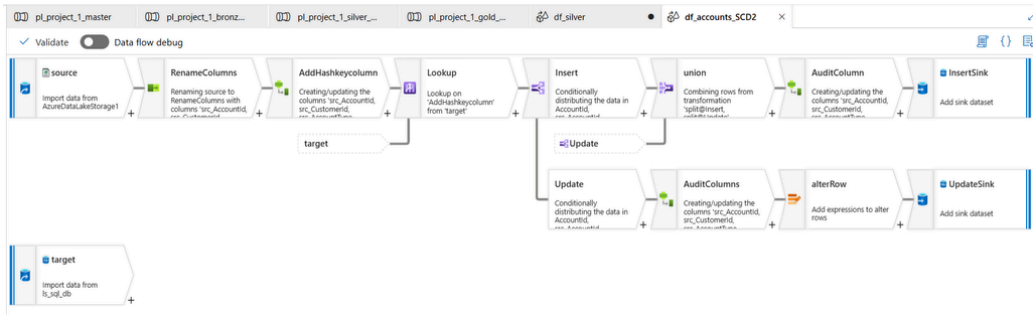
- **Mapping**:
  - Map input to target, exclude `createdby` and `createddate`
  - Include updated audit fields and `Hashkey`



Mapping - UpdateSink

Implementation of SCD type 2 logic for Accounts

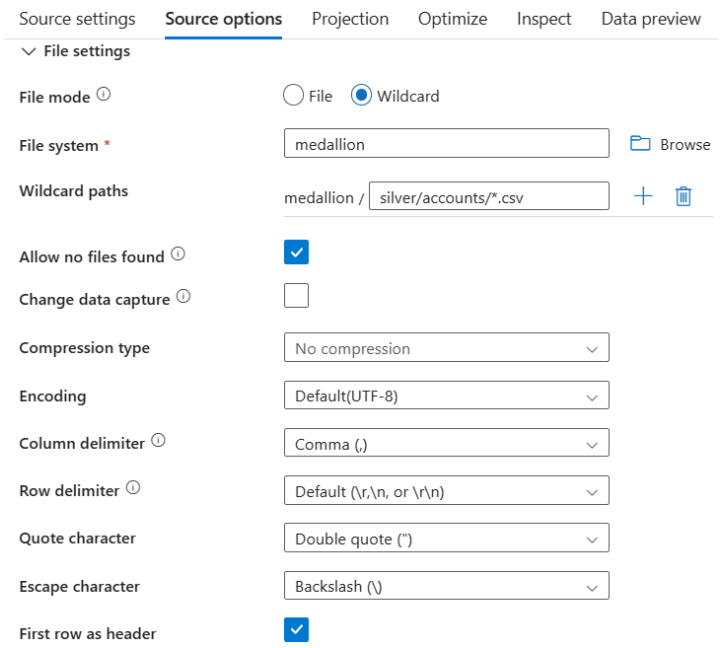
Here's a step-by-step explanation of implementing SCD Type 2 logic using the `df_accounts_SCD2` dataflow.



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

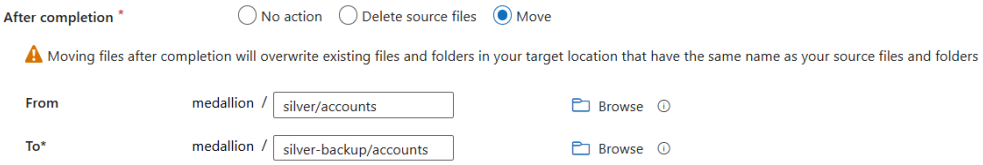
Step 4.2.1: Add Source: Current Data from ADLS Gen2 (Silver Layer)

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



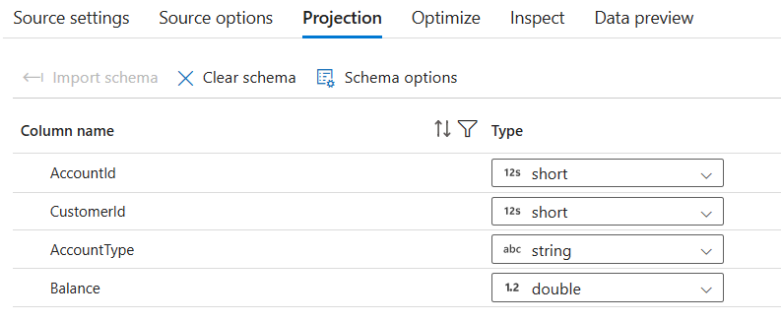
Source options - accounts

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



Source options - accounts

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



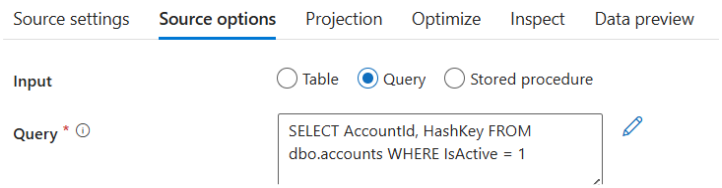
Import schema - accounts

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

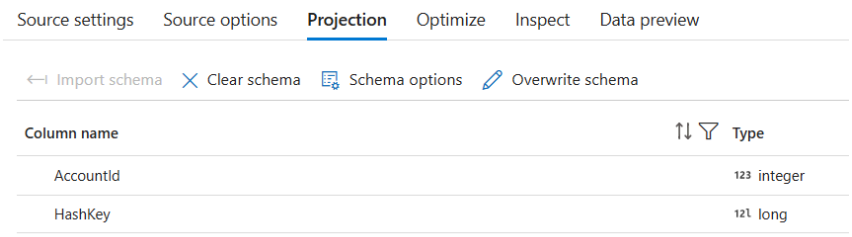
- Transformation Name: `target`
- Source Type: Azure SQL Database (`ls_sql_db`)
- Query:

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

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



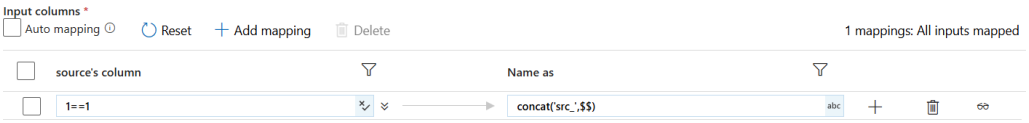
Source options - accounts table



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`)



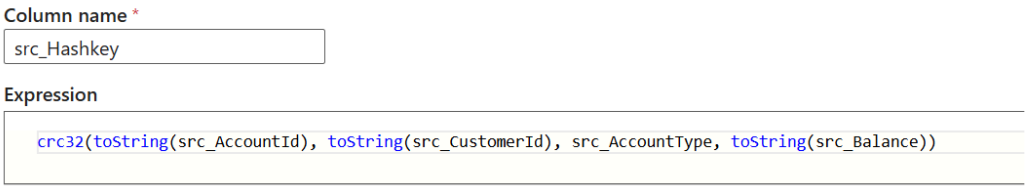
Select transformation - rename columns

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))
```

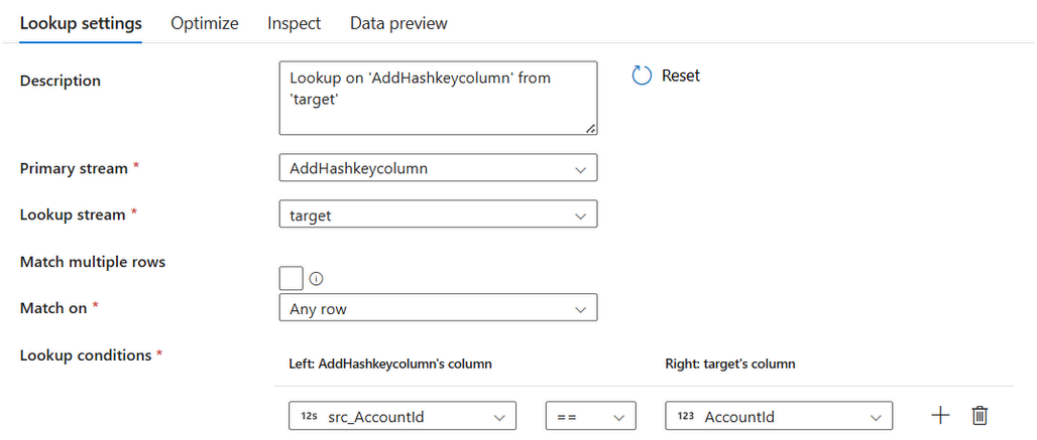
- Purpose: Creates a fingerprint of each record to detect changes.



Creating src_Hashkey column

Step 4.2.5: Lookup Existing Records from SQL DB

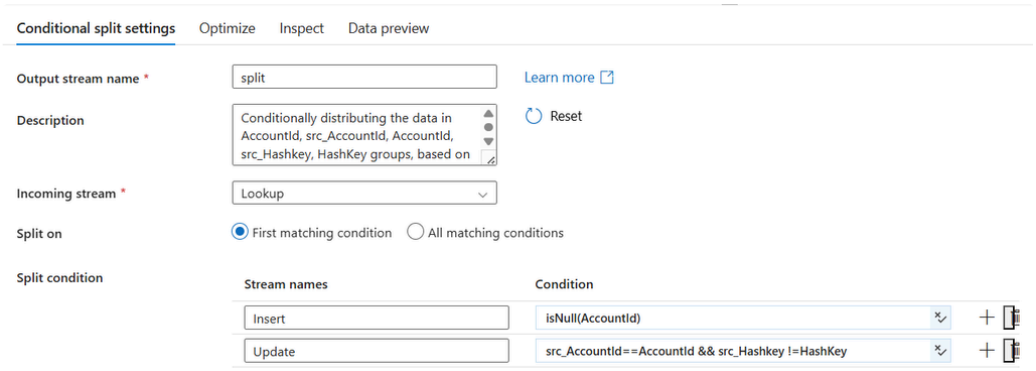
- Transformation Name: `Lookup`
- Join Key: `src_AccountId == AccountId`
- Joins source data with the SQL data to find matches.
- Output includes matched `HashKey`.



Lookup transformation - joining source and target

Step 4.2.6: Split Records into New and Updated

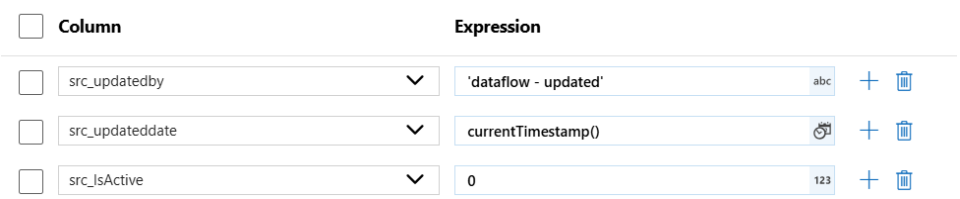
- Transformation Name: `split`
- Split Conditions:
  - **Insert (New)**: `isNull(AccountId)`
  - **Update (Changed)**: `src_AccountId == AccountId && src_Hashkey != HashKey`



Conditional split - insert and update conditions

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

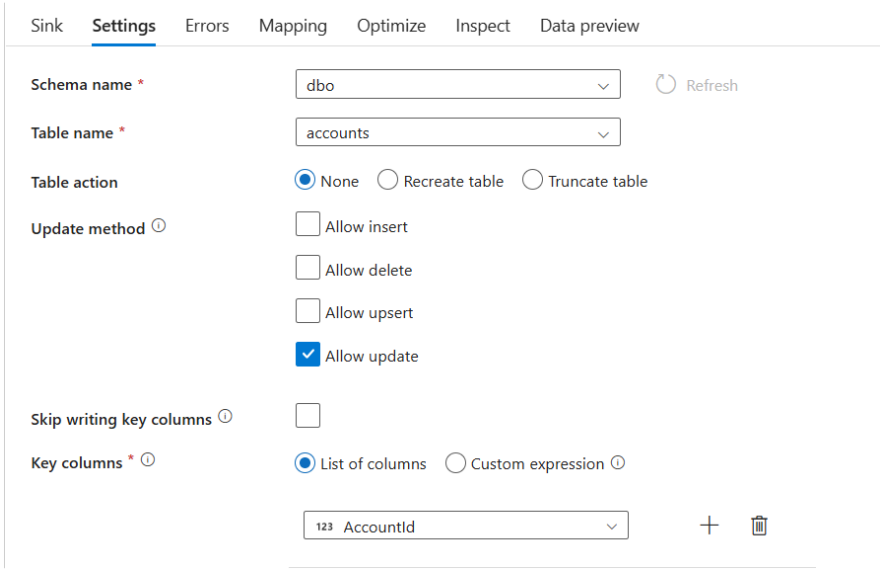
- From `split@Update`, add:
  - **Derived Columns** (`AuditColumns`):
    - `src_updatedby = 'dataflow - updated'`
    - `src_updateddate = currentTimestamp()`
    - `src_IsActive = 0` (mark old record inactive)



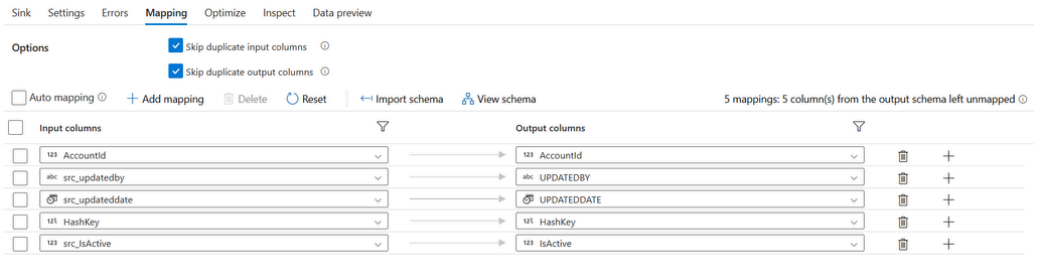
AuditColumns for UpdateSink

  - **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`



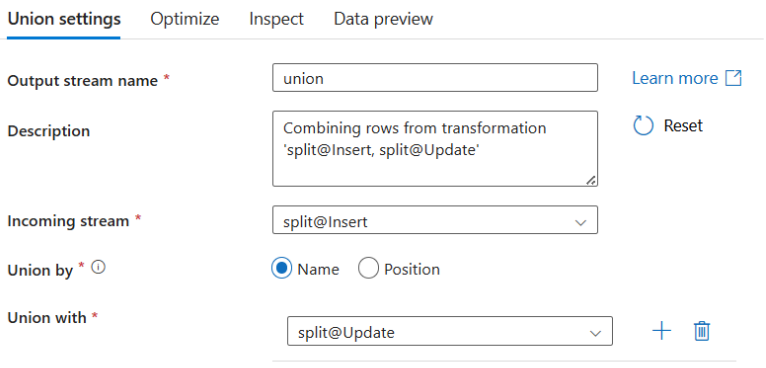
UpdateSink Settings



UpdateSink Maaping

Step 4.2.8: Merge Paths for New Inserts

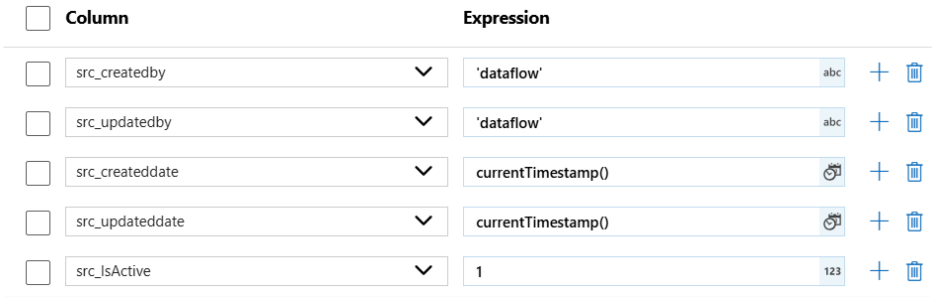
- **Union Transformation** (`union`) combines records from both `split@Insert` and `split@Update` .



Union Settings for Insert records

Step 4.2.9: Add Audit Columns for Inserted Records

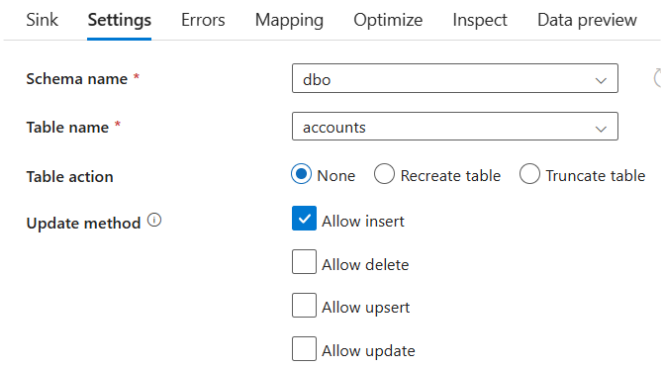
- Transformation Name: `AuditColumn`
- Derived Columns:
  - `src_createdby = 'dataflow'`
  - `src_createddate = currentTimestamp()`
  - `src_updatedby = 'dataflow'`
  - `src_updateddate = currentTimestamp()`
  - `src_IsActive = 1`



Audit Columns for Inserted Records

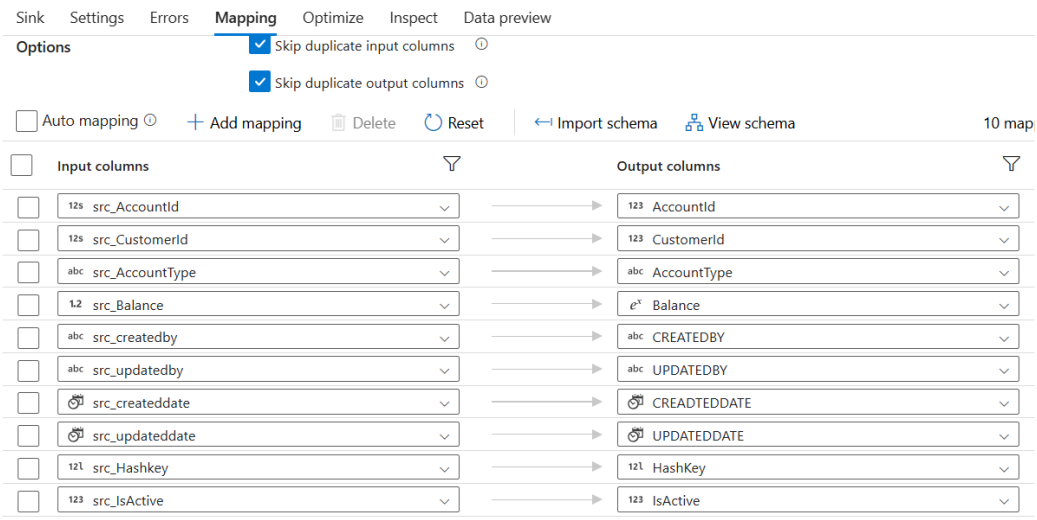
Step 4.2.10: Sink: Insert New Records into SQL

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



- Sink Settings for InsertSink

- Mapping:



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.

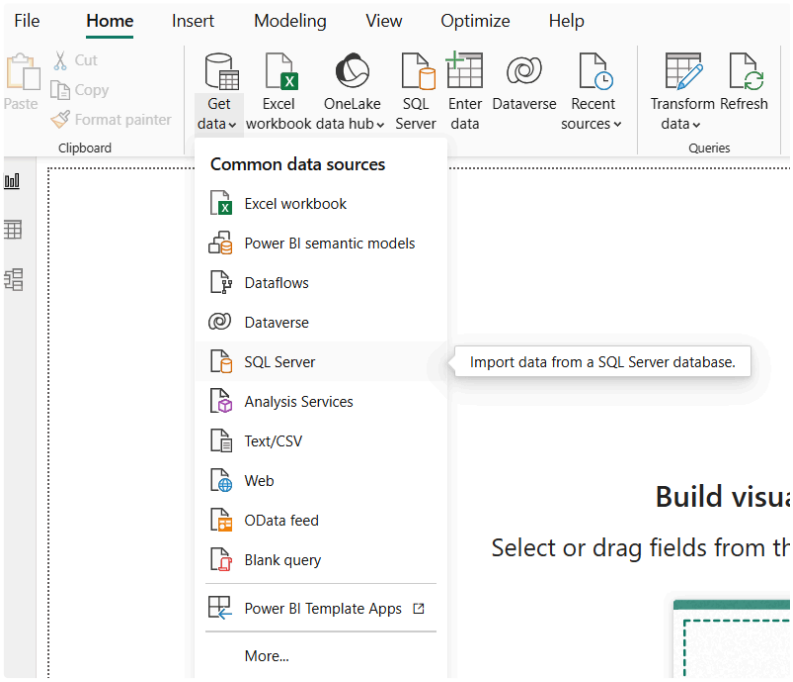
STEP 5 - Create and publish reports

Step 5.1: Sign In with Your Microsoft Fabric Account

- Launch Power BI Desktop.
- Click on the "Sign in" button on the top right corner.
- Use your Microsoft Fabric account credentials to sign in.

Step 5.2: Connect to the SQL Server

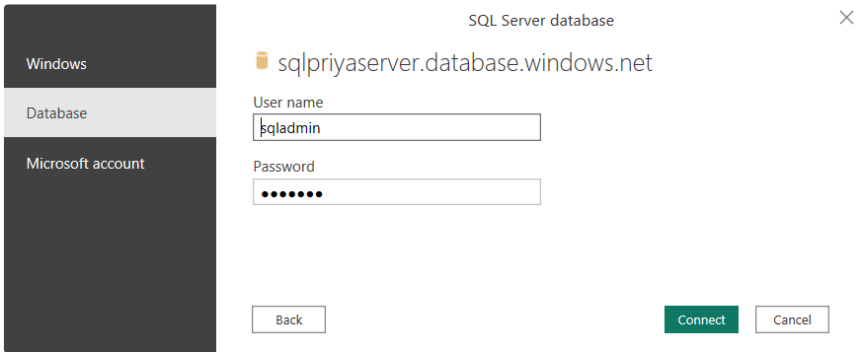
- In Power BI Desktop, go to the **Home** tab.
- Click **Get Data** > **SQL Server**.



- In the **SQL Server database** window:
  - Enter the **Server name** (from your Azure SQL Database).
  - If required, enter the **Database name** or leave it blank to select later.

Step 5.3: Authentication

- Select **Database authentication** if prompted, and provide your SQL Server username and password.



Step 5.4: Select Tables

- After successful connection, the **Navigator** pane appears.
- Select the tables you want to include in your report.
- Click **Load** to import data into Power BI.

Step 5.5: Create the Report

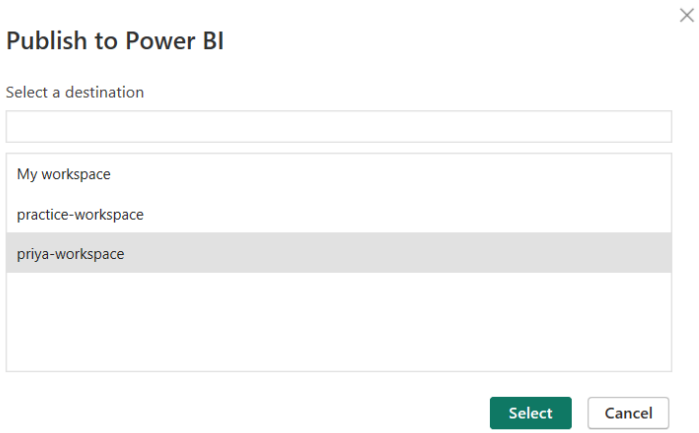
- Use the fields pane to drag and drop data onto the canvas.
- Build visuals like tables, charts, slicers, or maps depending on your requirements.

Step 5.6: Save the Report

- Click **File** > **Save As**.
- Choose a local folder and save the report with a suitable name (e.g., `SalesReport.pbix`).

Step 5.7: Publish the Report to Microsoft Fabric

- Go to the **Home** tab and click **Publish**.
- If prompted, sign in again with your Microsoft Fabric account.
- Choose the target **Workspace** where you want to publish the report.



Step 5.8: View the Report in Microsoft Fabric

- Log in to Microsoft Fabric.
- Navigate to the **Workspace** you published the report to.
- Open the report and explore or share it as needed.