Taksh Shah

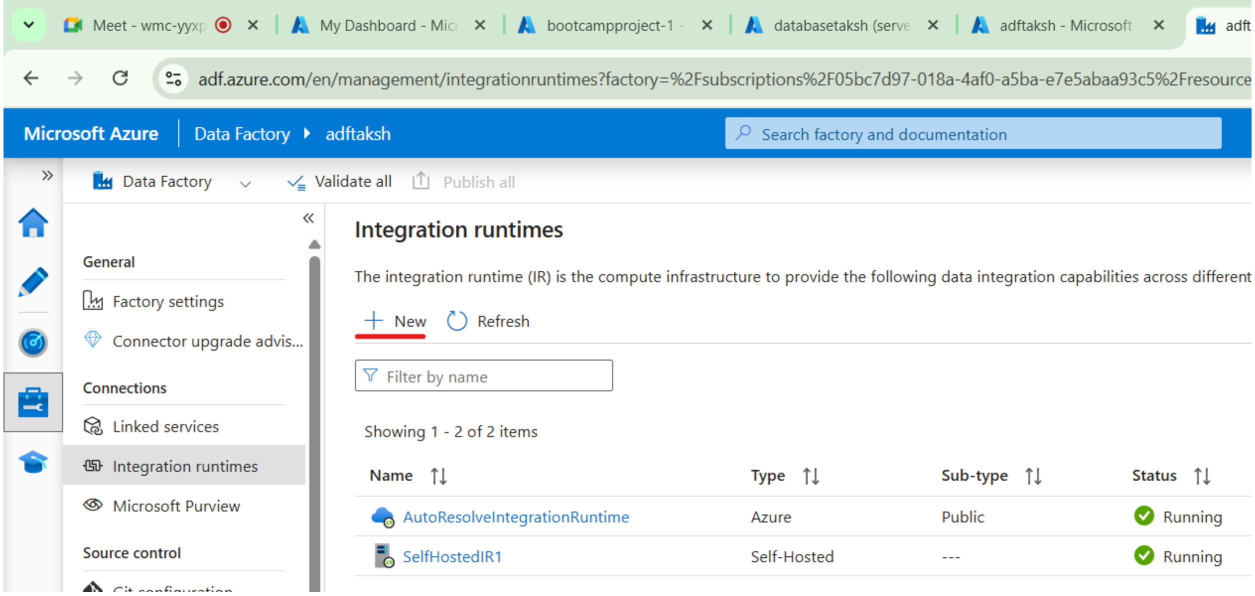
Project: Customer Account Analysis

Objective:

The project aims to design and implement a robust data pipeline for processing customer account data. This includes copying data from a backend team's storage account, performing necessary transformations using ADF and upserting (inserting or updating) data from a file stored in Azure Data Lake Storage ADLS Gen2 storage into sql database table. The pipeline aims to ensure efficient, accurate, and scalable data processing to support downstream analytics and reporting needs.

To begin with, in order to get data from a local system to adlsgen2 storage account on Microsoft Azure, we'll use self-hosted integration runtime (IR) to achieve this task. You can utilize virtual machine for this task as well, but we'll use self-hosted IR.

For that, go to this website: https://www.microsoft.com/en-us/download/details.aspx?id=39717 and download "IntegrationRuntime_5.50.9171.1.msi" into your local system(pc/laptop). Then, in your ADF (Azure Data Factory), go to manage -> integration runtimes and create a new IR.



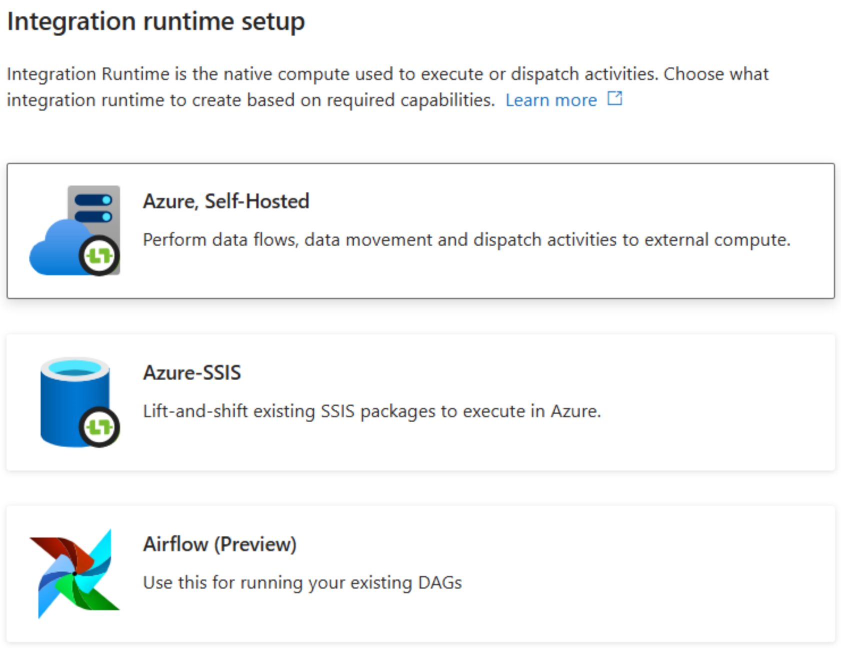
Integration runtimes

The integration runtime (IR) is the compute infrastructure to provide the following data integration capabilities across different

Showing 1 - 2 of 2 items

Name	Type	Sub-type	Status
AutoResolveIntegrationRuntime	Azure	Public	Running
SelfHostedIR1	Self-Hosted	---	Running

Then, select “Azure, Self-Hosted” option and select continue.



Integration runtime setup

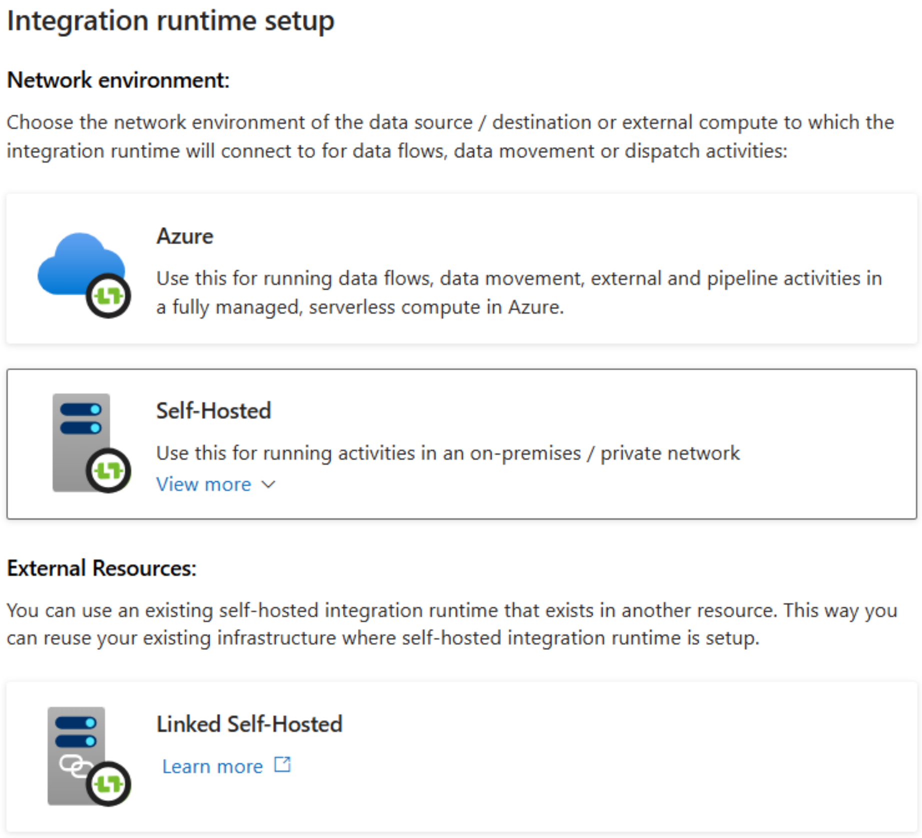
Integration Runtime is the native compute used to execute or dispatch activities. Choose what integration runtime to create based on required capabilities. Learn more

Azure, Self-Hosted
Perform data flows, data movement and dispatch activities to external compute.

Azure-SSIS
Lift-and-shift existing SSIS packages to execute in Azure.

Airflow (Preview)
Use this for running your existing DAGs

Again, select “Self-Hosted” and continue.



Integration runtime setup

Network environment:

Choose the network environment of the data source / destination or external compute to which the integration runtime will connect to for data flows, data movement or dispatch activities:

Azure
Use this for running data flows, data movement, external and pipeline activities in a fully managed, serverless compute in Azure.

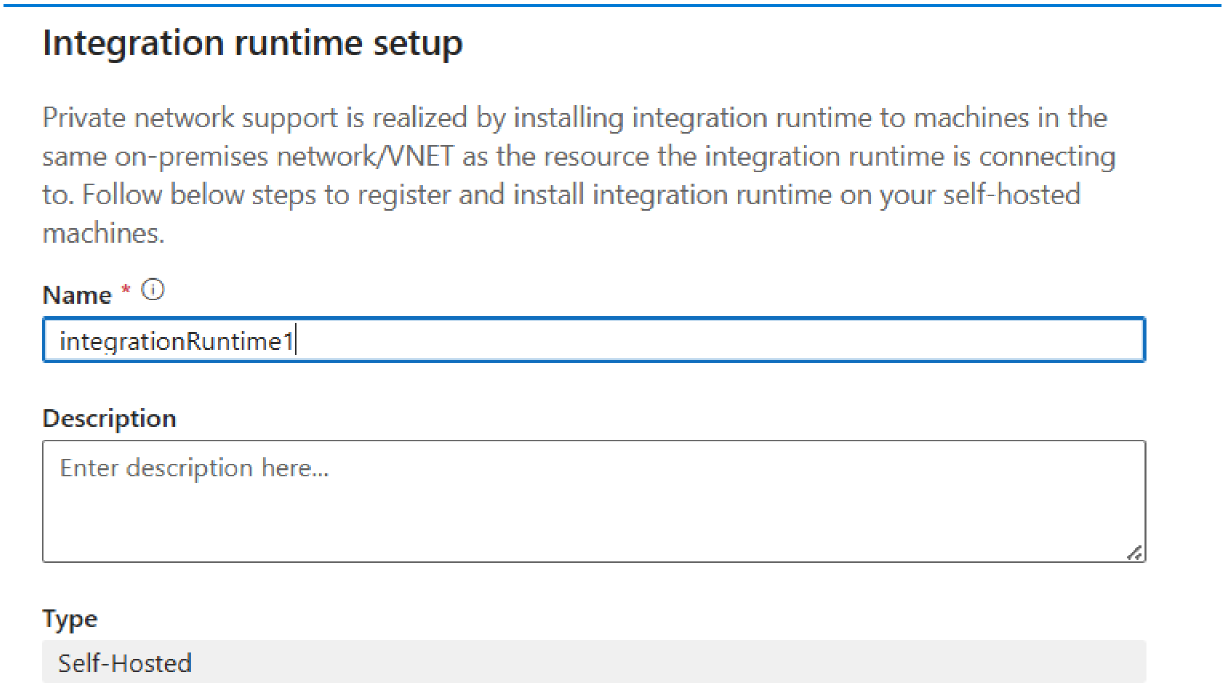
Self-Hosted
Use this for running activities in an on-premises / private network
View more

External Resources:

You can use an existing self-hosted integration runtime that exists in another resource. This way you can reuse your existing infrastructure where self-hosted integration runtime is setup.

Linked Self-Hosted
Learn more

Now, give a name to your self-hosted IR and click create.



Integration runtime setup

Private network support is realized by installing integration runtime to machines in the same on-premises network/VNET as the resource the integration runtime is connecting to. Follow below steps to register and install integration runtime on your self-hosted machines.

Name * ⓘ

integrationRuntime1

Description

Enter description here...

Type

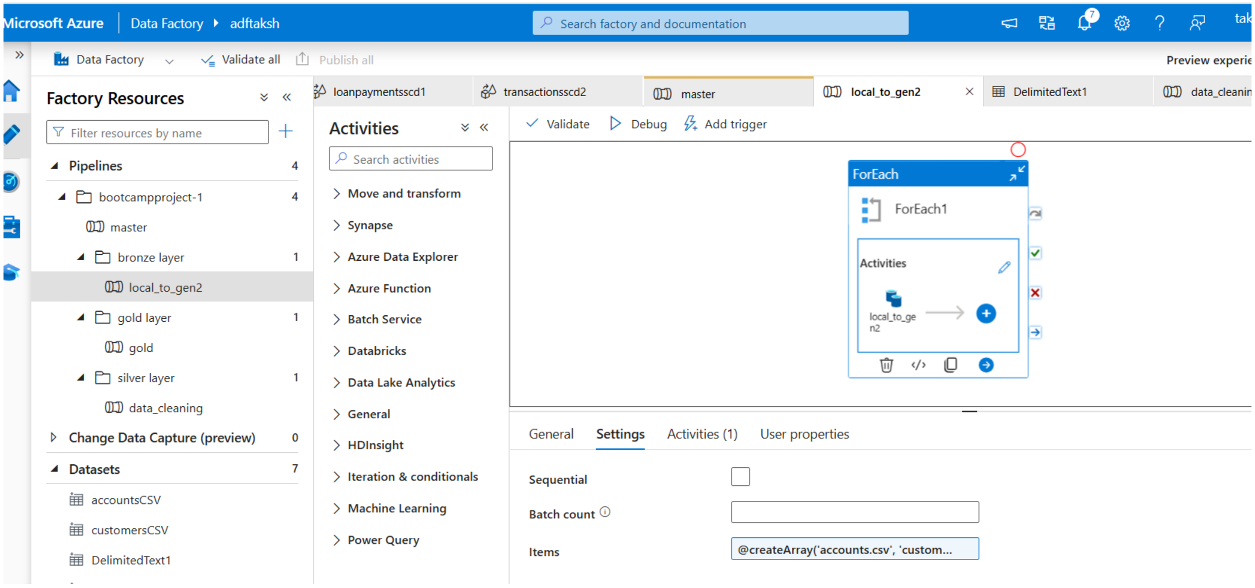
Self-Hosted

This will create your self-hosted IR and show 2 keys – key1 and key2 – copy key1 as we'll use it later. Now, let's setup the IR that we downloaded from the previously mentioned Microsoft website. Once you open its setup and install all the files, it'll then prompt you to enter a key for your IR – this is where we'll utilize the key1 that we had previously copied.

Bronze Layer Pipeline:

This pipeline will be responsible in copying the data from our local system to the adlsgen2 storage account. We have 5 csv files in our dataset namely 'accounts.csv', 'customers.csv', 'loan_payments.csv', 'loans.csv', 'transactions.csv'

Dataset link: https://www.kaggle.com/datasets/varunkumari/ai-bank-dataset



Sequential □

Batch count

Items @createArray('accounts.csv', 'custom...

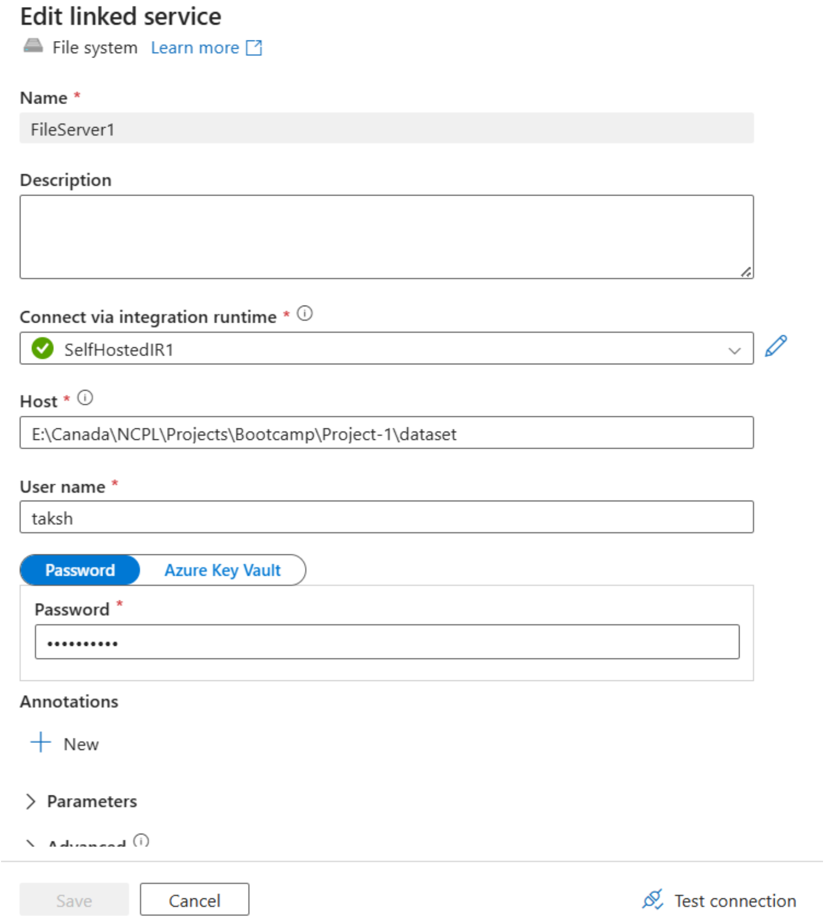
Bronze layer pipeline – for-each loop with a copy data activity

To build this pipeline, we'll use a copy data activity within a for-each loop, so instead of using the copy data activity multiple times to get all the files from the dataset into our storage account, a loop will prove more convenient and efficient to do the same task.

Create a for-each activity. Under its settings, in "items" field, we'll have to provide a value as it's a requirement. Now for-each loop takes input in the form of arrays, meaning we'll have to pass a value as an array. Since we want to get the files from our dataset folder present on our local system, we'll provide an array of the filenames to our loop, so it'll iterate through those names and then the copy data activity within the loop will copy all the files into our storage account. Use the following command as an input to the "items" field:

```
@createArray('accounts.csv', 'customers.csv', 'loan_payments.csv', 'loans.csv',
'transactions.csv')
```

Now, let's create a copy-data activity within the for-each loop. Under "source", let's create a new dataset. Here, we'll have to create a new linked service for our file system. So we'll select source as "file system" as data is present in our local system and source will be adlsgen2 where the data will be stored a CSV file. The image below shows what the linked service settings will look like.



Edit linked service

File system Learn more

Name *

FileServer1

Description

Connect via integration runtime *

SelfHostedIR1

Host *

E:\Canada\NCPL\Projects\Bootcamp\Project-1\dataset

User name *

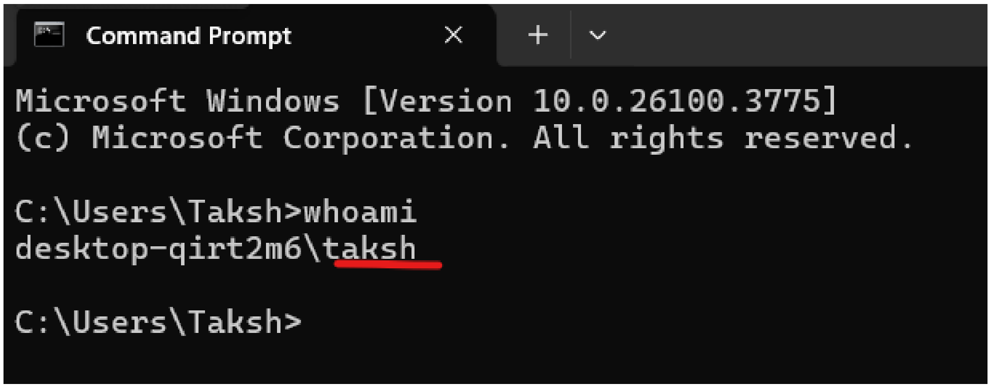
taksh

Password *

••••••••••

Here, “host” will be the path where the dataset is stored in the local system. Username and password – these credentials are crucial – they can be found as follows:

In your command prompt, write the command “whoami” to figure out the user name.

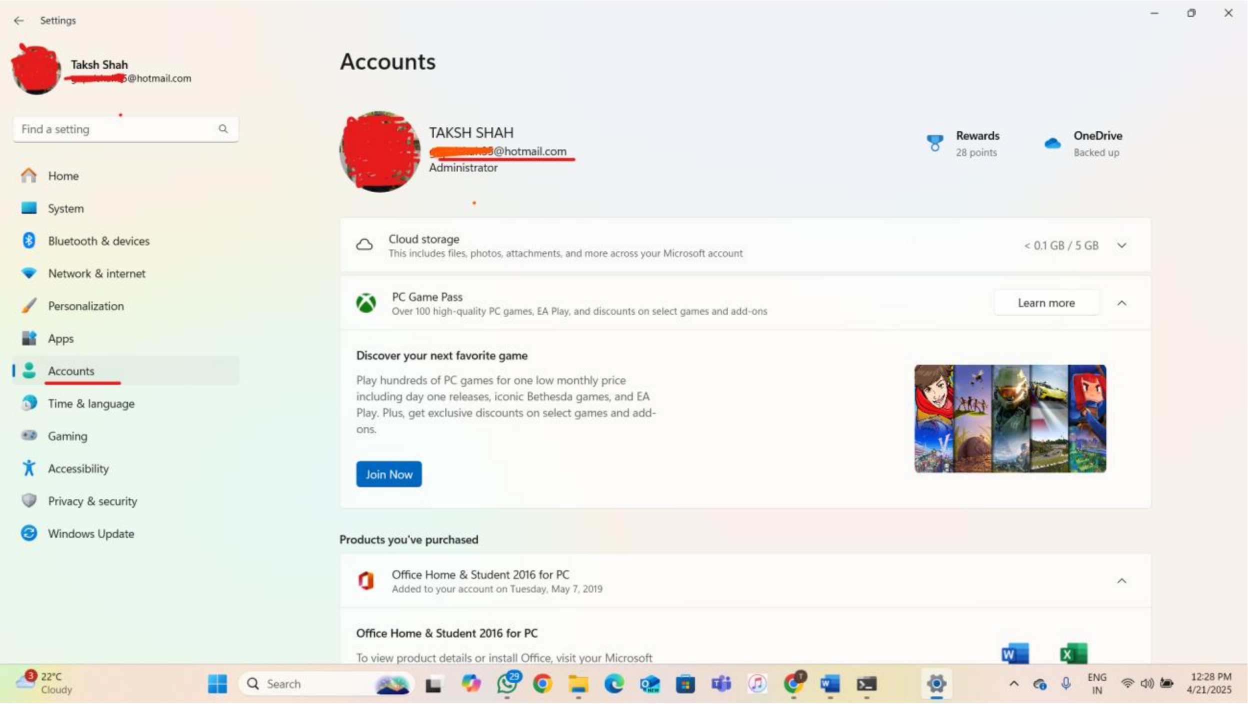


```
Microsoft Windows [Version 10.0.26100.3775]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Taksh>whoami
desktop-qirt2m6\taksh

C:\Users\Taksh>
```

The password will be either of these 2 – your laptop/pc user password (ie, the password with which you login to your laptop/pc) OR the password of the email with which you’re currently signed into in your laptop/pc – this can be found in settings -> accounts if it’s a windows system.



Taksh Shah
[redacted]@hotmail.com

Accounts

TAKSH SHAH
[redacted]@hotmail.com
Administrator

Rewards
28 points

OneDrive
Backed up

Cloud storage
This includes files, photos, attachments, and more across your Microsoft account
< 0.1 GB / 5 GB

PC Game Pass
Over 100 high-quality PC games, EA Play, and discounts on select games and add-ons

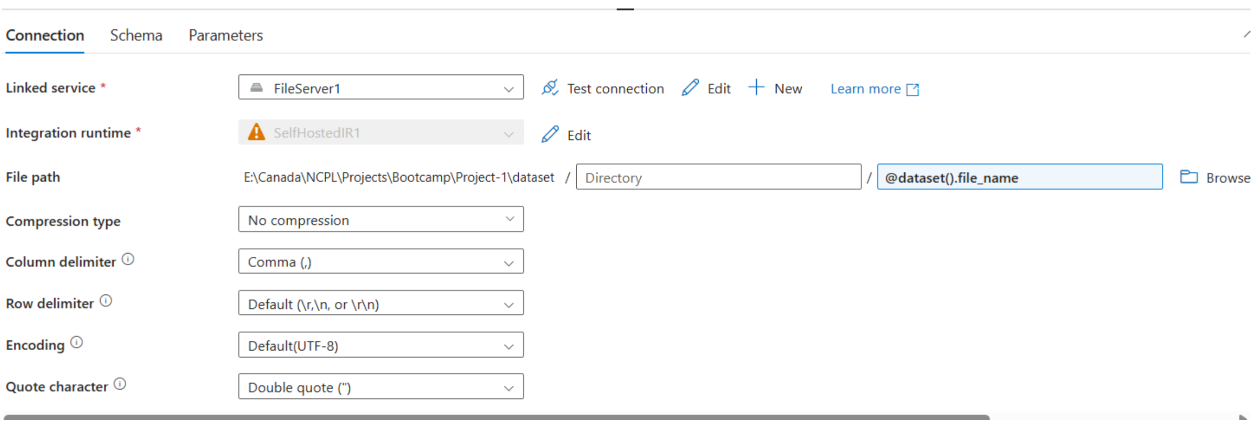
Discover your next favorite game
Play hundreds of PC games for one low monthly price including day one releases, iconic Bethesda games, and EA Play. Plus, get exclusive discounts on select games and add-ons.

Products you've purchased

Office Home & Student 2016 for PC
Added to your account on Tuesday, May 7, 2019

Office Home & Student 2016 for PC
To view product details or install Office, visit your Microsoft

Once your linked service is created, select the self-hosted integration runtime which we created earlier, provide file path as the location of the dataset folder and make sure to select first row as header option. Here, we’ll create a parameter called file_name which will be useful in iterating through all the files present in the dataset folder.



Linked service * FileServer1 — Test connection · Edit · New · Learn more

Integration runtime * SelfHostedIR1 — Edit

File path E:\Canada\NCPL\Projects\Bootcamp\Project-1\dataset / Directory / @dataset().file_name — Browse

Compression type: No compression

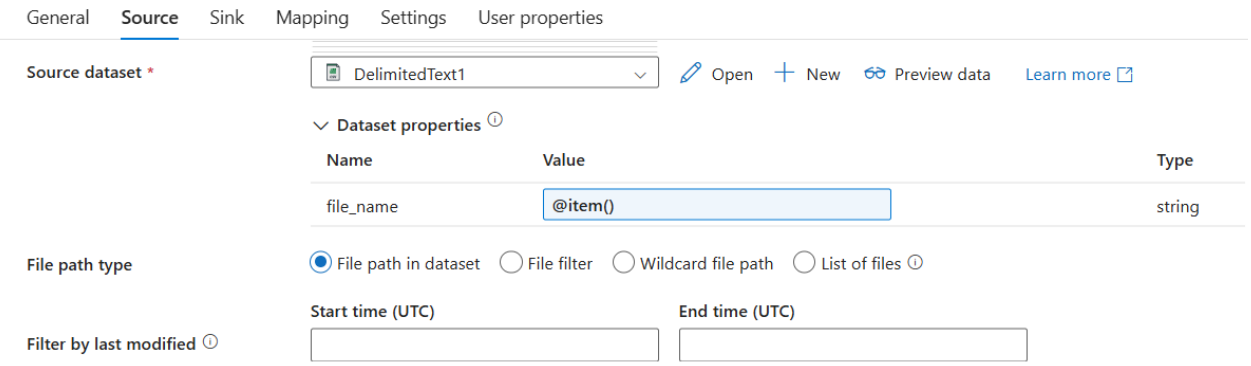
Column delimiter ⓘ: Comma (,)

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

Encoding ⓘ: Default(UTF-8)

Quote character ⓘ: Double quote (")

This completes the creation of our dataset for the copy data activity. Make sure to provide the value for file_name parameter which we created earlier.



Source dataset * DelimitedText1 — Open · New · Preview data · Learn more

Dataset properties ⓘ

Name	Value	Type
file_name	@item()	string

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

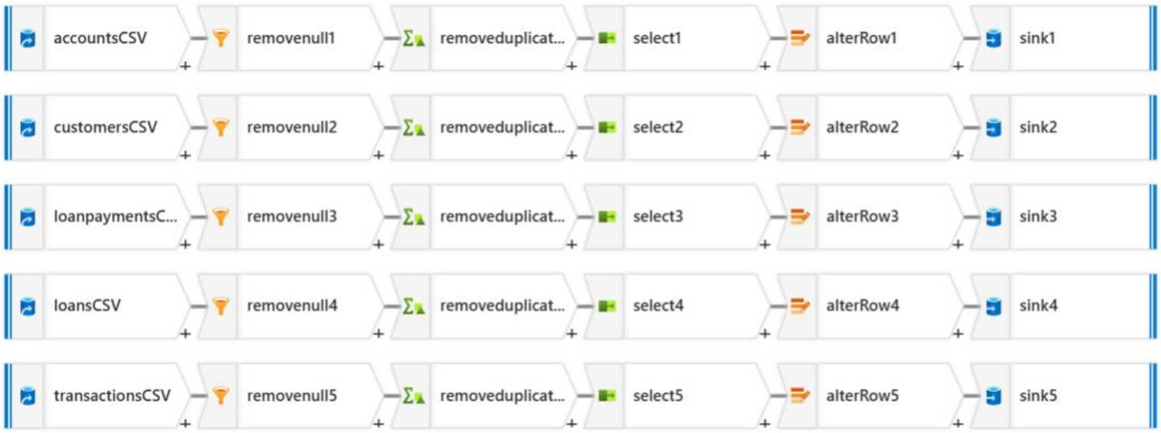
Filter by last modified ⓘ: Start time (UTC) — End time (UTC)

For the sink part of copy data activity, we'll create a new dataset. Linked service will be the one for adlsgen2 storage account with self-hosted IR. File path will be your container in the adlsgen2 storage account where you can create a folder called "bronze layer" and store the files. Make sure to select first row as header option. Under "sink" section, make sure to change file extension to ".csv". This marks the end of bronze layer pipeline.

<u>Silver Layer Pipeline:</u>

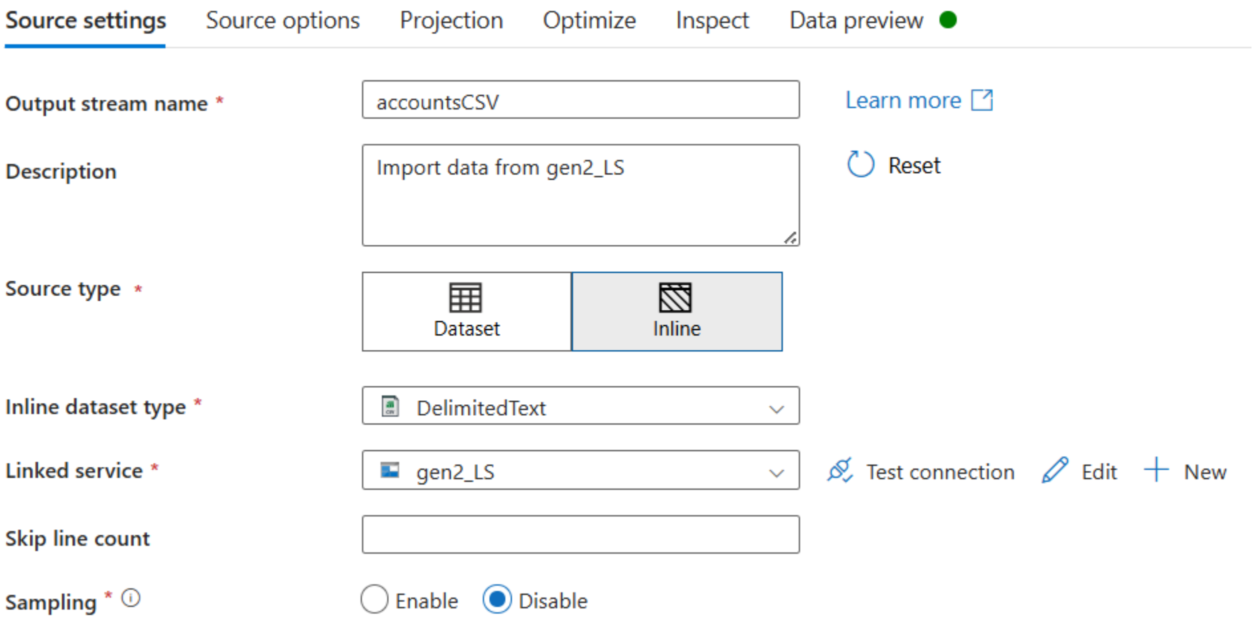
This pipeline will be responsible for cleaning the data that we just stored into our adlsgen2 storage account from the local system so that it can be used for processing and eventually storing the data into our sql database. We'll remove null values, remove duplicate values and rename the columns at this stage.

Here, we'll use a dataflow within which we'll use 5 sources – 1 for each file – and apply all the necessary transformations and eventually store the data back into our adlsgen2 account in a folder called "silver layer" and in delta format.



Silver layer pipeline – inside the dataflow

Let's take the first source to understand the pipeline – accountsCSV



Output stream name * accountsCSV

Description Import data from gen2_LS

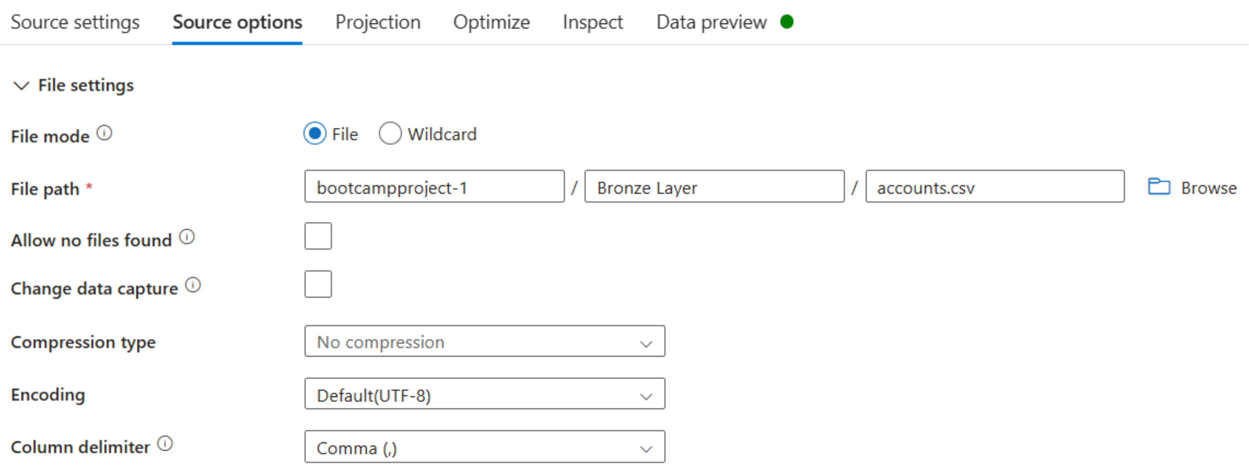
Source type * Dataset [✓ Inline]

Inline dataset type * DelimitedText

Linked service * gen2_LS

Skip line count

Sampling * ○ Enable ● Disable



File settings

File mode ⓘ ● File ○ Wildcard

File path * bootcampproject-1 / Bronze Layer / accounts.csv

Allow no files found ⓘ □

Change data capture ⓘ □

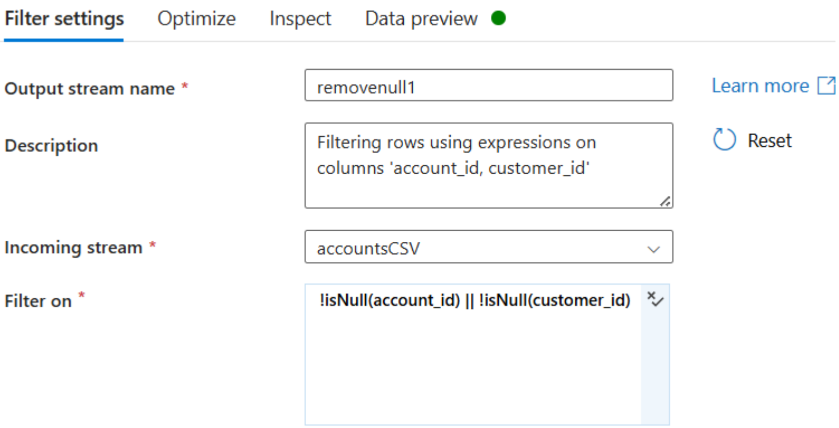
Compression type: No compression

Encoding: Default(UTF-8)

Column delimiter ⓘ Comma (,)

In source settings, make sure to select first row as header option.

The source is then connected to a filter transformation which will be used to check the data for null values. We’re only checking account_id and customer_id columns here because any of these fields being empty suggests that the record may be null or doesn’t exist.



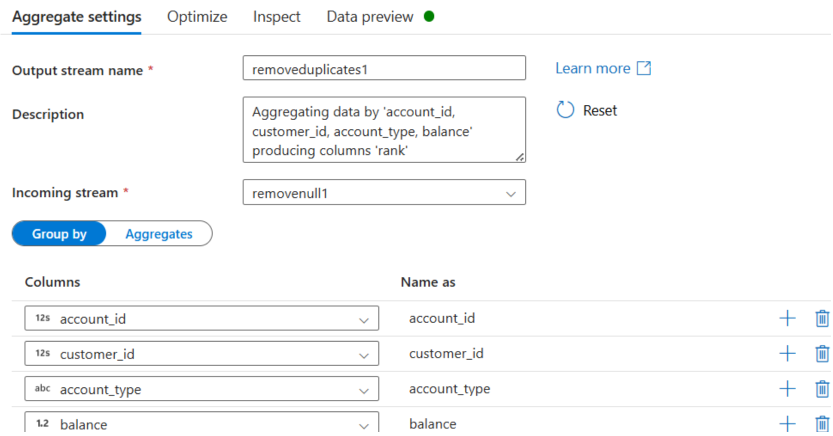
Output stream name * removenull1

Description: Filtering rows using expressions on columns 'account_id, customer_id'

Incoming stream * accountsCSV

Filter on * !isNull(account_id) || !isNull(customer_id)

Next, we connect the filter transformation to an aggregate transformation which will be used to remove duplicate values. In “group by” section, we’ll add all the columns of accountsCSV as we want to check for duplicates in each field. However, aggregate transformation requires us to provide atleast 1 column. To fulfil that requirement, we’ll create a dummy column called “rank” whose expression will be first(account_id) – this basically means that it’ll take the first occurring value in the column “account_id”.



Aggregate settings | Optimize | Inspect | Data preview ●

Output stream name *: removeduplicates1

Learn more

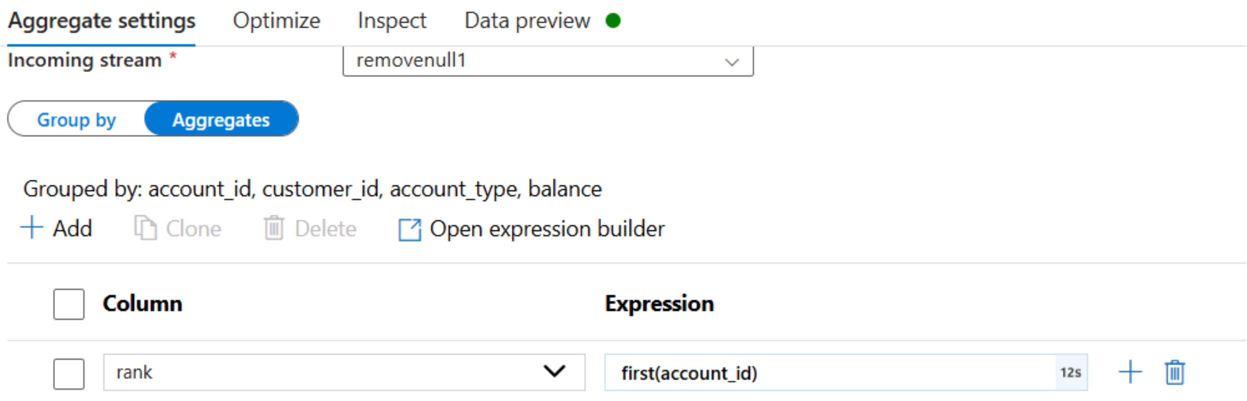
Description: Aggregating data by 'account_id, customer_id, account_type, balance' producing columns 'rank'

Reset

Incoming stream *: removenull1

Group by | Aggregates

Columns	Name as
12s account_id	account_id
12s customer_id	customer_id
abc account_type	account_type
1.2 balance	balance



Aggregate settings | Optimize | Inspect | Data preview ●

Incoming stream *: removenull1

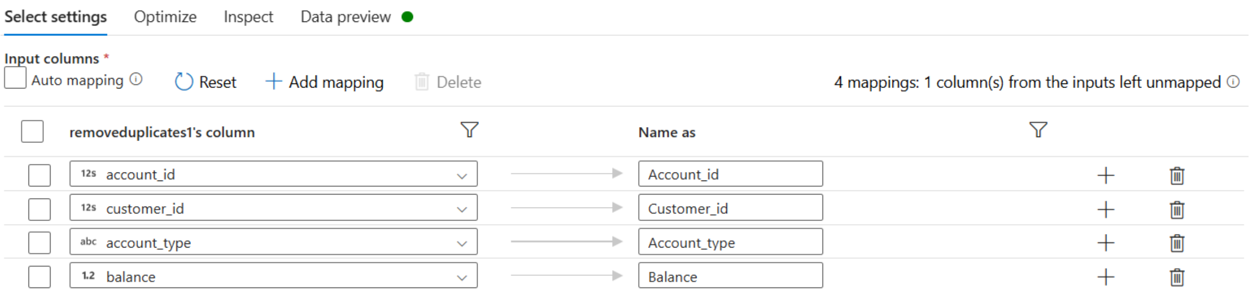
Group by | Aggregates

Grouped by: account_id, customer_id, account_type, balance

+ Add | Clone | Delete | Open expression builder

□	Column	Expression
□	rank	first(account_id) 12s

Next, connect the aggregate transformation to a select transformation which we'll use to drop the dummy "rank" column which we created in the last step and also rename the columns.



Select settings | Optimize | Inspect | Data preview ●

Input columns *

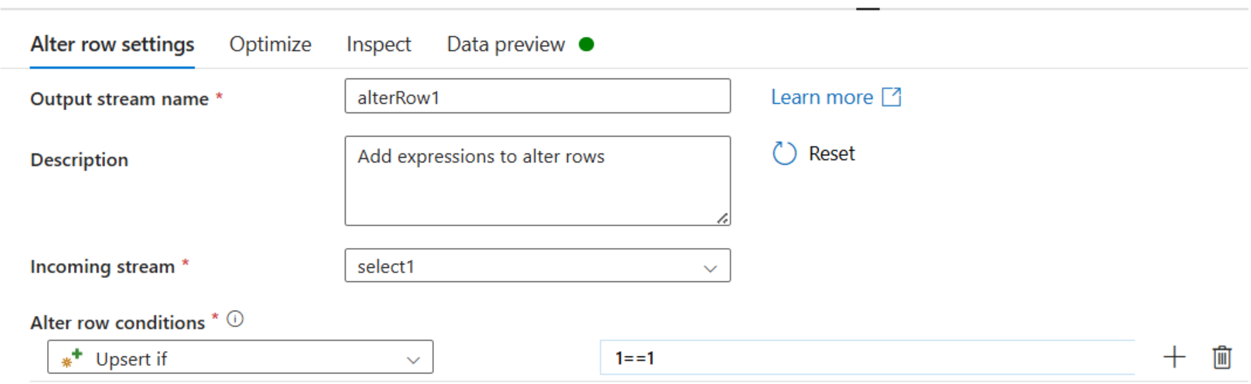
□ Auto mapping | Reset | + Add mapping | Delete

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

□	removeduplicates1's column	Name as
□	12s account_id	Account_id
□	12s customer_id	Customer_id
□	abc account_type	Account_type
□	1.2 balance	Balance

So, under select settings, simply delete the mapping of rank column and under "name as" section, you can rename the columns. We have simply capitalized the first letter of all the column names.

Now, we'll connect the select transformation to an alter row transformation.



Output stream name * alterRow1

Description: Add expressions to alter rows

Incoming stream * select1

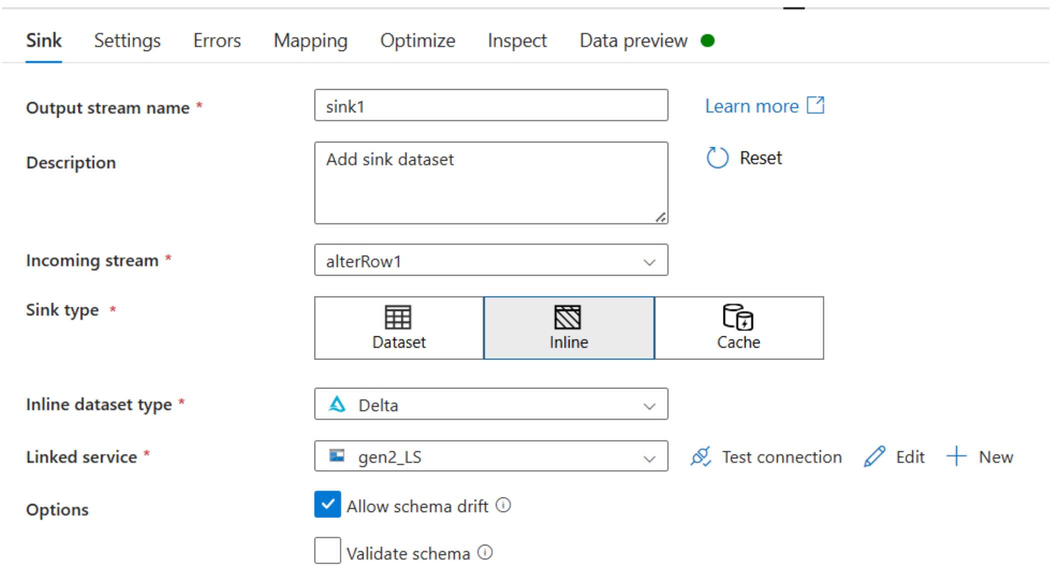
Alter row conditions *

Upsert if 1==1

Here, we select the **"Upsert if"** option in the Alter Row transformation and set the condition to **1==1**, which effectively allows **all changes (inserts and updates)** to pass through. This is because, by the time data reaches this point in the Data Flow, all necessary transformations have already been applied. Our goal now is simply to allow those changes and persist the final data in **Delta format** in the **Silver layer** folder within the **ADLS Gen2** account.

Using the Alter Row transformation here is essential: it enables **fine-grained control over row-level operations**, such as insert, update, delete, or upsert, and allows you to dynamically assign these actions based on conditions. In this case, we want to ensure that **new records are inserted** and **existing records are updated** appropriately before writing to the sink. This is why Alter Row is used—to direct the behavior of how each row is written to the target storage.

Finally, we connect the alter row transformation to a sink.



Output stream name * sink1

Description: Add sink dataset

Incoming stream * alterRow1

Sink type * Dataset / Inline / Cache (Inline selected)

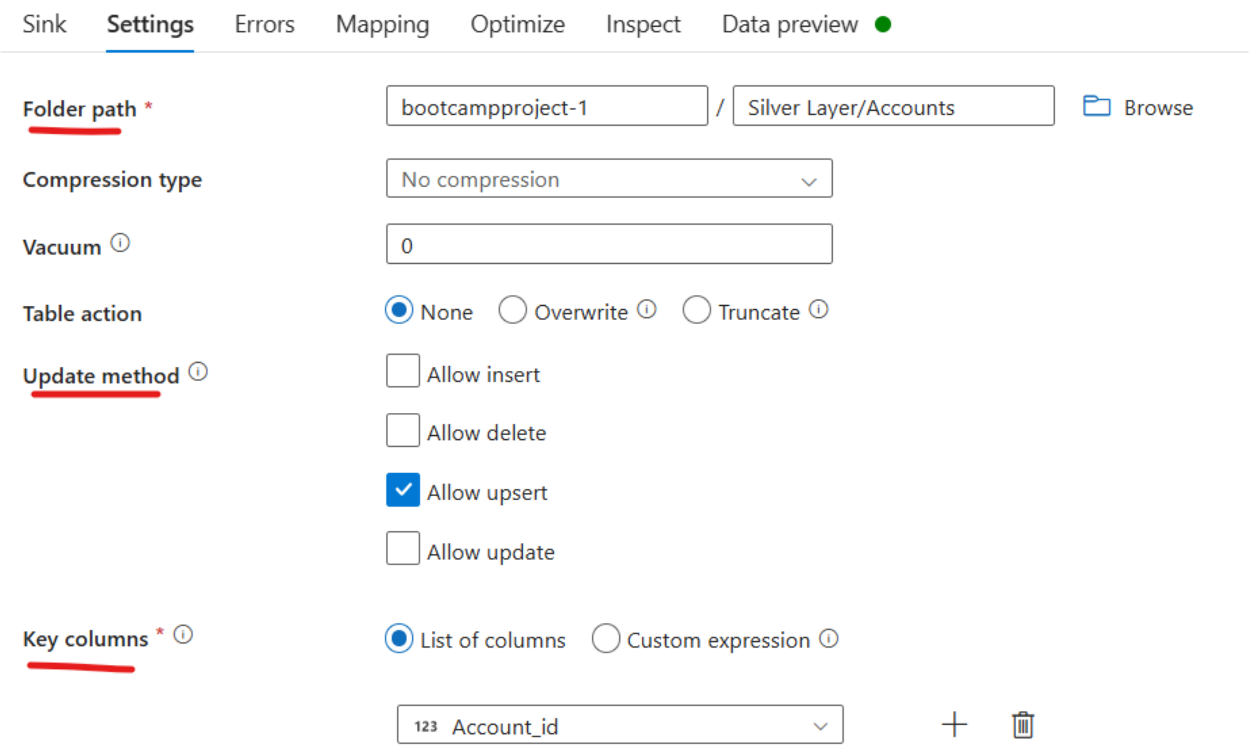
Inline dataset type * Delta

Linked service * gen2_LS

Options

☑ Allow schema drift

☐ Validate schema



Folder path * bootcampproject-1 / Silver Layer/Accounts

Compression type: No compression

Vacuum: 0

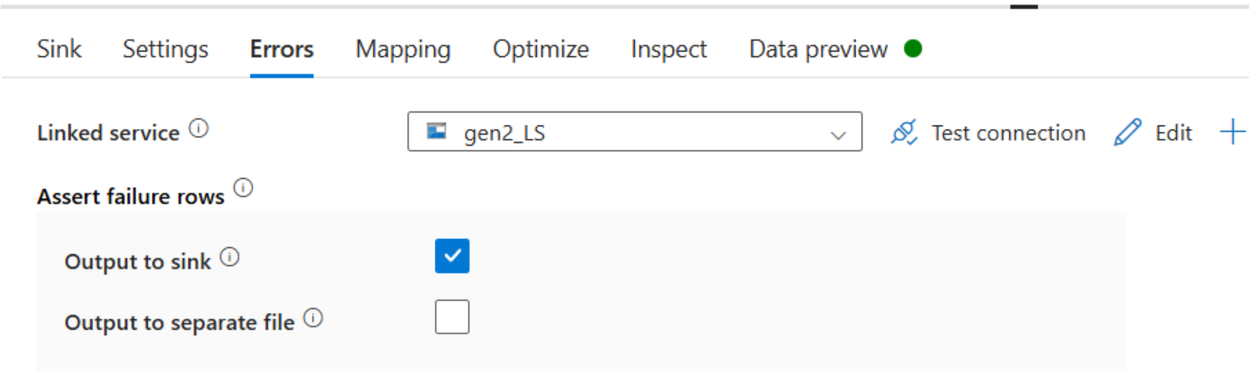
Table action: ● None ○ Overwrite ○ Truncate

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

Key columns * ● List of columns ○ Custom expression

123 Account_id

As per the above image, make sure to give path as the location where we want to store are cleaned data. We'll store that in a folder called "Silver Layer", but since we're storing the data in delta format, it'll also have a log file, so we'll create another folder for each csv file within the silver layer folder to avoid creating a mess and organize the data in a well-structured format. And make sure to select "allow upsert" as the update method and provide a key column for that particular dataset file.



Linked service: gen2_LS

Assert failure rows

Output to sink ☑

Output to separate file □

In "errors" section, we just have to select the linked service for adlsgen2 account. Here, we don't need to select the linked service with self-hosted IR as we're moving data within cloud, not from local system to cloud. Similarly, we have to build the pipeline for all the other sources (1 for each csv file in the dataset).

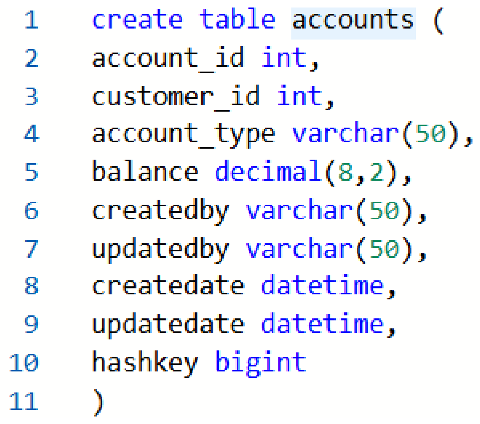
Gold Layer Pipeline:

This pipeline will be useful in processing the data in SCDType-1 and SCDType-2 fashion and eventually store the data into an azure sql database.

First, we'll create 5 tables in our sql database – 3 for scdtype-1 and 2 for scdtype-2

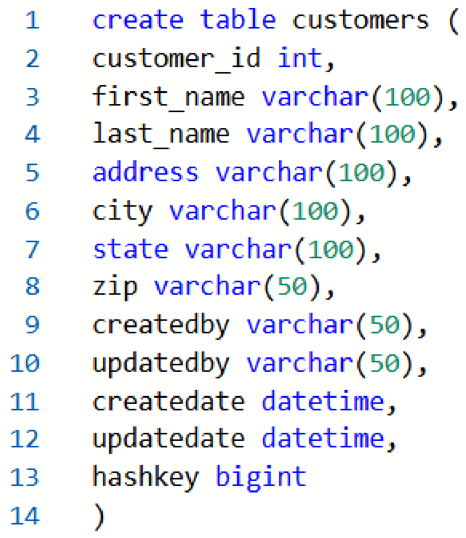
We'll use accounts.csv, customers.csv, loan_payments.csv for scdtype-1 and loans.csv, transformations.csv for scdtype-2

The images of the code used to create all the tables are provided below:



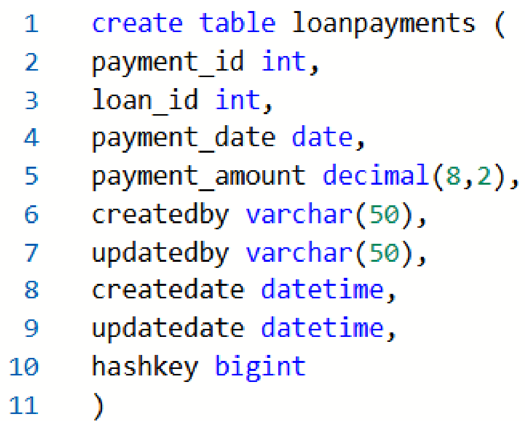
```sql
create table accounts (
account_id int,
customer_id int,
account_type varchar(50),
balance decimal(8,2),
createdby varchar(50),
updatedby varchar(50),
createdate datetime,
updatedate datetime,
hashkey bigint
)
```

Accounts table



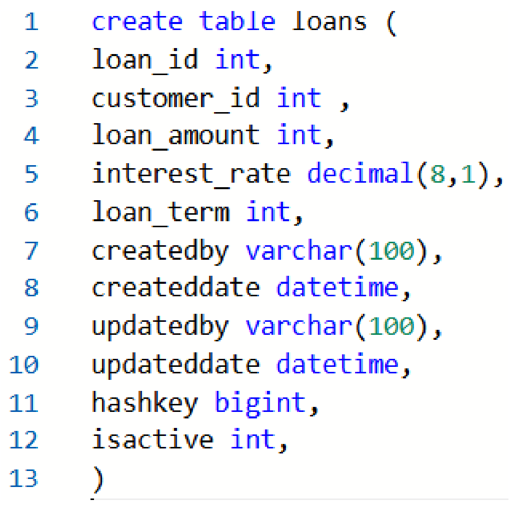
```sql
create table customers (
customer_id int,
first_name varchar(100),
last_name varchar(100),
address varchar(100),
city varchar(100),
state varchar(100),
zip varchar(50),
createdby varchar(50),
updatedby varchar(50),
createdate datetime,
updatedate datetime,
hashkey bigint
)
```

Customers table



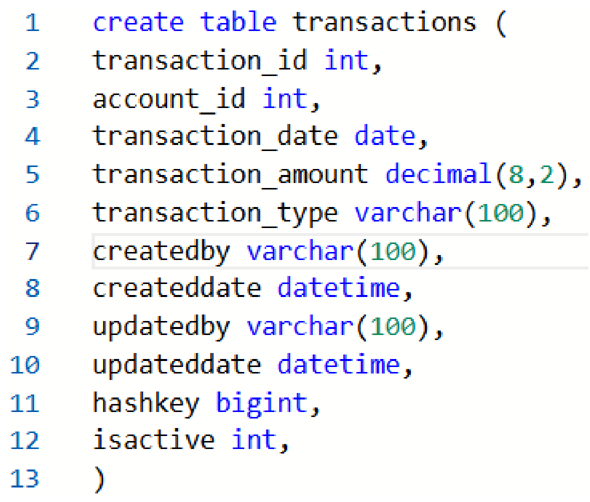
```sql
create table loanpayments (
payment_id int,
loan_id int,
payment_date date,
payment_amount decimal(8,2),
createdby varchar(50),
updatedby varchar(50),
createdate datetime,
updatedate datetime,
hashkey bigint
)
```

Loanpayments table



```sql
create table loans (
loan_id int,
customer_id int ,
loan_amount int,
interest_rate decimal(8,1),
loan_term int,
createdby varchar(100),
createddate datetime,
updatedby varchar(100),
updateddate datetime,
hashkey bigint,
isactive int,
)
```

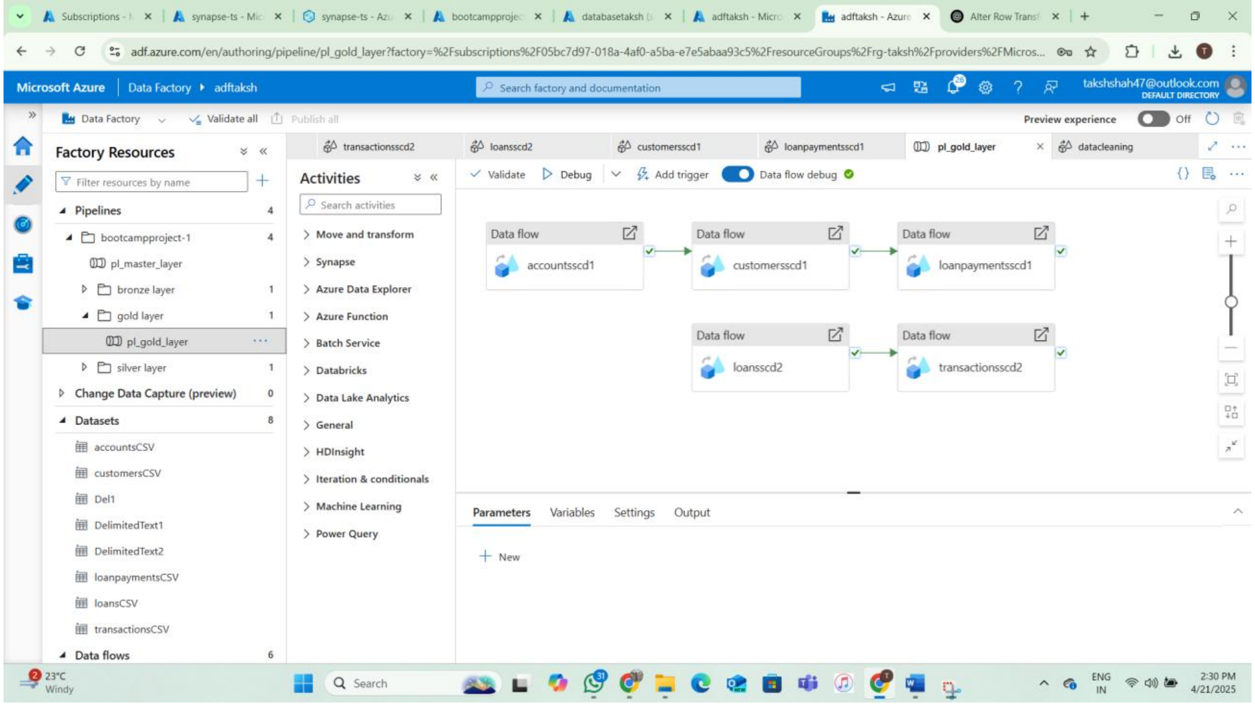
Loans table



```sql
create table transactions (
transaction_id int,
account_id int,
transaction_date date,
transaction_amount decimal(8,2),
transaction_type varchar(100),
createdby varchar(100),
createddate datetime,
updatedby varchar(100),
updateddate datetime,
hashkey bigint,
isactive int,
)
```

Transactions table

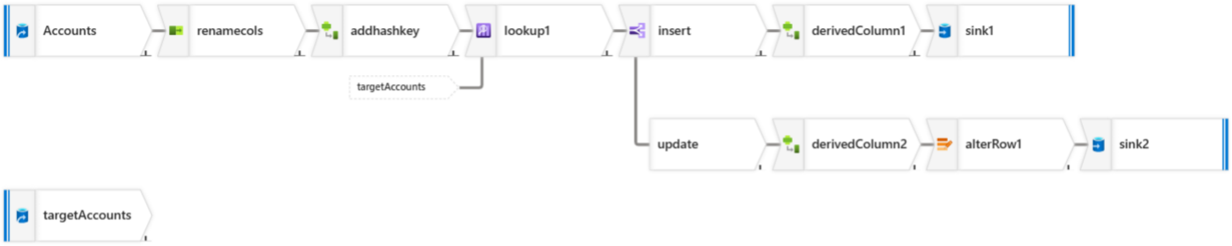
Once we’ve created these tables in our azure sql database, we’re ready to begin developing the gold layer pipeline.



Gold layer pipeline

We’ll utilize 5 dataflows here, 1 for each csv file in the dataset, where 3 of them will be used to implement scdtype-1 logic (accounts, customers, loanpayments) while the other 2 will be used for scdtype-2 logic (loans, transactions). We’ll connect the 3 dataflows for scdtype-1 and 2 dataflows for scdtype-2 as show in the image above which will then be parallelly processed when the pipeline is run.

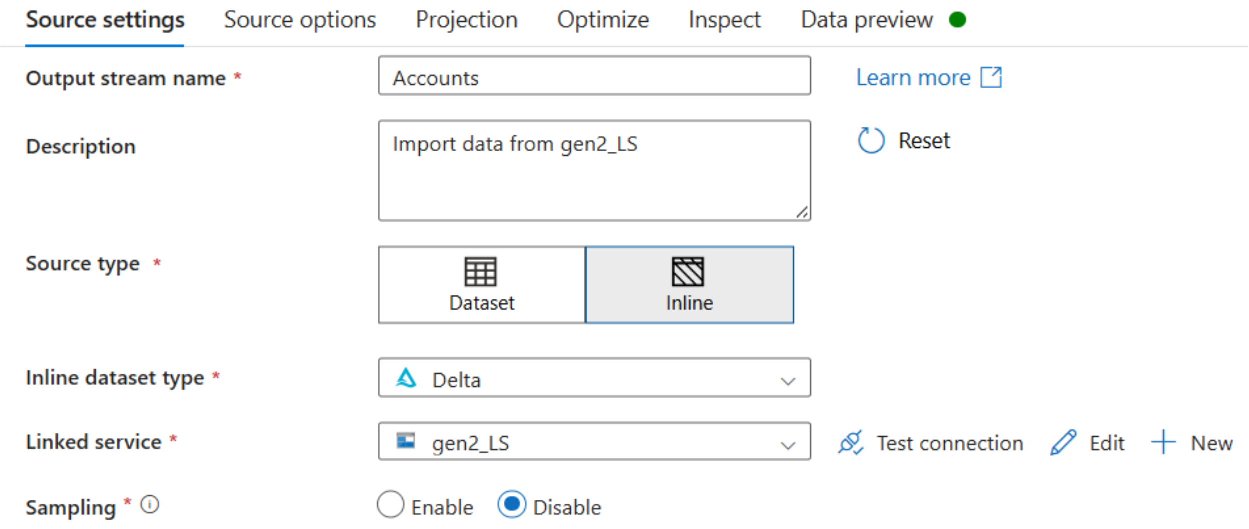
Let’s take accountssscd1 dataflow – scdtype-1:



Dataflow for accountsscd1 – scdtype-1

The image above shows the dataflow for scdtype-1 that we have to build.

First, let’s configure our source. The images are provided below which show the configuration.



Source settings | Source options | Projection | Optimize | Inspect | Data preview ●

Output stream name *: Accounts

Description: Import data from gen2_LS

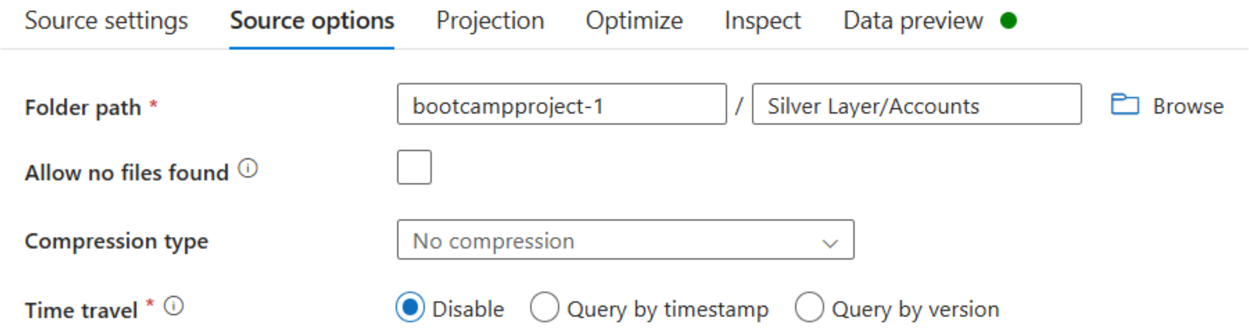
Source type *: Dataset | Inline (selected)

Inline dataset type *: Delta

Linked service *: gen2_LS

Sampling *: ○ Enable ● Disable

Here, inline dataset type is selected as “delta” because we had stored the cleaned data in delta format in our silver layer folder. Now we’re using that data for scdtype-1 and scdtype-2 logic.



Source settings | Source options | Projection | Optimize | Inspect | Data preview ●

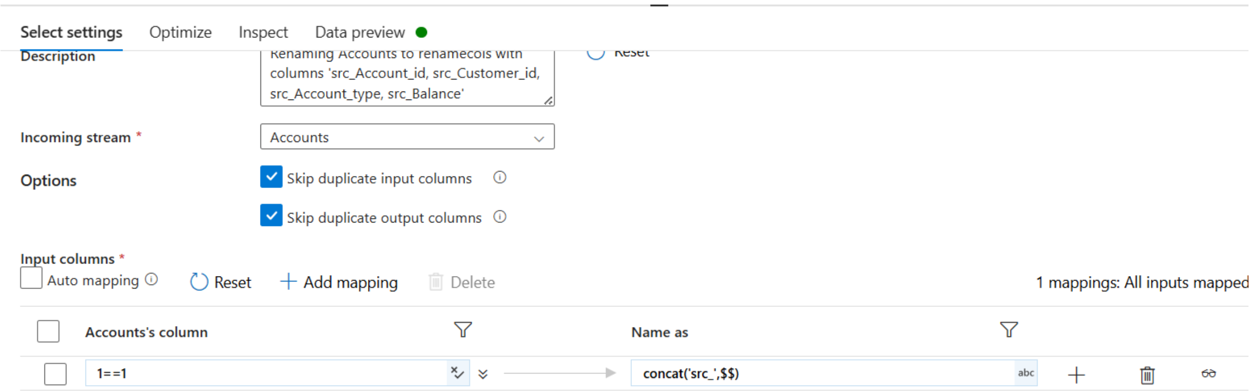
Folder path *: bootcampproject-1 / Silver Layer/Accounts

Allow no files found: □

Compression type: No compression

Time travel *: ● Disable ○ Query by timestamp ○ Query by version

Then we connect the source to a select transformation, which is used to rename columns by concatenating “src_” to all the column names.



Select settings | Optimize | Inspect | Data preview ●

Description: Renaming Accounts to renamecols with columns 'src_Account_id, src_Customer_id, src_Account_type, src_Balance'

Incoming stream *: Accounts

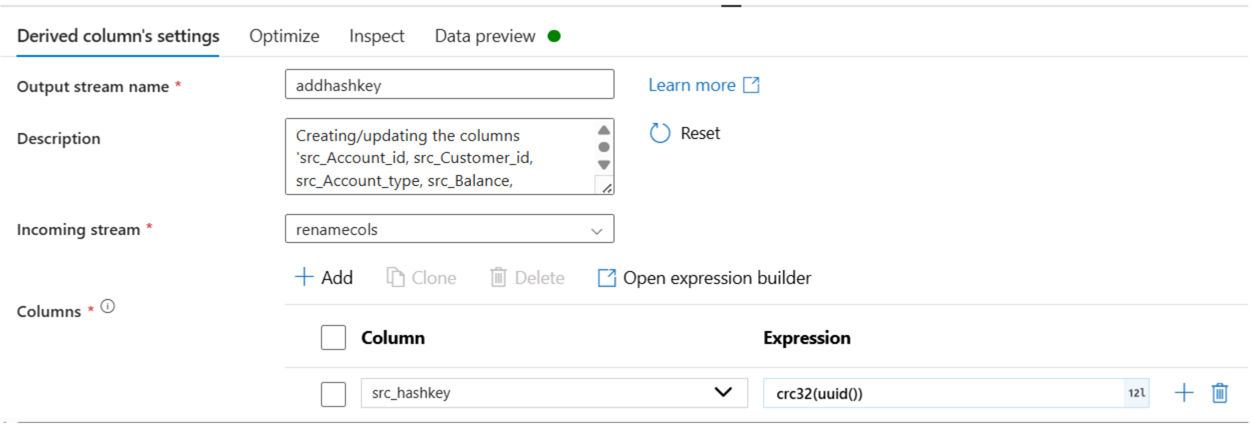
Options: ☑ Skip duplicate input columns ☑ Skip duplicate output columns

Input columns *: □ Auto mapping

1 mappings: All inputs mapped

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

Next, we connect a derived column transformation to add a hashkey column called "src_hashkey" using a hash function called crc32() and calling another function within it called uuid() which helps in generating random hashkeys.



Output stream name * addhashkey

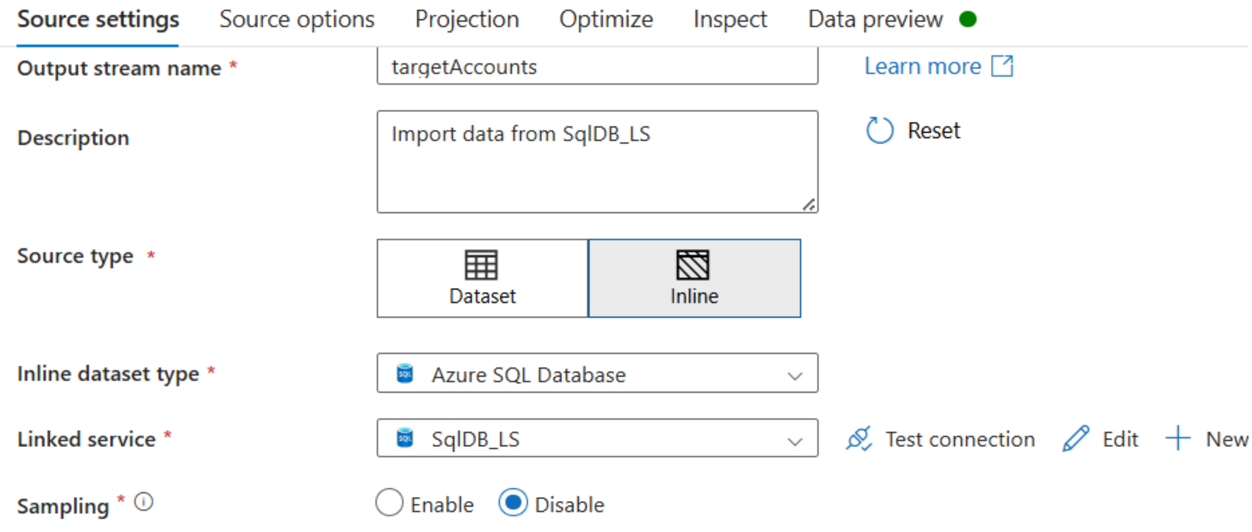
Description: Creating/updating the columns 'src_Account_id, src_Customer_id, src_Account_type, src_Balance,

Incoming stream * renamecols

Columns *

	Column	Expression
□	src_hashkey	crc32(uuid())

Then, we'll create our target (a second source) – where we want to store our data essentially – only this target will be used for a left join that we need to implement in our logic in the next step. Our target will be the azure sql database because that's where we'll eventually store the data.



Output stream name * targetAccounts

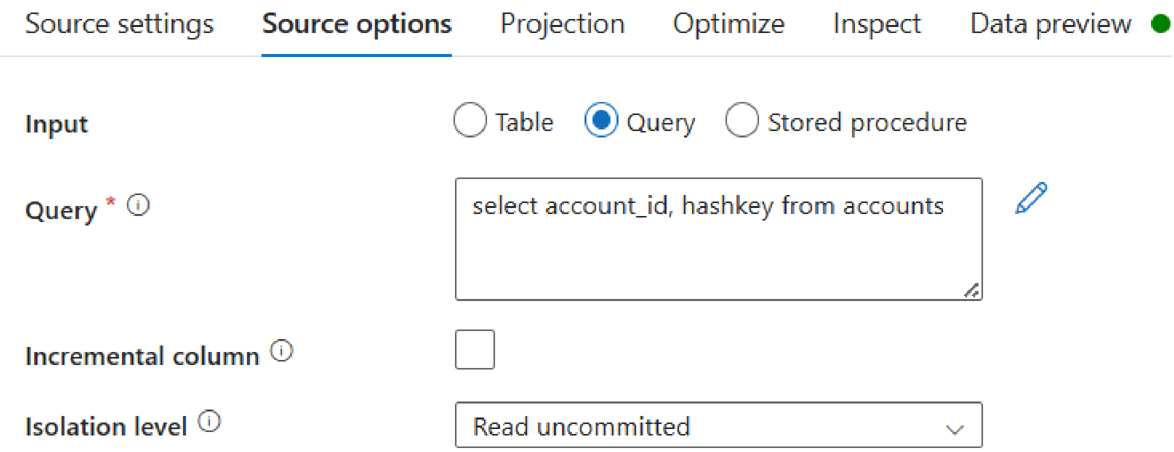
Description: Import data from SqlDB_LS

Source type * Dataset / Inline (selected: Inline)

Inline dataset type * Azure SQL Database

Linked service * SqlDB_LS

Sampling * ○ Enable ● Disable



Input: ○ Table ● Query ○ Stored procedure

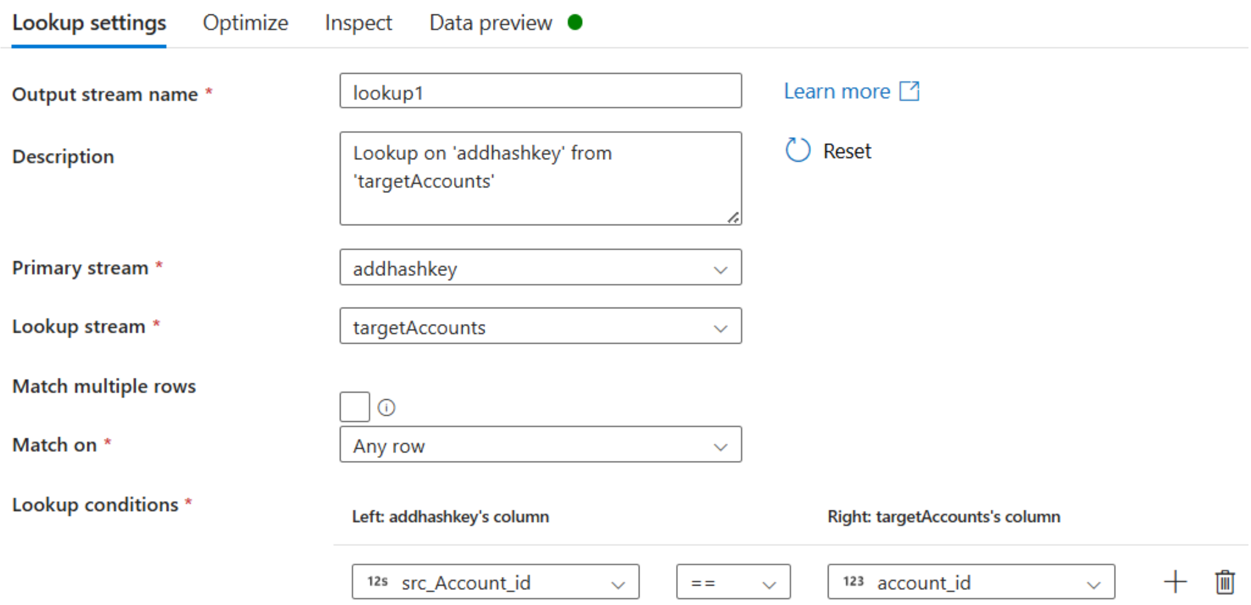
Query *: select account_id, hashkey from accounts

Incremental column: □

Isolation level: Read uncommitted

We only need to compare the key column and hashkey column with our source file so those are the only 2 columns needed in our target.

Now, we'll connect a lookup transformation to our derived column transformation to implement a left join between source and target on the column "account_id".



Output stream name *: lookup1

Description: Lookup on 'addhashkey' from 'targetAccounts'

Primary stream *: addhashkey

Lookup stream *: targetAccounts

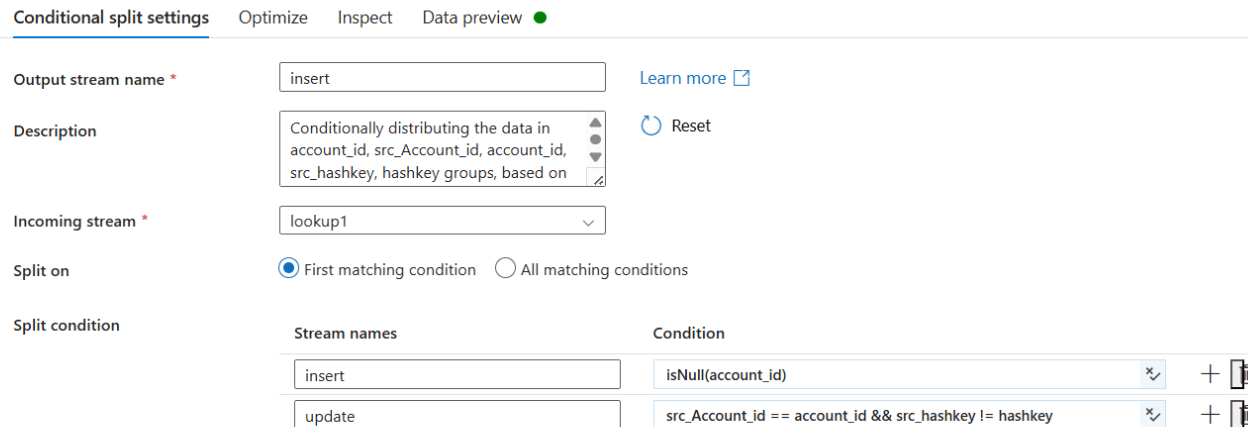
Match multiple rows: □

Match on *: Any row

Lookup conditions *:

Left: addhashkey's column		Right: targetAccounts's column
12S src_Account_id	==	123 account_id

Next, we'll connect a conditional split to this lookup transformation which will help us to either insert or update data in our sink based on certain conditions.



Output stream name *: insert

Description: Conditionally distributing the data in account_id, src_Account_id, account_id, src_hashkey, hashkey groups, based on

Incoming stream *: lookup1

Split on: ● First matching condition ○ All matching conditions

Split condition

Stream names	Condition
insert	isNull(account_id)
update	src_Account_id == account_id && src_hashkey != hashkey

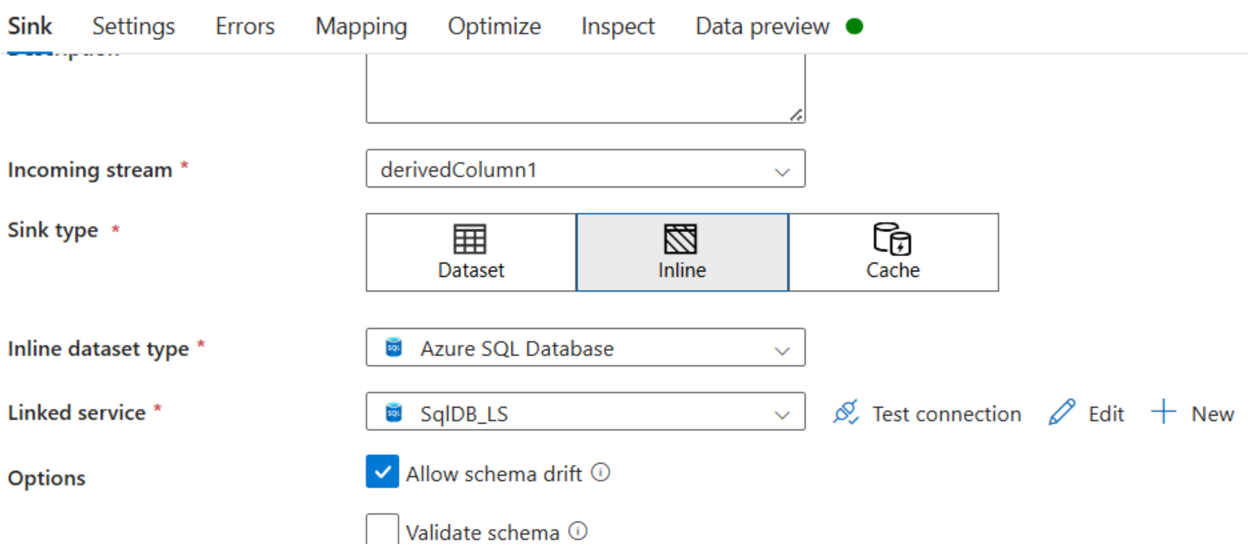
As shown in the image above, we'll add 2 conditional streams – insert and update. Condition for insert – if account_id is NULL in target/sink, that means our source file has an ID which does not exist in our target/sink, so it needs to be inserted.

Condition for update – if an update needs to be made, that means a record must already exist in the target/sink, so we'll check if the IDs match or not. Also, hashkeys come into play here – if a change is made, the hashkey changes, so we have to ensure that hashkeys of source record and target record do not match.

Insert stream:

This stream is then connected to a derived column transformation which is used to enter the values of src_createdby, src_createddate, src_updatedby, src_updateddate columns (which we had created while creating the tables in our sql database) as 'dataflow', currentTimestamp(), 'dataflow', currentTimestamp() respectively. We write 'dataflow' just for us to understand in the output that a particular change was made from the insert stream part of our dataflow. CurrentTimestamp() function records the date and time of any changes that will be made.

Finally, it is connected to a sink – where we'll store our data – so this will be same as target, ie, sql database.



Incoming stream * derivedColumn1

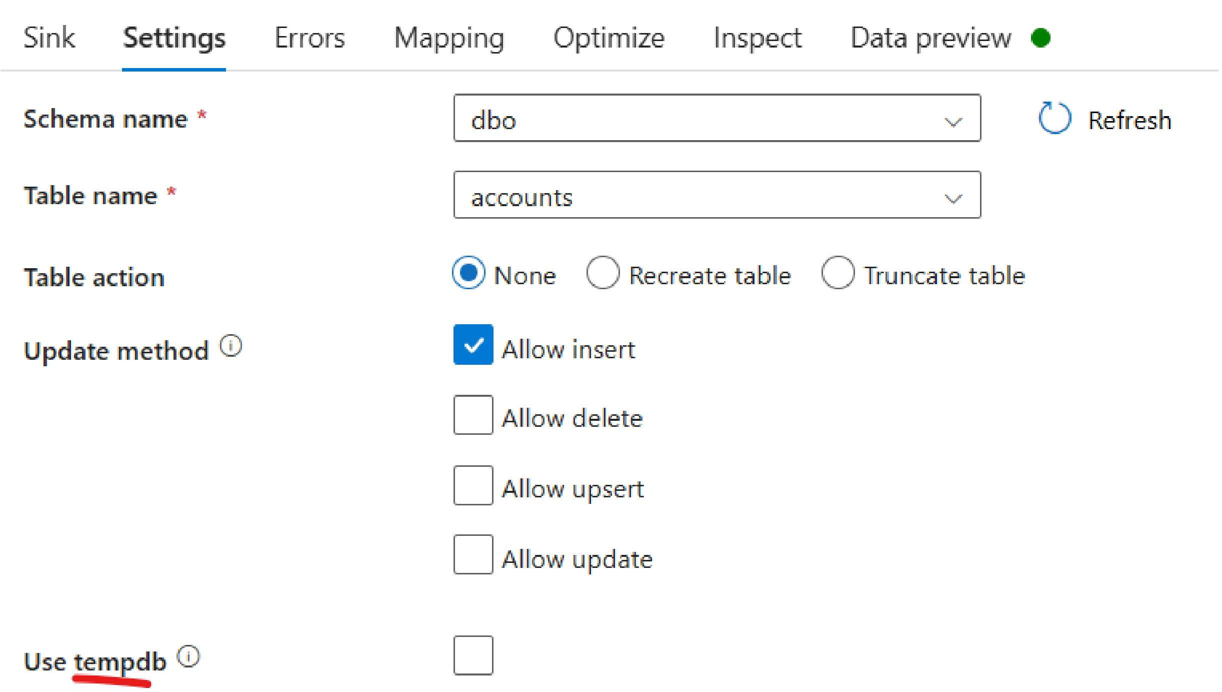
Sink type * Dataset | **Inline** | Cache

Inline dataset type * Azure SQL Database

Linked service * SqlDB_LS

Options ☑ Allow schema drift

□ Validate schema



Schema name * dbo

Table name * accounts

Table action ● None ○ Recreate table ○ Truncate table

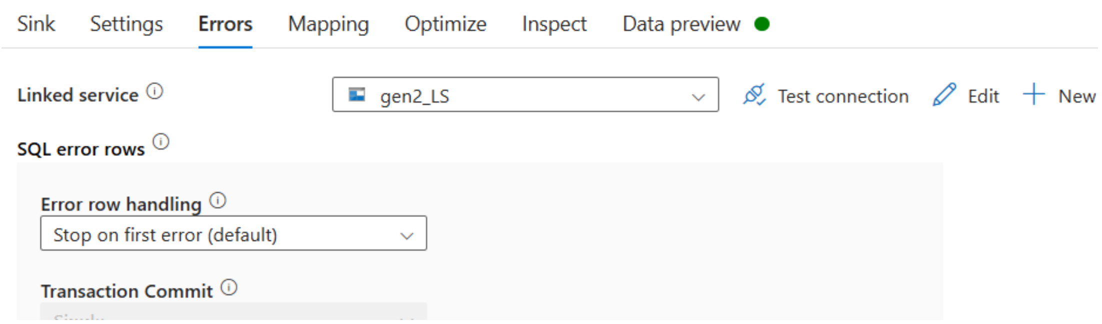
Update method ☑ Allow insert

□ Allow delete

□ Allow upsert

□ Allow update

Use tempdb □

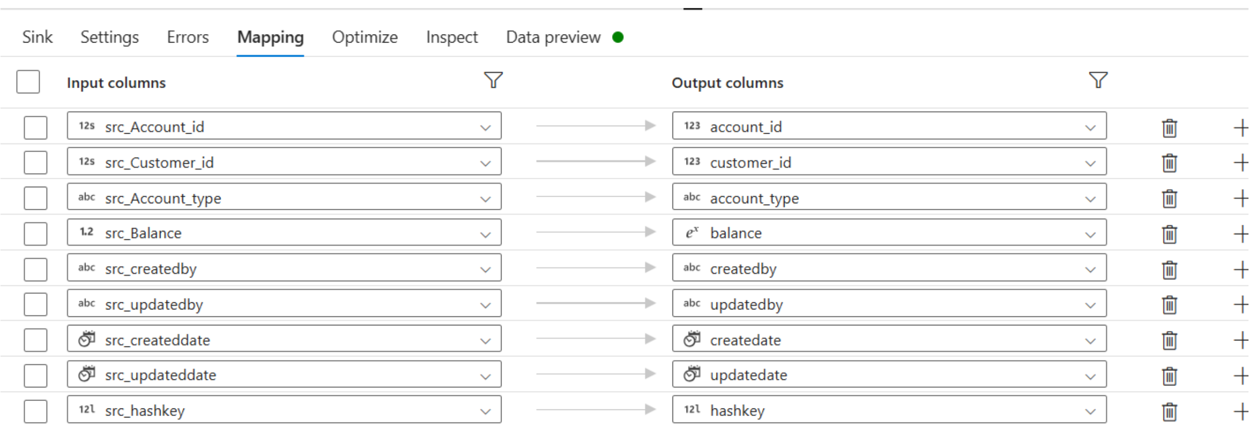


Linked service gen2_LS

SQL error rows

Error row handling: Stop on first error (default)

Transaction Commit

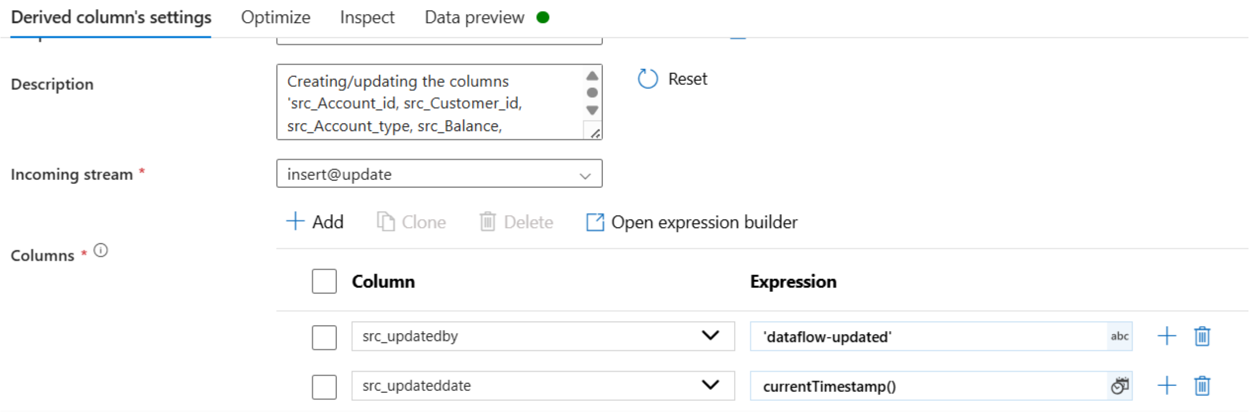


Input columns	Output columns
src_Account_id	account_id
src_Customer_id	customer_id
src_Account_type	account_type
src_Balance	balance
src_createdby	createdby
src_updatedby	updatedby
src_createddate	createdate
src_updateddate	updatedate
src_hashkey	hashkey

Under "mapping" section of the sink, first delete all the mappings, import schema and then reset mappings. Then map all the columns from source to target.

Update stream:

Similarly, update stream is connected to a derived column transformation to insert the values of src_updatedby and src_updateddate columns as 'dataflow-updated' and currentTimestamp() respectively. The logic behind this remains similar to that of what we did in insert stream. We don't need src_createdby and src_createddate as this is the update stream so these values will not be changed here.



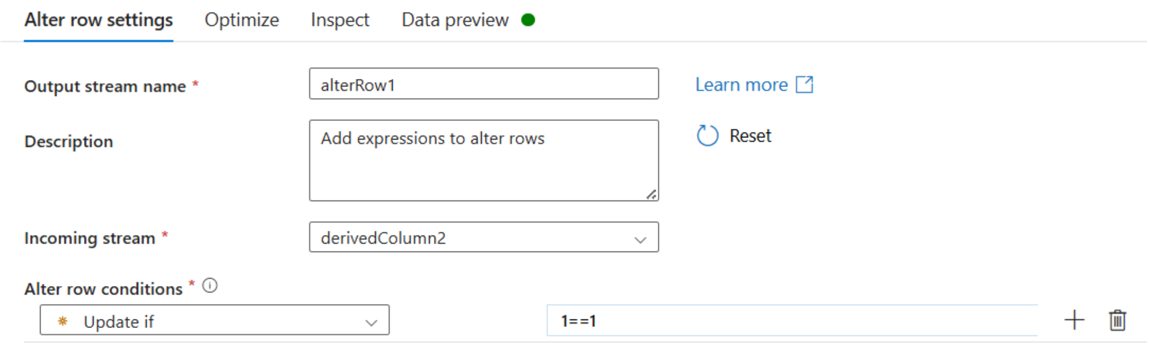
Description: Creating/updating the columns 'src_Account_id, src_Customer_id, src_Account_type, src_Balance,

Incoming stream *: insert@update

Columns *

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

Next, we connect it to an alter row transformation to give it access to our sink, ie, the sql database so the dataflow can update the data.



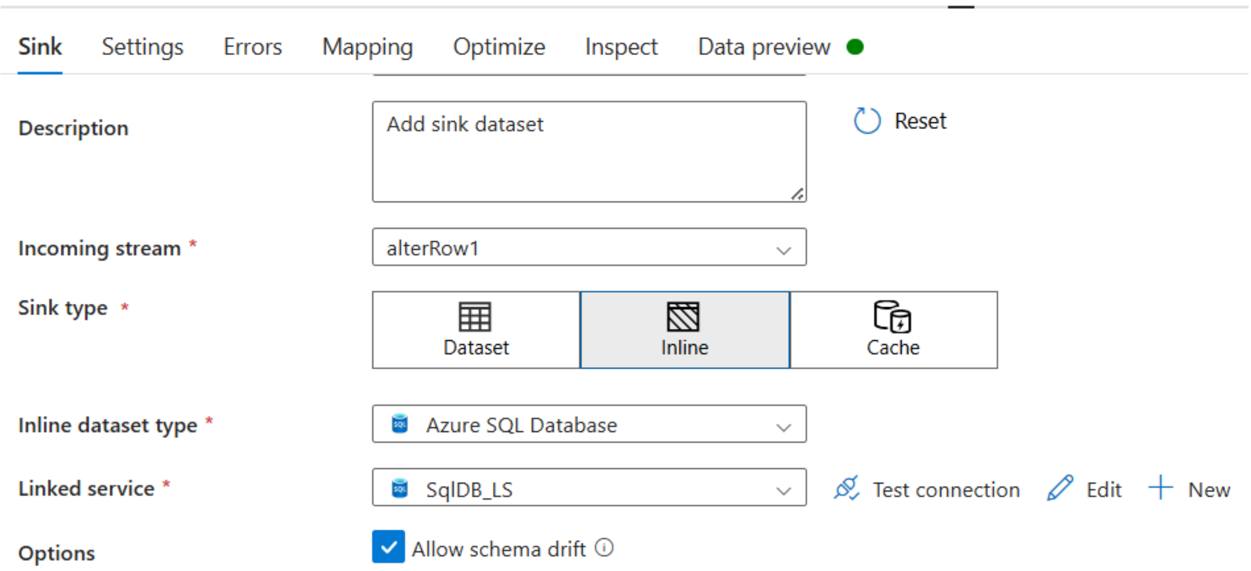
Output stream name * alterRow1

Description: Add expressions to alter rows

Incoming stream * derivedColumn2

Alter row conditions *: Update if 1==1

Finally, we connect it to a sink – same as target, ie, sql database.



Description: Add sink dataset

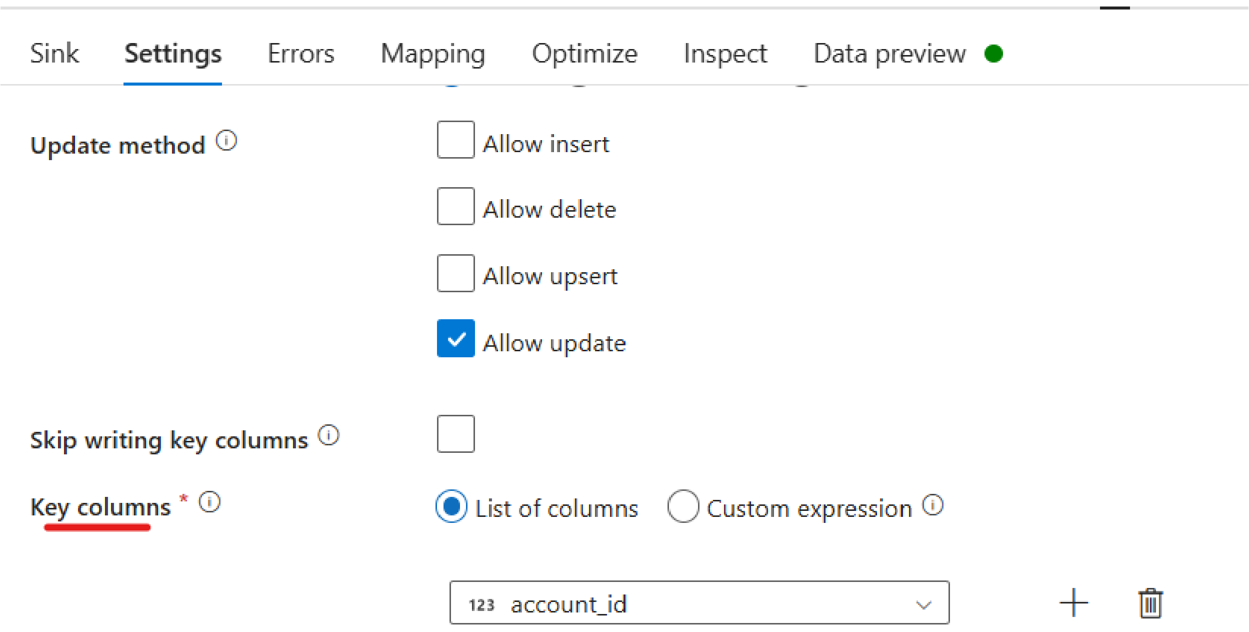
Incoming stream * alterRow1

Sink type * Inline

Inline dataset type * Azure SQL Database

Linked service * SqlDB_LS

Options ☑ Allow schema drift

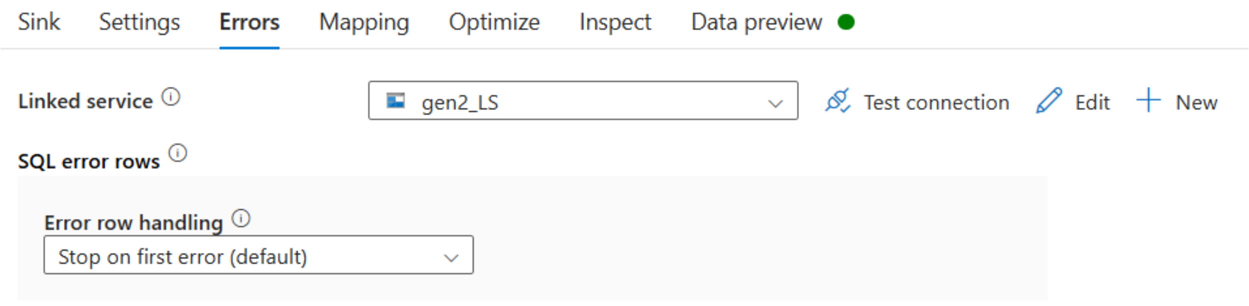


Update method
☐ Allow insert
☐ Allow delete
☐ Allow upsert
☑ Allow update

Skip writing key columns ☐

Key columns *
● List of columns ○ Custom expression

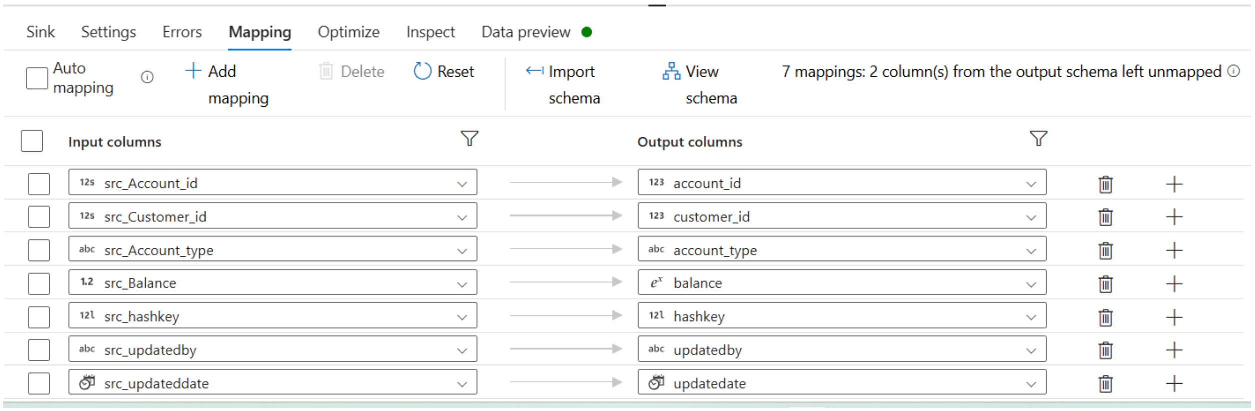
123 account_id



Linked service: gen2_LS

SQL error rows

Error row handling: Stop on first error (default)



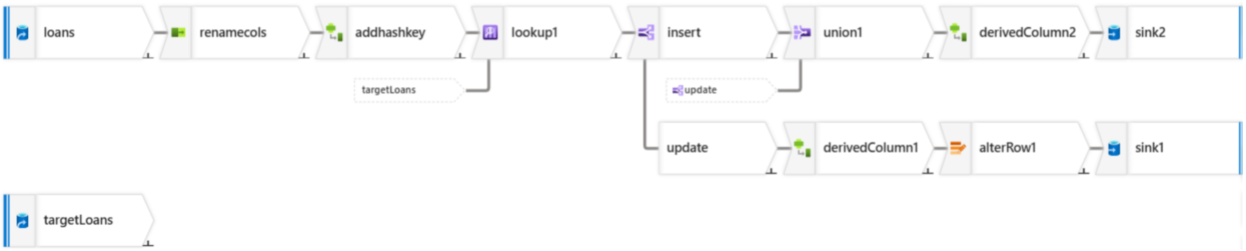
7 mappings: 2 column(s) from the output schema left unmapped

Input columns	Output columns
src_Account_id	account_id
src_Customer_id	customer_id
src_Account_type	account_type
src_Balance	balance
src_hashkey	hashkey
src_updatedby	updatedby
src_updateddate	updatedate

Similar to the insert sink mapping, we'll get rid of all the mappings, import schema and then reset the mappings and map all columns from source to target. Note that created_by and createddate columns are not needed here as those values are not affected in this stream.

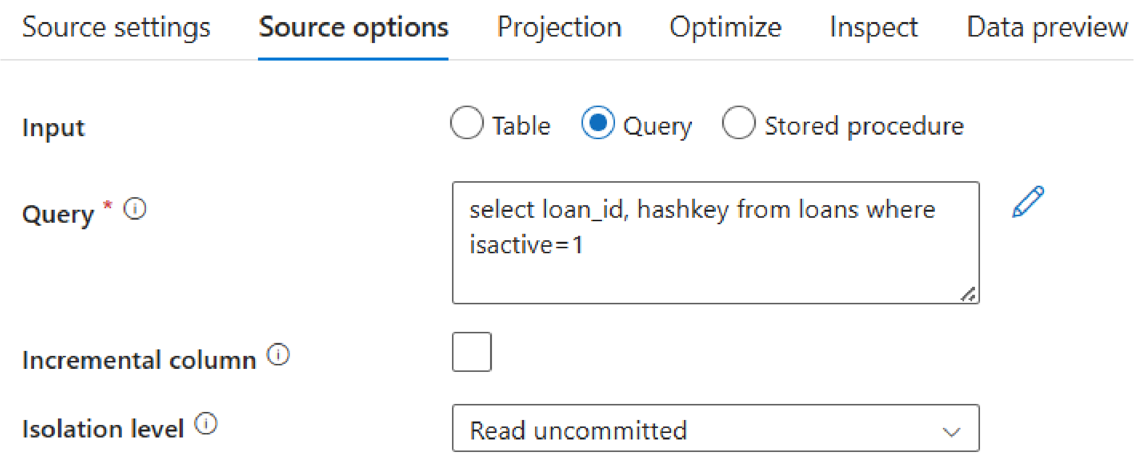
This marks the end of the scdtype-1 dataflow. Similarly, we can clone this dataflow and make necessary changes for the remaining 2 scdtype-1 dataflows.

Let’s discuss scdtype-2 dataflow for loans.csv



It is very similar to scdtype-1 dataflow, in fact, until the conditional split for insert and update streams, the logic remains the same. The changes are as follows:

In target, we select only those records of the key column and hashkey where isactive = 1



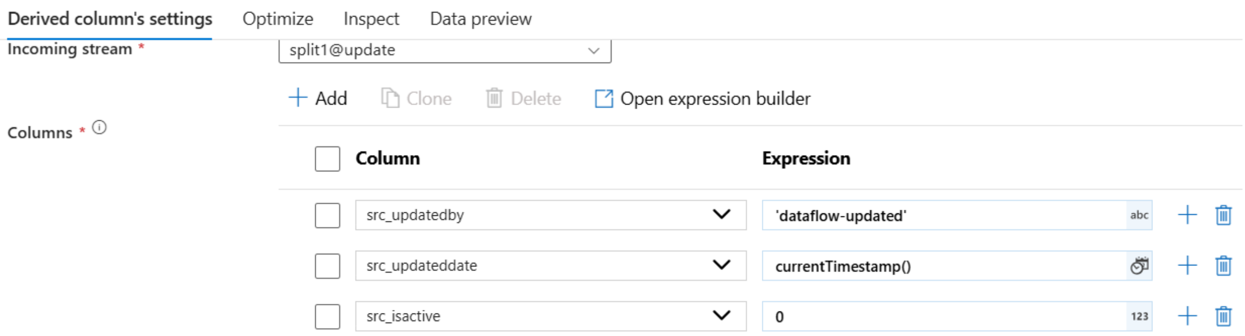
Input ○ Table ● Query ○ Stored procedure

Query * ⓘ select loan_id, hashkey from loans where isactive=1

Incremental column ⓘ □

Isolation level ⓘ Read uncommitted

We first complete the update stream and then complete the insert stream.



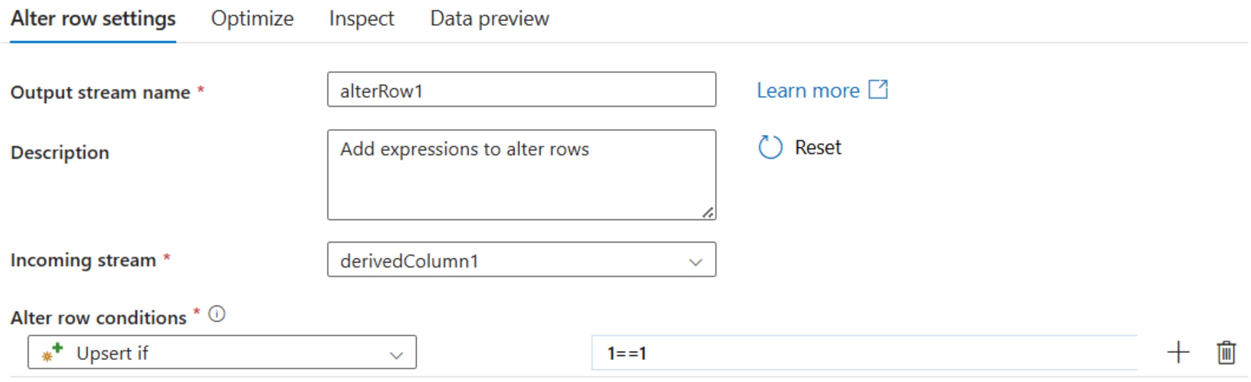
Incoming stream * split1@update

Columns * ⓘ

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

Derived column transformation of update stream

Here, we set the value for “src_isactive” column as zero because once a record is updated, THAT record will become inactive and a new record with the UPDATED DATA will be added in to the target/sink whose status will be active (ie,1 and remember that scdtype-2 logic stores historical data).



Output stream name * alterRow1

Description: Add expressions to alter rows

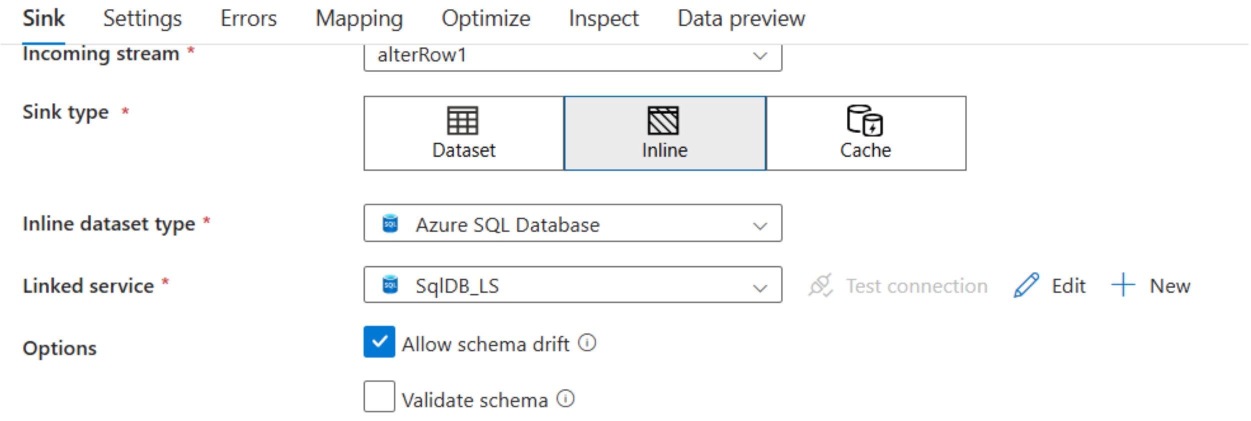
Incoming stream * derivedColumn1

Alter row conditions *

Upsert if 1==1

For scdtype-2 logic, we select “upsert if” in the alter row transformation of update stream because we’ll actually INSERT a new record with UPDATED data into our target/sink, and to do that we need to provide both insert and update permission – which is enabled by upsert.

Then comes the update sink as follows:



Incoming stream * alterRow1

Sink type * Inline

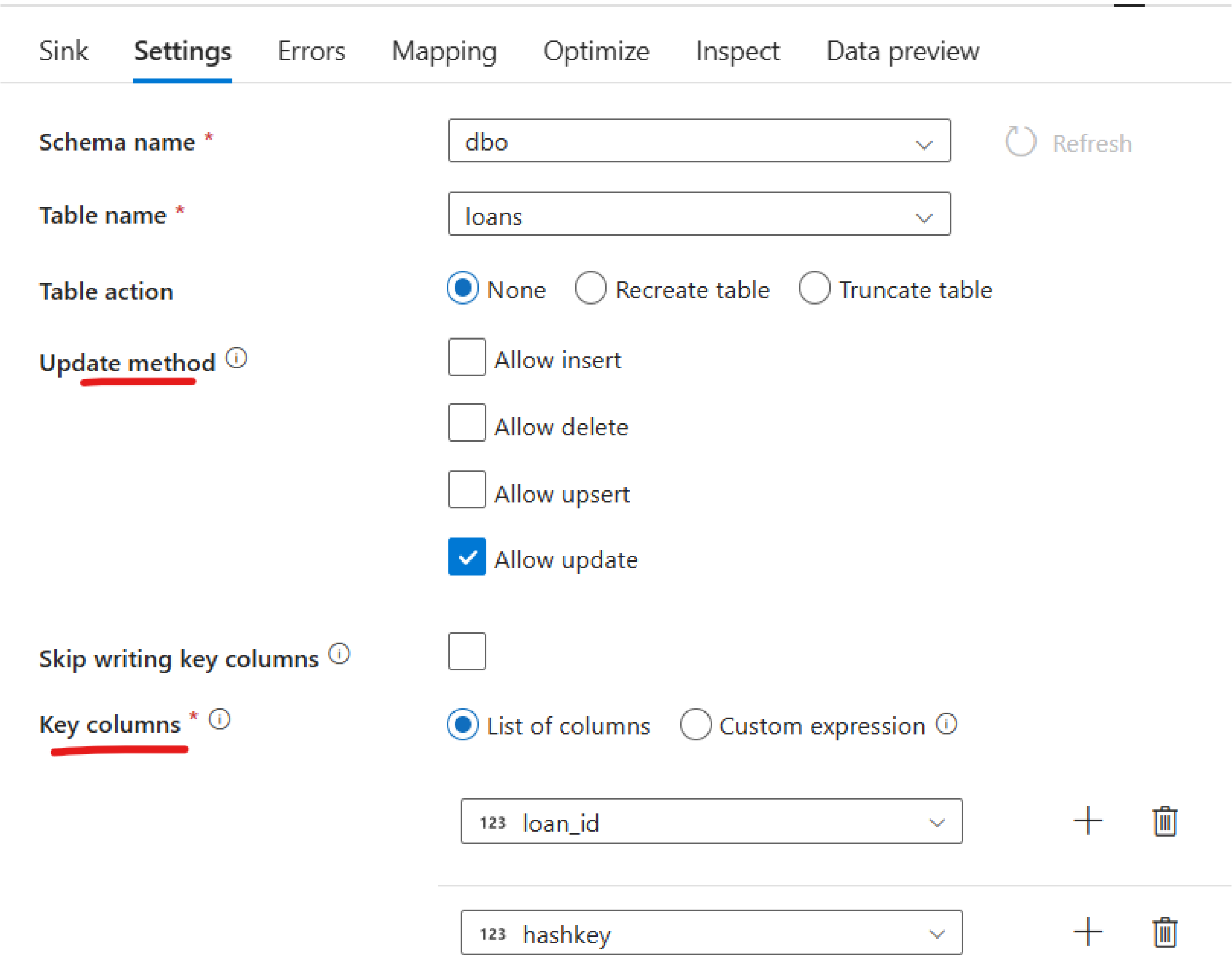
Inline dataset type * Azure SQL Database

Linked service * SqlDB_LS

Options

☑ Allow schema drift

□ Validate schema



Schema name * : dbo

Table name * : loans

Table action: ● None ○ Recreate table ○ Truncate table

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

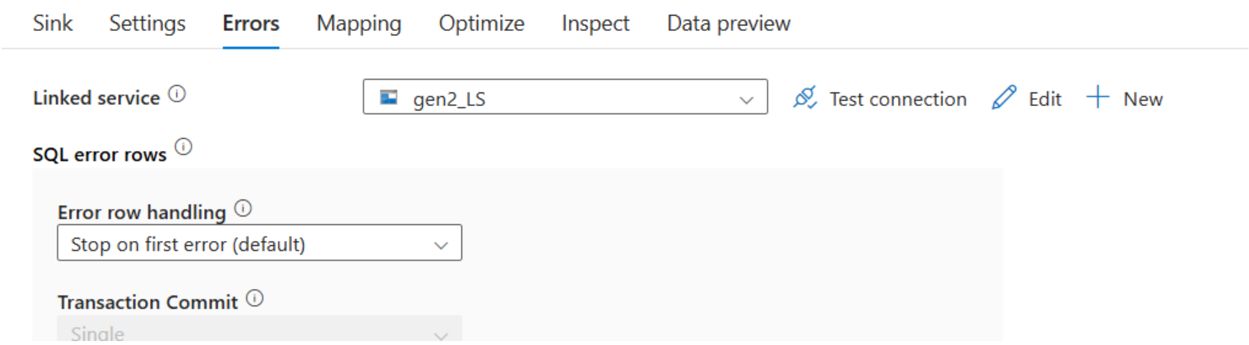
Skip writing key columns ⓘ: □

Key columns * ⓘ: ● List of columns ○ Custom expression ⓘ

123 loan_id

123 hashkey

Note that under sink settings, we'll select "allow update" as through update stream, we'll only be updating data but inserting the data in new record will be done by insert stream, where we'll unionize the output of update stream. Hence, its necessary to give permission to upsert in alter row transformation but only update permission in sink will do the job. Make sure to add the key columns loan_id and hashkey. We add the hashkey as a key column as well because we'll be storing historical (old) data as well, so both the id and hashkey are required to store the old data and once the data is updated, hashkey will change.

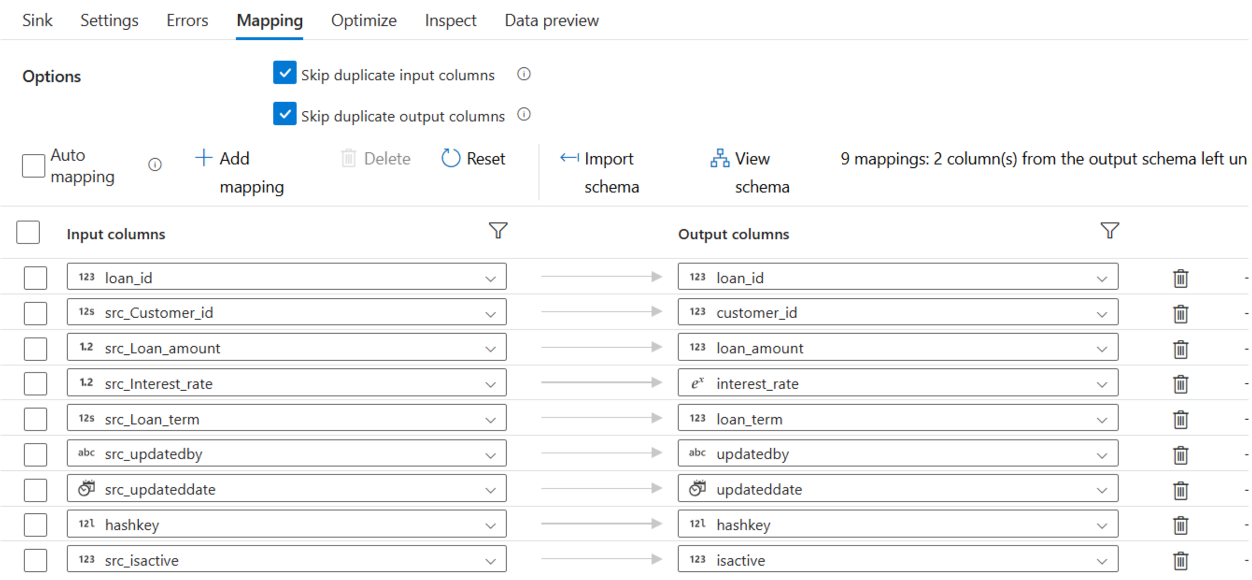


Linked service ⓘ: gen2_LS

SQL error rows ⓘ

Error row handling ⓘ: Stop on first error (default)

Transaction Commit ⓘ: Single



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

Options

☑ Skip duplicate input columns ⓘ

☑ Skip duplicate output columns ⓘ

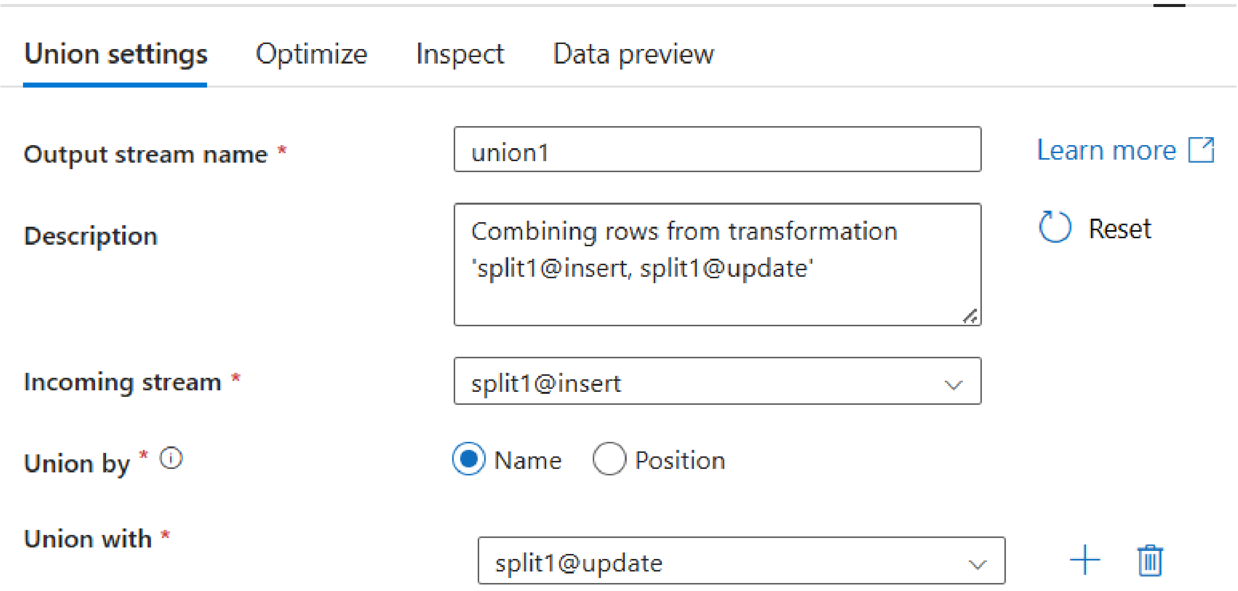
□ Auto mapping ⓘ | + Add mapping | Delete | Reset | Import schema | View schema | 9 mappings: 2 column(s) from the output schema left un

□	Input columns		Output columns
□	123 loan_id	→	123 loan_id
□	12s src_Customer_id	→	123 customer_id
□	1.2 src_Loan_amount	→	123 loan_amount
□	1.2 src_Interest_rate	→	eˣ interest_rate
□	12s src_Loan_term	→	123 loan_term
□	abc src_updatedby	→	abc updatedby
□	src_updateddate	→	updateddate
□	12l hashkey	→	12l hashkey
□	123 src_isactive	→	123 isactive

Similar to scdtype-1 mappings, we'll get rid of all the mappings initially and reset them after importing the schema and finally map all the columns from source to target. Here, the only exceptions will be key column (loan_id) and hashkey column will be mapped from target to target.

Insert stream for scdtype-2:

We'll first connect the split with a union transformation.



Union settings | Optimize | Inspect | Data preview

Output stream name *: union1 | Learn more

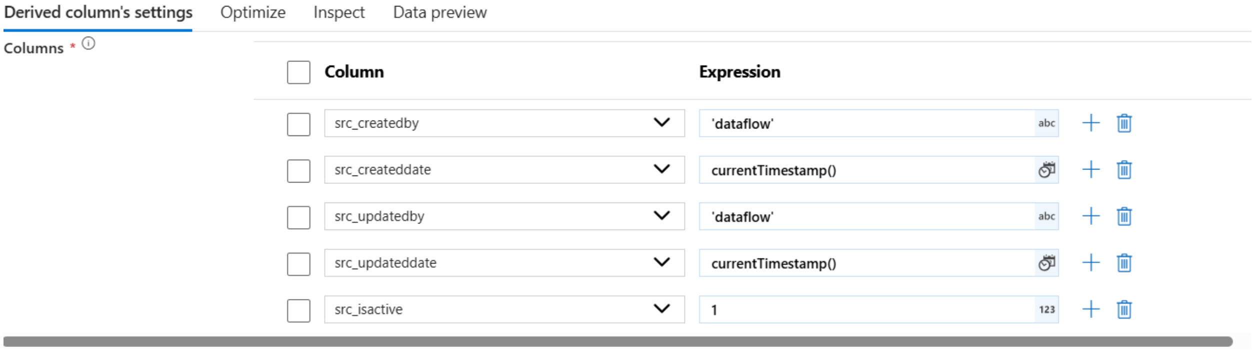
Description: Combining rows from transformation 'split1@insert, split1@update' | Reset

Incoming stream *: split1@insert

Union by * ⓘ: ● Name ○ Position

Union with *: split1@update | +

Then comes the derived column transformation. Here, src_isactive will be 1 because whenever a new record is inserted, it'll always be active(1) initially until a change is made to that record.

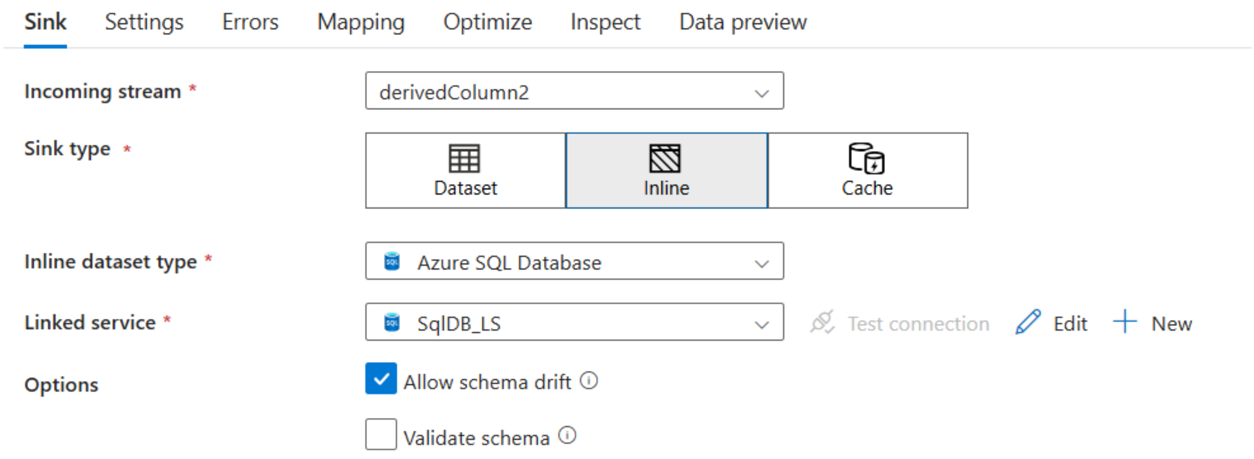


Derived column's settings | Optimize | Inspect | Data preview

Columns *

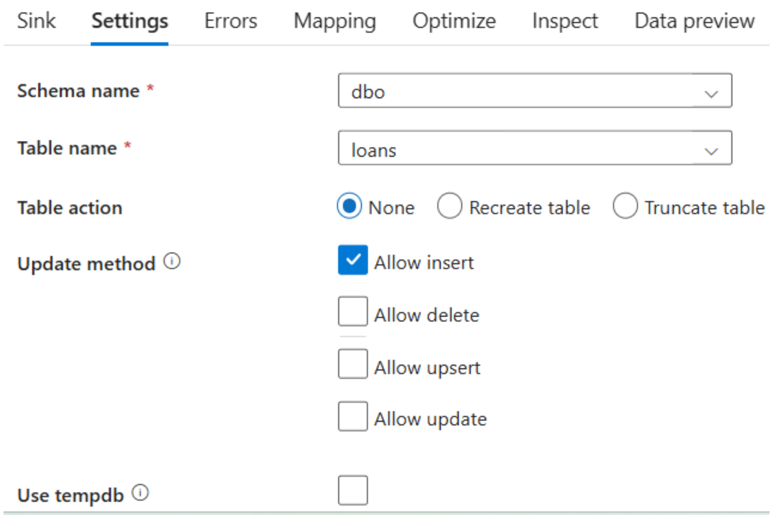
Column	Expression
src_createdby	'dataflow'
src_createddate	currentTimestamp()
src_updatedby	'dataflow'
src_updateddate	currentTimestamp()
src_isactive	1

In the end comes the sink for input stream of scdtype-2



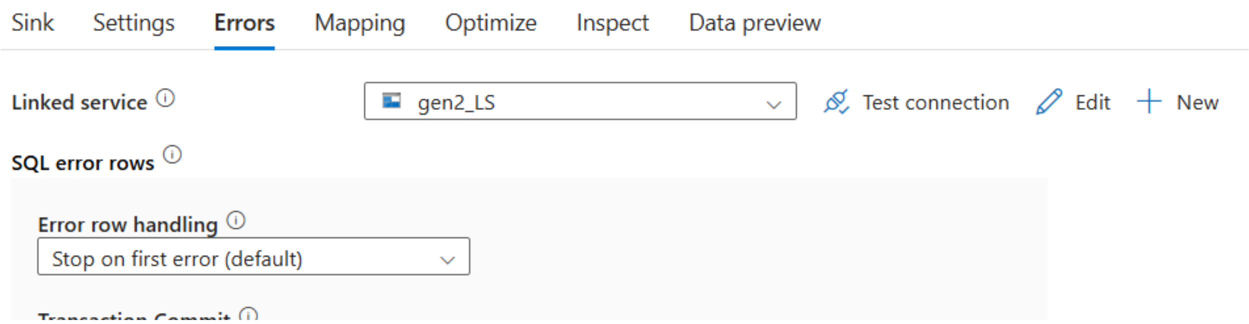
Sink | Settings | Errors | Mapping | Optimize | Inspect | Data preview

Incoming stream *	derivedColumn2
Sink type *	Dataset / **Inline** / Cache
Inline dataset type *	Azure SQL Database
Linked service *	SqlDB_LS
Options	☑ Allow schema drift □ Validate schema



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

Schema name *	dbo
Table name *	loans
Table action	● None ○ Recreate table ○ Truncate table
Update method	☑ Allow insert □ Allow delete □ Allow upsert □ Allow update
Use tempdb	□

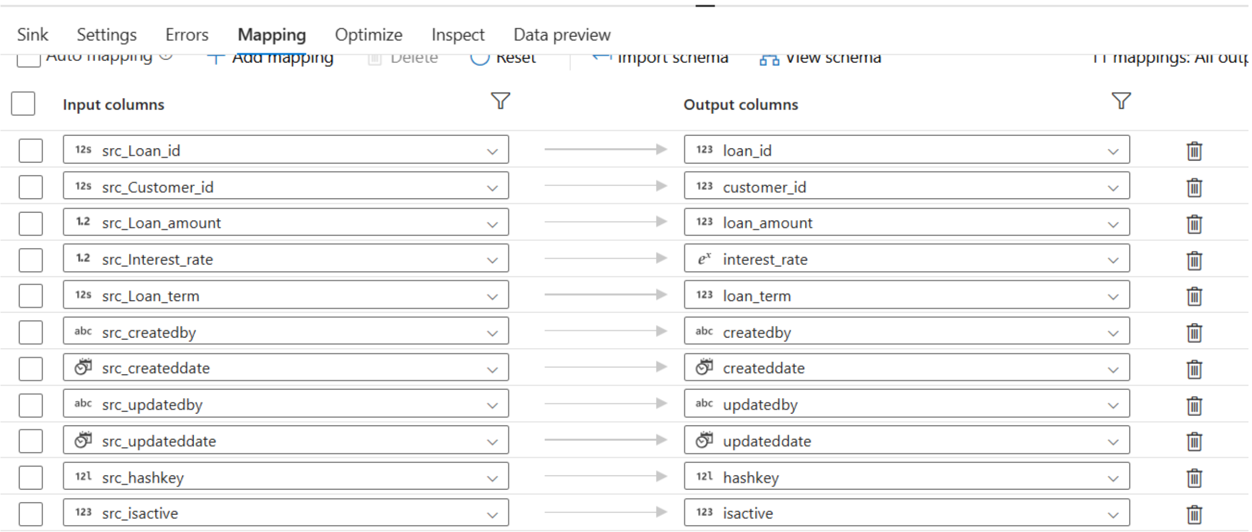


Linked service: gen2_LS

SQL error rows

Error row handling: Stop on first error (default)

Transaction Commit

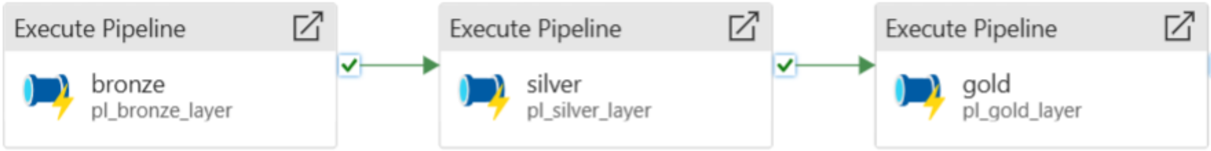


Input columns	Output columns
src_Loan_id	loan_id
src_Customer_id	customer_id
src_Loan_amount	loan_amount
src_Interest_rate	interest_rate
src_Loan_term	loan_term
src_createdby	createdby
src_createddate	createddate
src_updatedby	updatedby
src_updateddate	updateddate
src_hashkey	hashkey
src_isactive	isactive

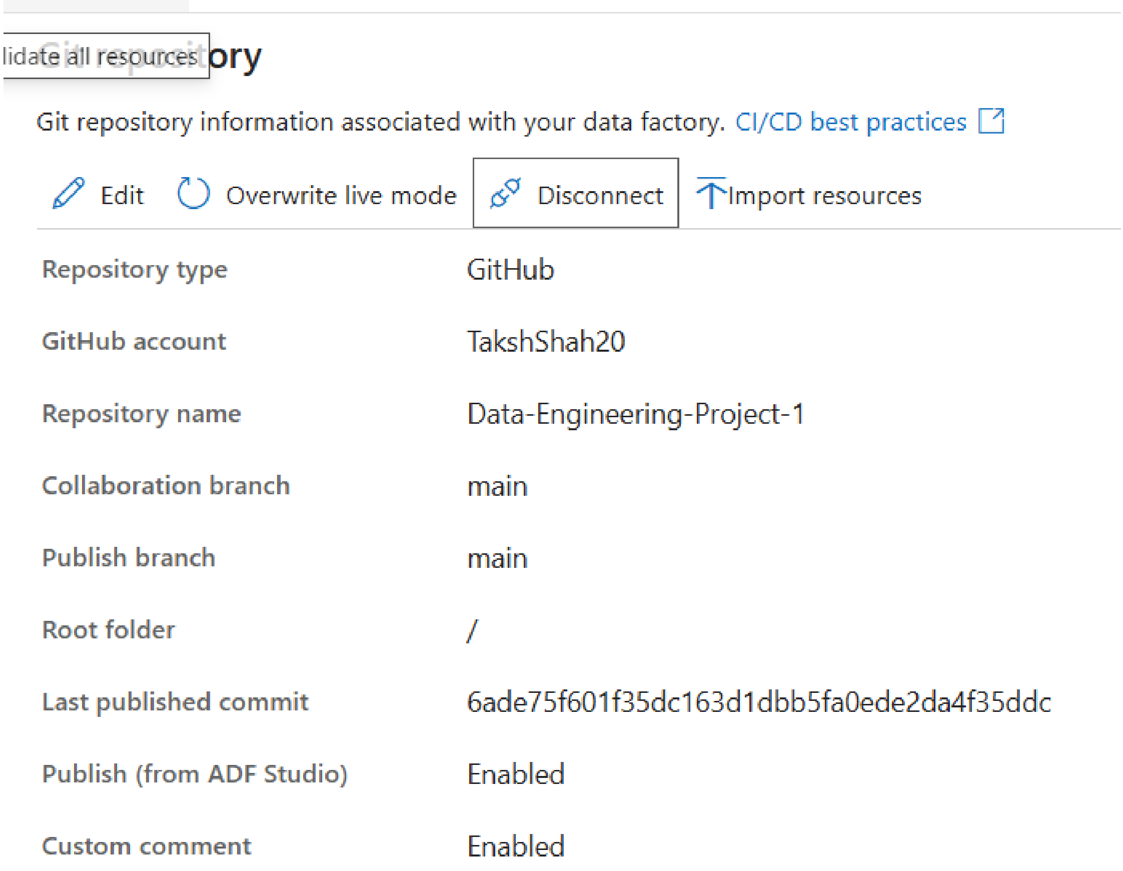
All columns are mapped from source to target.

This marks the end of scdtype-2 dataflow. Similary, this can be cloned to make the remaining scdtype-2 dataflow with necessary changes.

Finally, we connect everything in a master pipeline where we invoke the bronze layer pipeline followed by silver layer pipeline and finally gold layer pipeline and connect them in the same fashion.



In order to publish this project to github, create a new repository in github and connect it from the manage -> git configuration section in azure data factory in azure portal.



Git repository information associated with your data factory. CI/CD best practices

Repository type	GitHub
GitHub account	TakshShah20
Repository name	Data-Engineering-Project-1
Collaboration branch	main
Publish branch	main
Root folder	/
Last published commit	6ade75f601f35dc163d1dbb5fa0ede2da4f35ddc
Publish (from ADF Studio)	Enabled
Custom comment	Enabled