

BOOTCAMP PROJECT 4

**INCREMENTAL DATA LOADING AND AUTOMATED
NOTIFICATIONS USING MICROSOFT FABRIC**

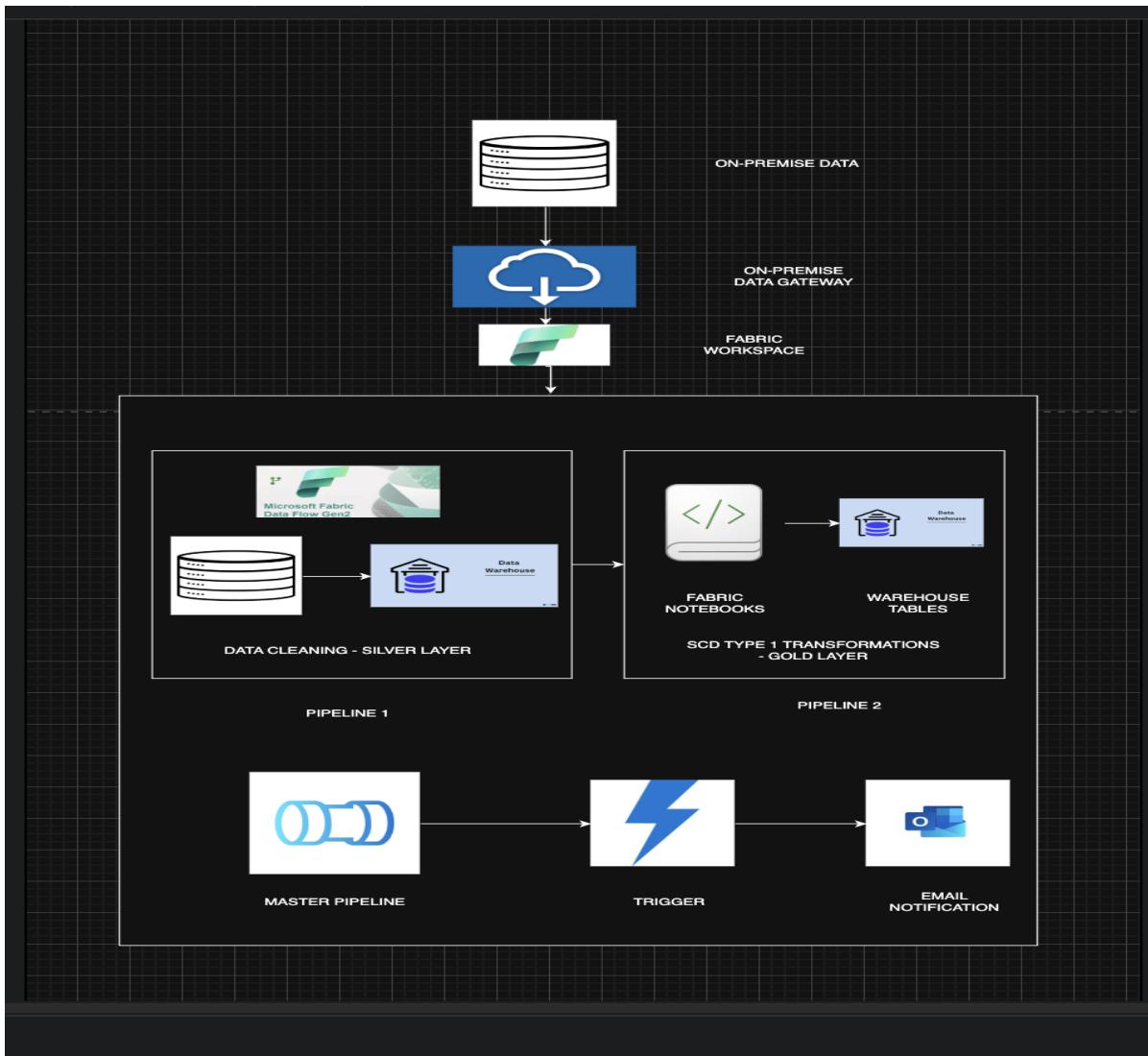
SUBMITTED BY DRUSYA SURESH

OBJECTIVE:

To build an end-to-end data pipeline on Microsoft Fabric that:

1. Ingests structured data from on-premises environments into a Fabric Lakehouse using the On-Prem Gateway.
2. Utilizes the AI Bank Dataset as the source.
3. Implements Dataflow Gen 1 to join tables, remove duplicates, and clean data
- Loads the cleansed data into a Fabric Warehouse
4. Applies Slowly Changing Dimension (SCD) Type 1 logic using Fabric Notebooks and writes the results into separate warehouse tables
5. Schedules and monitors the pipeline, sending an automated email notification (via Outlook or Gmail) upon successful pipeline completion.

ARCHITECTURE DIAGRAM:



STEPS INVOLVED:

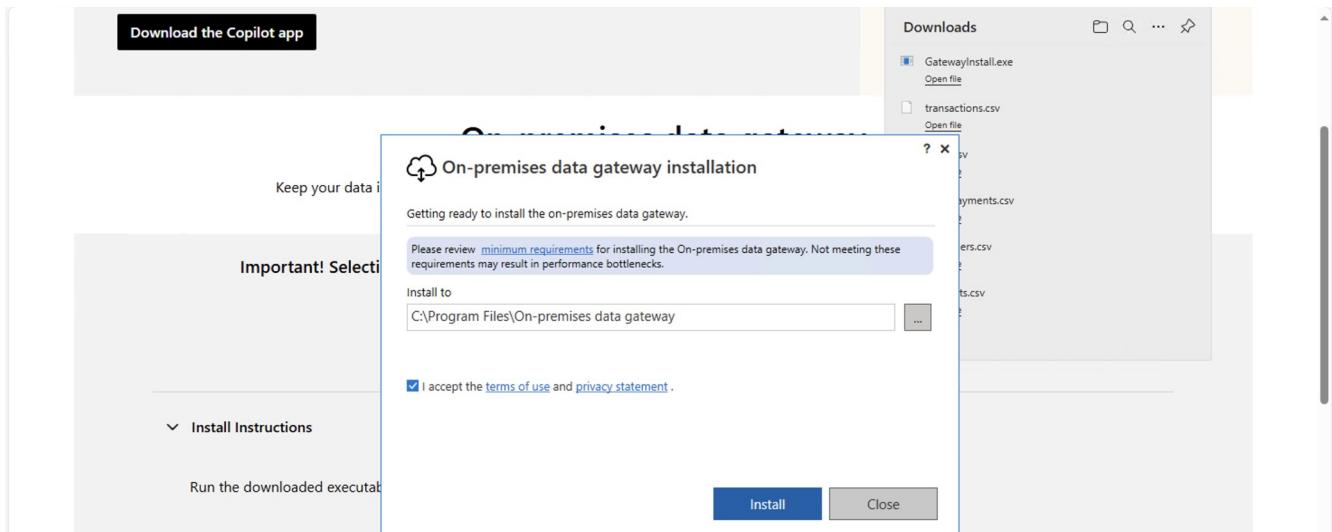
- Connect Microsoft Fabric with on-premise environment using the virtual machine and on-prem gateway.
- Copy the data in the on-prem to the data lake using pipeline activity and load them into lake house tables – Bronze layer.
- Using data flow activity clean and standardize the data and load them to data warehouse – Silver layer.
- Perform SCD Type 1 logic on the cleaned data using Fabric notebooks and load them into separate tables in warehouse – Gold layer.
- Using a master pipeline trigger all the activities so that can schedule to run it on specific time and also set an email notification upon successful pipeline completion.

TOOLS AND TECHNOLOGIES USED:

- Microsoft Fabric
- On-Premises Data Gateway
- Fabric Lakehouse and Warehouse
- Fabric Dataflow Gen 1
- Fabric Notebook
- Email Notification Task (in-built).

CONNECTING ON-PREMISE WITH FABRIC WORKSPACE USING ON-PREM DATA GATEWAY

For this I am using virtual machine to download the on-prem data gateway. I have my data in the virtual machine downloads folder.



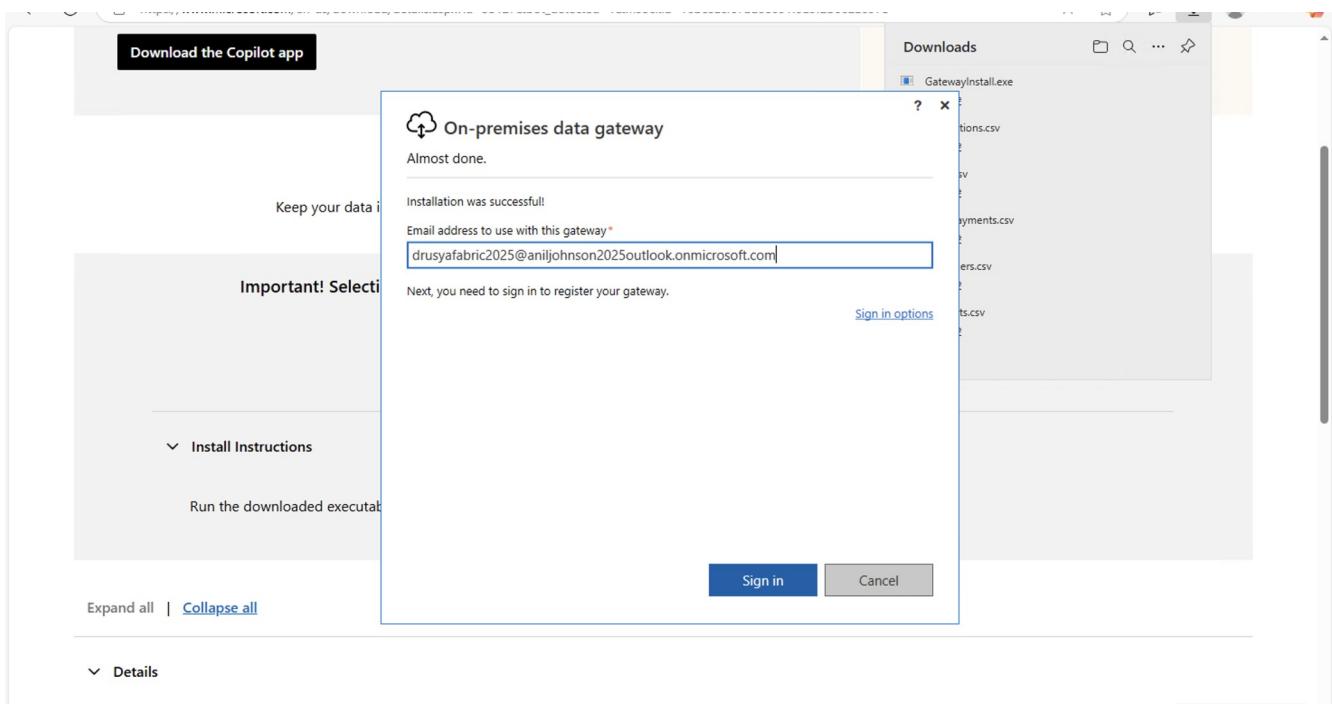
[Expand all](#) | [Collapse all](#)

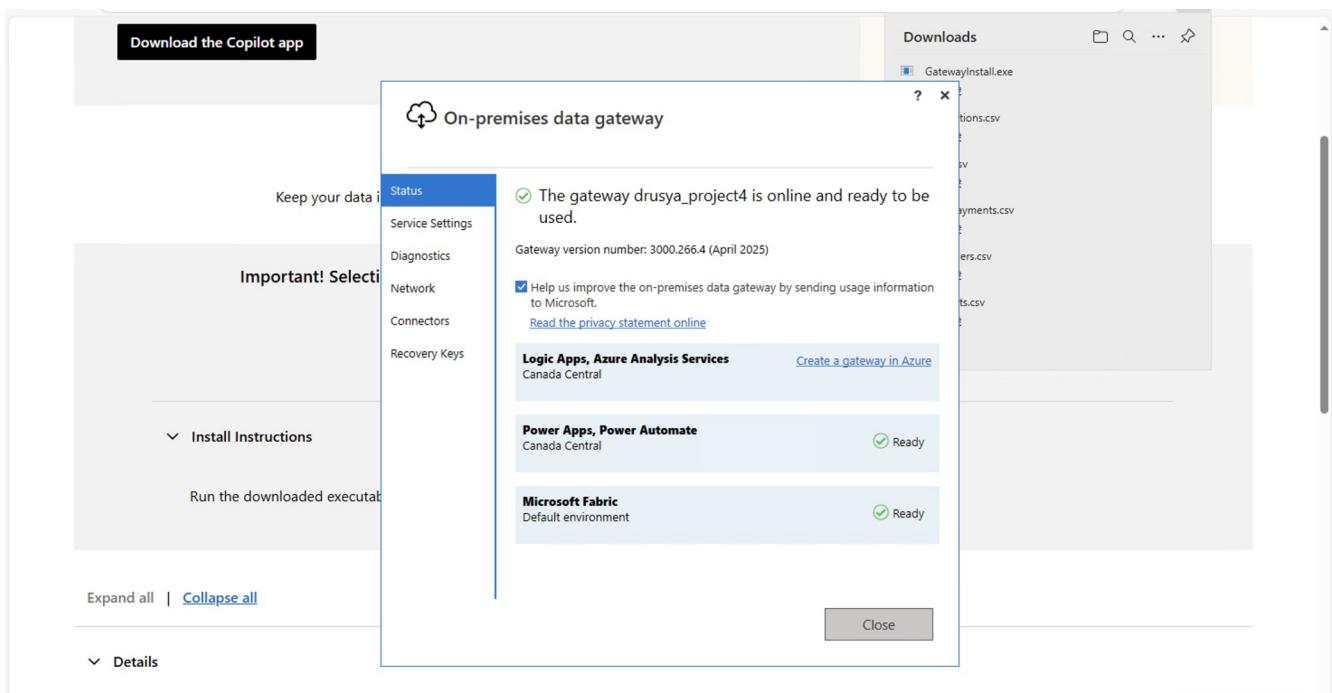
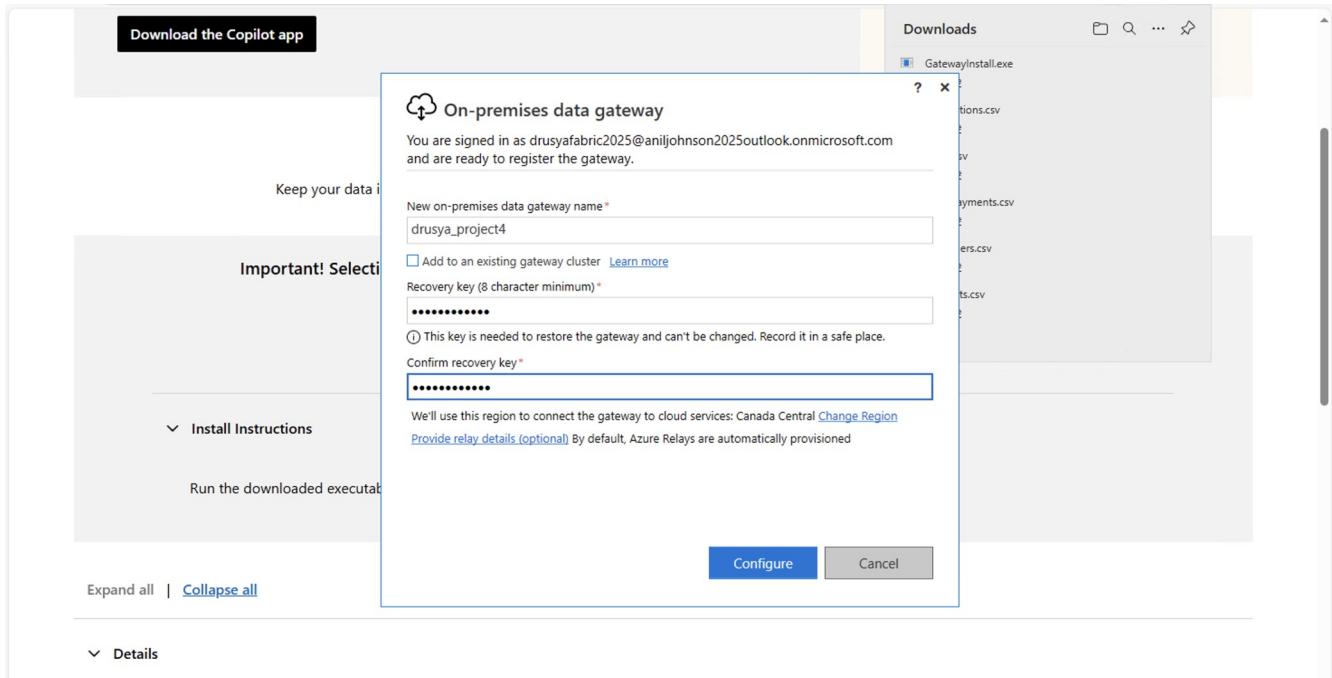
[▼ Details](#)

Version:

Date Published:

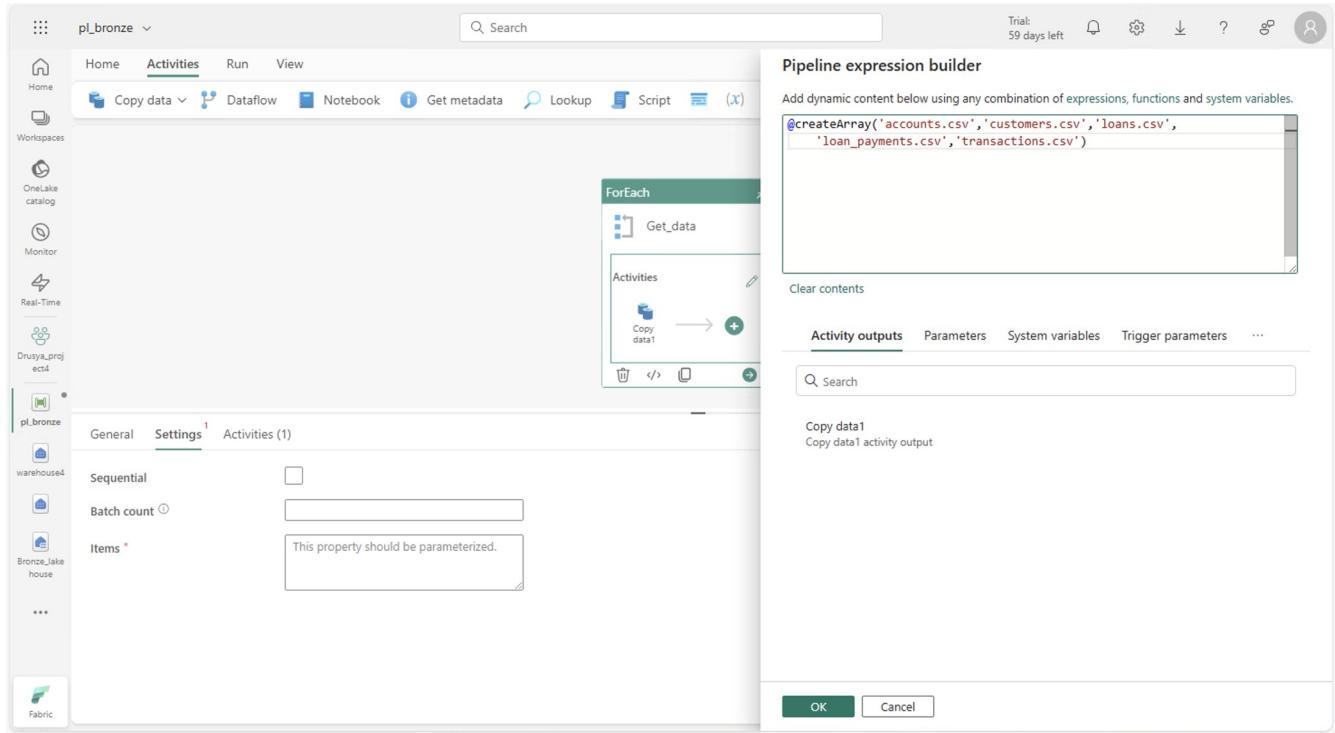
[↑ Back To Top](#)





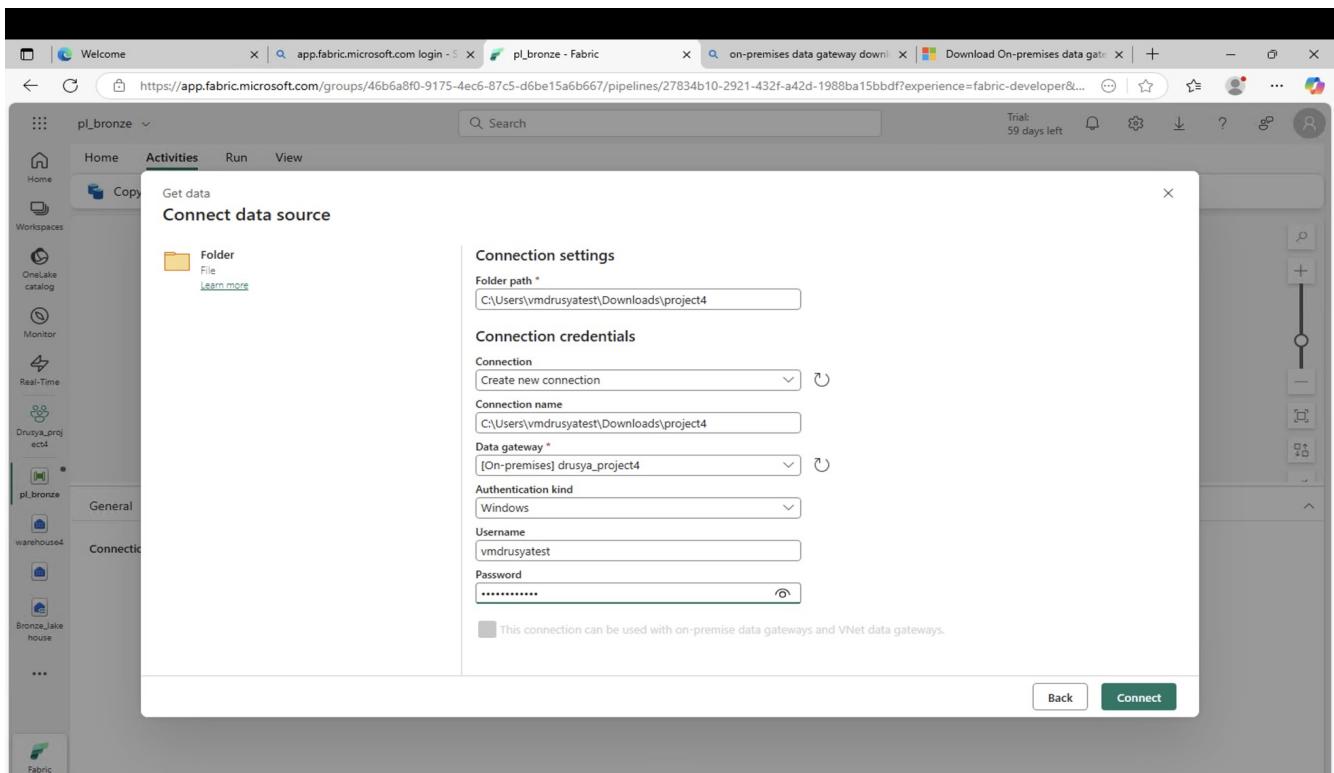
COPY THE FILES USING FOR EACH FROM ON-PREM TO LAKEHOUSE AS TABLES - BRONZE LAYER

For this we use a pipeline inside which we add a for each activity to pass all the files.



Now add a copy data activity inside the for each and add the source as on-prem data gateway and destination as lakehouse tables.

Created a new connection as data gateway in the source side.



Added the source details.

The screenshot shows the 'Copy data' activity configuration screen. The 'Source' tab is selected. It displays a 'ForEach' loop activity with a 'Get_data' child activity. The 'Source' tab includes fields for 'Connection' (set to 'C:\Users\vmdrusyatest\Downloads\project4'), 'File path type' (set to 'File path'), 'File path' ('Directory / @item()'), 'Recursively' (checked), 'File format' ('DelimitedText'), and 'Advanced' options like 'Start time (UTC)' and 'End time (UTC)'. The 'Destination' tab is visible above the 'Source' tab.

Added the destination details.

The screenshot shows the Azure Data Factory pipeline editor interface. A 'ForEach' activity is selected, which contains a single 'Copy data' activity. In the 'Destination' tab of the configuration pane, the connection is set to 'Bronze_Lakehouse' and the root folder is 'Tables'. The table is specified as '@item()' and the action is set to 'Overwrite'. The pipeline is currently in 'Design' mode, indicated by the green checkmark icon.

Run the pipeline and confirmed the data is loaded into lakehouse tables.

The screenshot shows the Azure Data Factory pipeline run history. The pipeline run ID is ff35c623-69e3-4173-b3d7-598bc4c02dff and the status is 'Succeeded'. The table lists six activities: 'Get_data' and five 'Copy data1' activities, all of which completed successfully. The 'Copy data1' activities were triggered by the 'Get_data' activity and copied data from it to the lakehouse table.

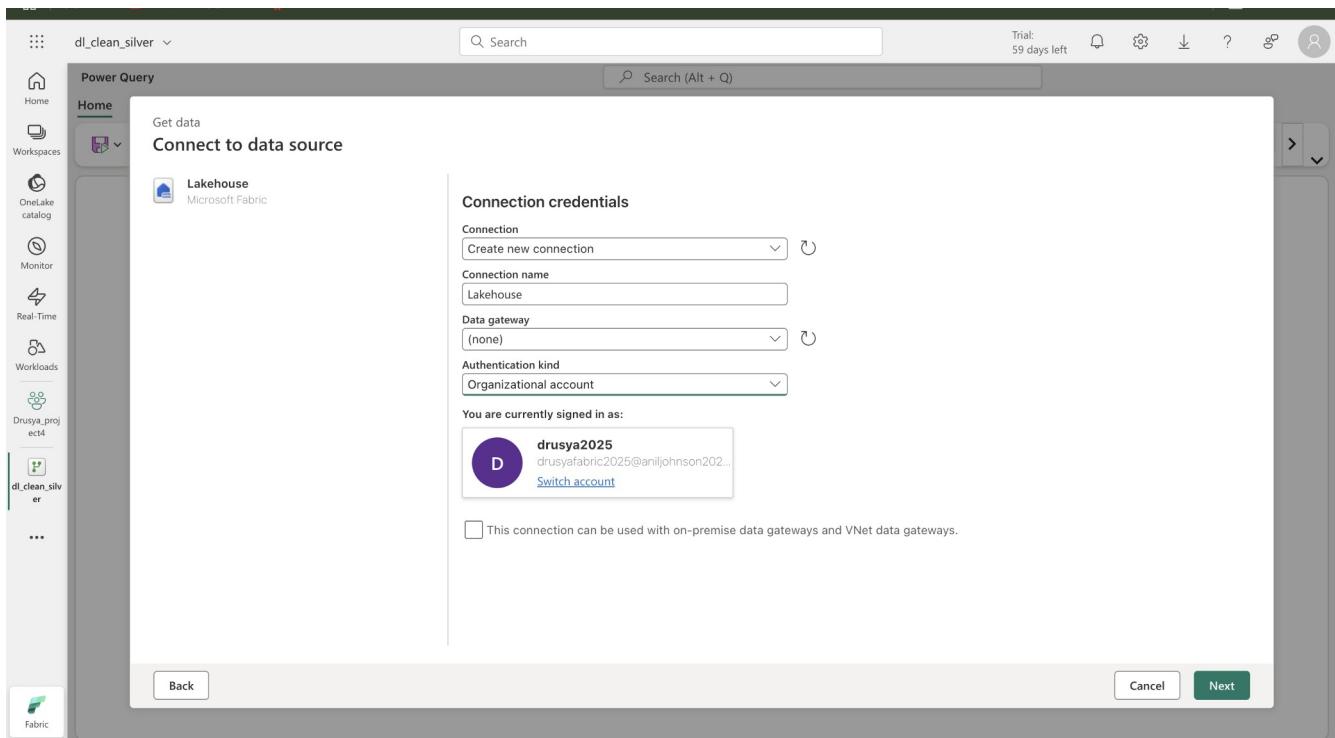
Activity name	Activity status	Run start	Duration	Input	Output
Get_data	Succeeded	5/6/2025, 5:07:50 PM	1m 10s		
Copy data1	Succeeded	5/6/2025, 5:07:52 PM	57s		
Copy data1	Succeeded	5/6/2025, 5:07:52 PM	52s		
Copy data1	Succeeded	5/6/2025, 5:07:52 PM	1m 3s		
Copy data1	Succeeded	5/6/2025, 5:07:52 PM	47s		
Copy data1	Succeeded	5/6/2025, 5:07:52 PM	1m 6s		

The screenshot shows the Microsoft Fabric Data Lakehouse interface. The left sidebar lists workspaces: Home, OneLake catalog, Monitor, Drusya_project4, Bronze_lakehouse, p_bronze, warehouse4, and another Bronze_lakehouse. The main area shows the 'Bronze_lakehouse' workspace. The top navigation bar includes 'Search', 'Trial: 59 days left', and various icons. Below the navigation is a toolbar with 'Get data', 'New semantic model', 'Open notebook', and 'Manage OneLake data access (preview)'. A message indicates a SQL analytics endpoint was created. The 'Explorer' panel on the left shows the 'Bronze_lakehouse' folder containing 'Tables' (accounts.csv, customers.csv, loan_payments.csv, loans.csv, transactions.csv) and 'Files'. The central area displays the 'customers.csv' table with 87 rows. The table has columns: ABC_customer_id, ABC_first_name, ABC_last_name, ABC_address, ABC_city, ABC_state, and ABC_zip. The data includes rows from 2 to 22, showing names like Jane Smith, Michael Johnson, etc., with addresses across Canada. The bottom status bar shows 'Succeeded (5 sec 89 ms)' and 'Columns 7 Rows 87'.

CLEANING THE DATA AND LOADING INTO WAREHOUSE – SILVER LAYER

Now we have our data in the lakehouse we have to clean the data before any transformation for that we are using the data flow activity.

Added the source as lakehouse.



Selected the tables I want to clean.

The screenshot shows the 'Power Query' interface in Microsoft Fabric, with 'dl_clean_silver' selected in the workspace sidebar. The main area is titled 'Get data' and 'Choose data'. A search bar at the top right shows 'transactions.csv'. The 'Display options' dropdown is open, showing several table names with checkboxes. Several checkboxes are checked for tables like 'accounts.csv', 'customers.csv', 'loan_payments.csv', 'loans.csv', and 'transactions.csv'. To the right, a preview pane displays the 'transactions.csv' data, which includes columns: transaction_id, account_id, transaction_date, transaction_amount, and transaction_type. The data shows various transactions from January 1st to January 22nd, with amounts ranging from 100.50 to 160.00 and types including Deposit and Withdrawal. At the bottom are 'Back', 'Cancel', and 'Create' buttons.

The transformations I used are data type changed.

This screenshot shows the Microsoft Power Query Editor interface. The left sidebar lists workspaces, and the main area shows three queries: 'accounts csv' (5 steps), 'customers csv' (4 steps), and 'loan_payments csv' (4 steps). The 'Transform' tab is selected. In the center, a table preview shows columns: account_id, customer_id, account_type, and balance. A formula bar at the top indicates a transformation step: `Table.TransformColumnTypes(#"Navigation 3", {"account_id", Int64.Type}, {"customer_id", Int64.Type}, {"balance", type number})`. The 'Applied steps' pane on the right shows a step named 'ABC 123 Changed c...'. The bottom status bar indicates 'Completed (0.62 s) Columns: 4 Rows: 99+'.

Replaced null with unknown values.

This screenshot shows the Microsoft Power Query Editor interface. The left sidebar lists workspaces, and the main area shows three queries: 'accounts csv' (5 steps), 'customers csv' (4 steps), and 'loan_payments csv' (4 steps). The 'Transform' tab is selected. A 'Replace values' dialog box is open over the data preview. It has two tabs: 'Basic' (selected) and 'Advanced'. The 'Value to find' field contains 'null' and the 'Replace with' field contains 'unknown'. The preview table below shows the same data as before, with the 'balance' column values. The 'Applied steps' pane on the right shows a step named 'ABC 123 Changed c...'. The bottom status bar indicates 'Completed (0.62 s) Columns: 4 Rows: 99+'.

Removed duplicates.

The screenshot shows the Power Query Editor interface with the 'Transform' tab selected. In the 'Queries [5]' pane, there are three queries: 'accounts csv', 'customers csv', and 'loan_payments csv'. The 'accounts csv' query has a context menu open over its table preview, with the 'Removed duplicates' option highlighted. The 'Applied steps' pane on the right shows the 'Removed duplicates' step applied to the 'accounts csv' source. The 'Data destination' section indicates 'No data destination'.

Add destination as warehouse.

The screenshot shows the Power Query Editor interface with the 'Transform' tab selected. In the 'Queries [5]' pane, there are three queries: 'accounts csv', 'customers csv', and 'loan_payments csv'. The 'accounts csv' query has a context menu open over its table preview, with the 'Warehouse' option selected under the 'New destination' dropdown. The 'Applied steps' pane on the right shows the 'Warehouse' step applied to the 'accounts csv' source. The 'Data destination' section indicates 'No data destination'.

dl_clean_silver

Power Query Draft saved

Home Data destination Connect to data destination

Warehouses [5] > Warehouse Microsoft Fabric

Connection credentials

Connection: Create new connection

Connection name: Warehouse

Data gateway: (none)

Authentication kind: Organizational account

You are currently signed in as: drusya2025 drusyafabric2025@aniljohnson202...

This connection can be used with on-premise data gateways and VNet data gateways.

Back Cancel Next

Columns: 4 Rows: 98 Add default destination...

This screenshot shows the 'Connect to data destination' dialog in Power Query. It's a step in a data flow named 'dl_clean_silver'. The user has selected 'Warehouse' from the Microsoft Fabric workspace. The 'Connection credentials' section is filled out with a new connection named 'Warehouse', no data gateway, and organizational authentication. The user is signed in as 'drusya2025'. A note at the bottom indicates the connection can be used with on-premise and VNet data gateways. Navigation buttons 'Back', 'Cancel', and 'Next' are visible at the bottom right, along with standard Power BI ribbon tabs.

dl_clean_silver

Power Query Draft saved

Home Data destination Choose destination target

Queries [5] > Warehouse

For performance reasons, only Warehouses in the current workspace are shown.

New table Existing table

Search

Display options

Warehouse [2]

StagingWarehouseForData... warehouse4

A new table will be created in warehouse4

Table name * accountscsv

Back Cancel Next

Columns: 4 Rows: 98 Add default destination...

This screenshot shows the 'Choose destination target' dialog in Power Query. The user has selected 'New table' and chosen the 'warehouse4' warehouse. They are creating a new table named 'accountscsv'. The dialog also notes that a new table will be created in the selected warehouse. Navigation buttons 'Back', 'Cancel', and 'Next' are visible at the bottom right, along with standard Power BI ribbon tabs.

The screenshot shows the 'Power Query' interface with a query named 'dl_clean_silver'. The 'Choose destination settings' dialog is open, displaying two main sections: 'Update method' and 'Schema options on publish'. In the 'Update method' section, there are four options: 'Existing data' (grid icon), 'New data' (grid icon with green dots), 'Append' (grid icon with a plus sign), and 'Replace' (grid icon with a circular arrow). In the 'Schema options on publish' section, there are three options: 'Existing schema' (grid icon), 'Dynamic schema' (grid icon with blue dots), and 'Fixed schema' (grid icon with blue squares). Below these sections is a 'Column mapping' table:

Source	Source type	Destination	Destination type
account_id	Whole number	account_id	Whole number
customer_id	Whole number	customer_id	Whole number
account_type	Text	account_type	Text
balance	Decimal number	balance	Decimal number

At the bottom right of the dialog are 'Cancel' and 'Save settings' buttons.

The same process repeated for all tables and confirmed the cleaned data is loaded in the warehouse as tables.

The screenshot shows the Azure Data Studio interface connected to a database named 'warehouse4'. The 'Explorer' pane on the left shows the database structure, including 'warehouses', 'schemas', and tables like 'dbo.accountscsv'. The 'accountscsv' table is currently selected. The 'Data preview - accountscsv' pane on the right displays the following data:

	account_id	customer_id	account_type	balance
1	1	45	Savings	1000.5
2	2	12	Checking	2500.75
3	3	78	Savings	1500
4	4	34	Checking	3000.25
5	5	56	Savings	500
6	6	23	Checking	1200.5
7	7	89	Savings	800.75
8	8	67	Checking	2200
9	9	14	Savings	900.25
10	10	92	Checking	1800.5
11	11	3	Savings	1100.75
12	12	81	Checking	2700
13	13	29	Savings	1300.25
14	14	64	Checking	3200.5
15	15	47	Savings	700.75
16	16	18	Checking	1400
17	17	99	Savings	600.25
18	18	5	Checking	1600.5
19	19	76	Savings	400.75
20	20	21	Checking	2000

At the bottom of the preview pane, it says 'Succeeded (1 sec 730 ms)' and 'Columns: 4 Rows: 98'.

warehouse4

Home Reporting Help

Get data New SQL query SQL templates Query activity Model layouts Download SQL database project Copilot Share

This warehouse has a default Power BI semantic model. To automatically add objects, go to warehouse settings. To manually add objects, use Manage default semantic model. Learn more

Explorer

+ Warehouses

warehouse4

- Schemas
 - dbo
 - Tables
 - accounts
 - customers
 - loan_pymt
 - loanscsv
 - transactior
 - Views
 - Functions
 - Stored Proc...
- INFORMATION_...
- queryinsights
- sys

customerscsv

Data preview - customerscsv

Showing 1000 rows Search

	customer_id	first_name	last_name	address	city	state
1	1	John	Doe	123 Elm St	Toronto	ON
2	2	Jane	Smith	456 Maple Ave	Ottawa	ON
3	3	Michael	Johnson	789 Oak Dr	Montreal	QC
4	4	Emily	Davis	101 Pine Rd	Calgary	AB
5	5	David	Wilson	202 Birch Blvd	Vancouver	BC
6	6	Emma	Clark	505 Cedar St	Halifax	NS
7	7	James	Martinez	606 Spruce Ln	Winnipeg	MB
8	8	Olivia	Garcia	707 Fir St	Edmonton	AB
9	9	William	Lopez	808 Redwood Dr	Victoria	BC
10	10	Ava	Anderson	909 Cypress Ave	Quebec City	QC
11	11	Alexander	Thomas	1010 Willow Rd	St. John's	NL
12	12	Isabella	Lee	1111 Poplar St	Fredericton	NB
13	13	Daniel	Harris	1212 Ash Blvd	Charlottetown	PE
14	14	Sophia	Young	1313 Beech Dr	Yellowknife	NT
15	15	Matthew	King	1414 Cedar Ln	Whitehorse	YT
16	16	Charlotte	Scott	1515 Elm St	Iqaluit	NU
17	17	Joseph	Green	1616 Maple Ave	Regina	SK
18	18	Amelia	Adams	1717 Oak Dr	Saskatoon	SK
19	19	Christopher	Baker	1818 Pine Rd	Thunder Bay	ON
20	20	Mia	Nelson	1919 Birch Blvd	London	ON

Columns: 6 Rows: 87

(Succeeded (1 sec 151 ms))

warehouse4

Home Reporting Help

Get data New SQL query SQL templates Query activity Model layouts Download SQL database project Copilot Share

This warehouse has a default Power BI semantic model. To automatically add objects, go to warehouse settings. To manually add objects, use Manage default semantic model. Learn more

Explorer

+ Warehouses

warehouse4

- Schemas
 - dbo
 - Tables
 - accounts
 - customers
 - loan_pymt
 - loanscsv
 - transactior
 - Views
 - Functions
 - Stored Proc...
- INFORMATION_...
- queryinsights
- sys

loan_paymentscsv

Data preview - loan_paymentscsv

Showing 1000 rows Search

	payment_id	loan_id	payment_date	payment_amount
1	1	45	2024-01-01	100
2	2	23	2024-01-02	150
3	3	67	2024-01-03	200
4	4	89	2024-01-04	250
5	5	12	2024-01-05	300
6	6	34	2024-01-06	350
7	7	56	2024-01-07	400
8	8	78	2024-01-08	450
9	9	90	2024-01-09	500
10	10	11	2024-01-10	550
11	11	22	2024-01-11	600
12	12	33	2024-01-12	650
13	13	44	2024-01-13	700
14	14	55	2024-01-14	750
15	15	66	2024-01-15	800
16	16	77	2024-01-16	850
17	17	88	2024-01-17	900
18	18	99	2024-01-18	950
19	19	10	2024-01-19	1000
20	20	21	2024-01-20	1050

Columns: 4 Rows: 100

(Succeeded (0 sec 715 ms))



Screenshot of the Snowflake Data Preview interface for the 'loanscsv' table in the 'warehouse4' warehouse.

The table has 10 columns:

	loan_id	customer_id	loan_amount	interest_rate	loan_term
1	1	45	10000.5	5.5	36
2	2	12	20000.75	4.5	48
3	3	78	15000	6	60
4	4	34	30000.25	3.5	24
5	5	56	25000	5	36
6	6	23	17500.5	4	48
7	7	89	22500.75	6.5	60
8	8	67	27500	3	24
9	9	14	32500.25	5.5	36
10	10	92	37500.5	4.5	48
11	11	3	10000.75	6	60
12	12	81	20000	3.5	24
13	13	29	15000.25	5	36
14	14	64	30000.5	4	48
15	15	47	25000.75	6.5	60
16	16	18	17500	3	24
17	17	99	22500.25	5.5	36
18	18	5	27500.5	4.5	48
19	19	76	32500.75	6	60
20	20	21	37500	3.5	24

The preview succeeded in 734 ms with 100 rows.

Screenshot of the Snowflake Data Preview interface for the 'transactionscsv' table in the 'warehouse4' warehouse.

The table has 5 columns:

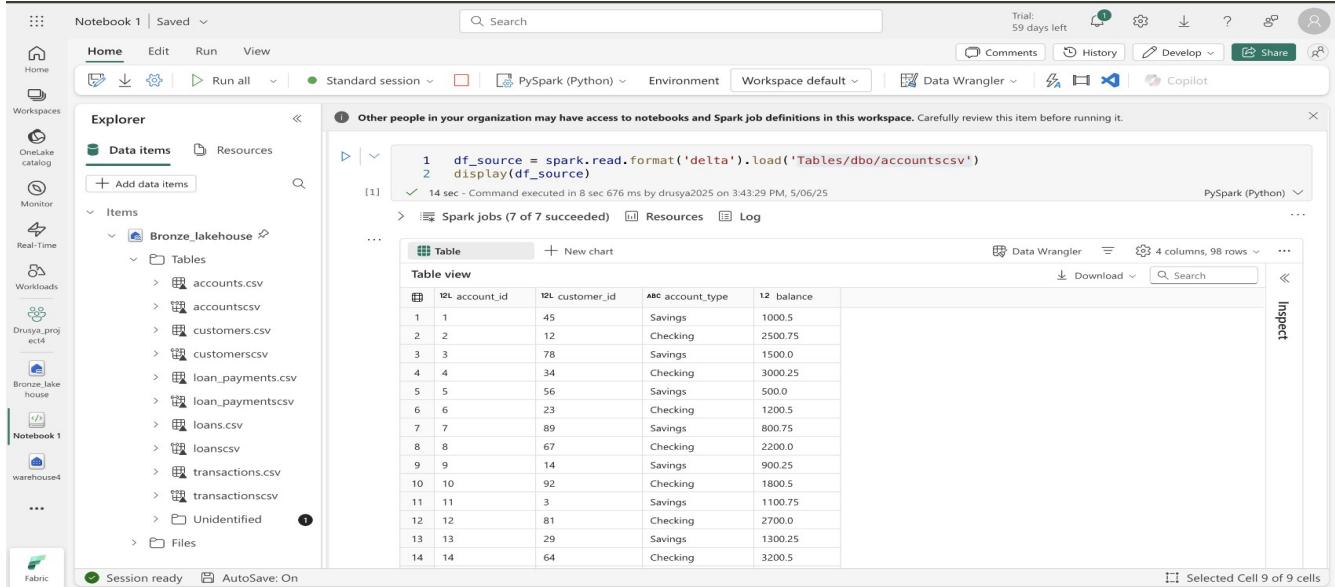
	transaction_id	account_id	transaction_date	transaction_amount	transaction_type
1	1	45	2024-01-01	100.5	Deposit
2	2	12	2024-01-02	200.75	Withdrawal
3	3	78	2024-01-03	150	Deposit
4	4	34	2024-01-04	300.25	Withdrawal
5	5	56	2024-01-05	250	Deposit
6	6	23	2024-01-06	175	Withdrawal
7	7	89	2024-01-07	225.5	Deposit
8	8	67	2024-01-08	275.75	Withdrawal
9	9	14	2024-01-09	325	Deposit
10	10	92	2024-01-10	375.25	Withdrawal
11	11	3	2024-01-11	100.5	Deposit
12	12	81	2024-01-12	200.75	Withdrawal
13	13	29	2024-01-13	150	Deposit
14	14	64	2024-01-14	300.25	Withdrawal
15	15	47	2024-01-15	250	Deposit
16	16	18	2024-01-16	175	Withdrawal
17	17	99	2024-01-17	225.5	Deposit
18	18	5	2024-01-18	275.75	Withdrawal
19	19	76	2024-01-19	325	Deposit
20	20	21	2024-01-20	375.25	Withdrawal

The preview succeeded in 863 ms with 100 rows.

PERFORMING SCD TYPE 1 TRANSFORMATIONS ON THE CLEANED DATA – GOLD LAYER

To perform the SCD Type 1 transformations on the cleaned data we use fabric notebooks here. I used the standard session cluster for running the notebooks and created a short cut to access the data from warehouse.

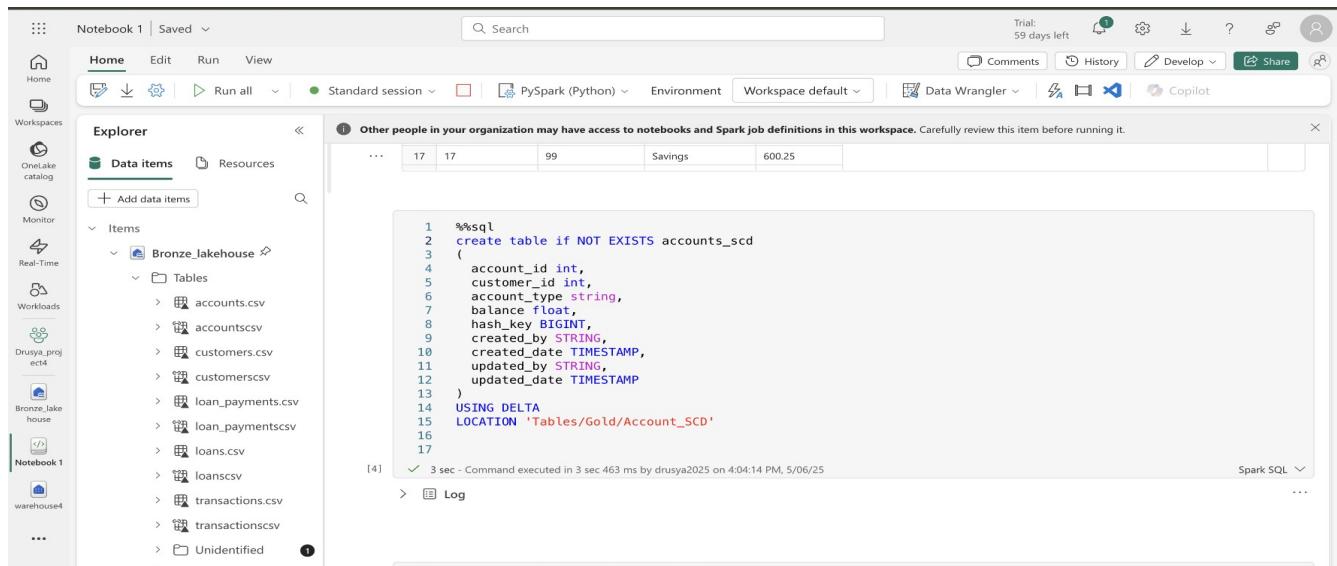
First I read my data from the warehouse table.



```
df_source = spark.read.format('delta').load('Tables/dbo/accounts.csv')
display(df_source)
```

account_id	customer_id	account_type	balance
1	45	Savings	1000.5
2	12	Checking	2500.75
3	78	Savings	1500.0
4	34	Checking	3000.25
5	56	Savings	500.0
6	23	Checking	1200.5
7	89	Savings	800.75
8	67	Checking	2200.0
9	14	Savings	900.25
10	92	Checking	1800.5
11	3	Savings	1100.75
12	81	Checking	2700.0
13	29	Savings	1300.25
14	64	Checking	3200.5

Next I created the delta target table for the file.



```
%sql
create table if NOT EXISTS accounts_scd
(
    account_id int,
    customer_id int,
    account_type string,
    balance float,
    hash_key BIGINT,
    created_by STRING,
    created_date TIMESTAMP,
    updated_by STRING,
    updated_date TIMESTAMP
)
USING DELTA
LOCATION 'Tables/Gold/Account_SCD'
```

Then I assigned my target file path and source file path, after that I added a new column hashkey to the source file which combines all the column data using the concat and crc32 command and converted it into a data frame.

The screenshot shows a Databricks notebook interface. The sidebar on the left lists workspaces: OneLake catalog, Monitor, Real-Time, Workloads, Drusya_project_ec4, Bronze_Lake_house, Notebook 1, warehouse4, and Fabric. The main area has tabs for Home, Edit, Run, View, Standard session (selected), PySpark (Python), Environment, Workspace default, Data Wrangler, Develop, Share, and Copilot. A search bar at the top right includes a 'Comments' button. The notebook title is 'Notebook 1 | Saved'. The code cell [5] contains:

```

1 target_path = 'Tables/Gold/Account_SCD'
2 source_path = 'Tables/dbo/accountscsv'

```

Cell [5] output: <1 sec - Command executed in 352 ms by drusya2025 on 4:04:27 PM, 5/06/25

The code cell [6] contains:

```

1 from pyspark.sql.functions import *
2 df_source1 = df_source.withColumn("hashkey", crc32(concat(*df_source.columns)))
3 display(df_source1)

```

Cell [6] output: 1 sec - Command executed in 1 sec 697 ms by drusya2025 on 4:10:17 PM, 5/06/25

The resulting DataFrame (Table view) is displayed below the code cell [6]. It has columns: account_id, customer_id, account_type, balance, hashkey. The data is:

	account_id	customer_id	account_type	balance	hashkey
1	1	45	Savings	1000.5	2454403084
2	2	12	Checking	2500.75	796571617
3	3	78	Savings	1500.0	199554578
4	4	34	Checking	3000.25	761699333
5	5	56	Savings	500.0	3886787448
6	6	23	Checking	1200.5	589336936
7	7	89	Savings	800.75	74260363

Now I converted my target table into a data frame.

The screenshot shows a Databricks notebook interface. The sidebar on the left lists workspaces: OneLake catalog, Monitor, Real-Time, Workloads, Drusya_project_ec4, Bronze_Lake_house, Notebook 1, warehouse4, and Fabric. The main area has tabs for Home, Edit, Run, View, Standard session (selected), PySpark (Python), Environment, Workspace default, Data Wrangler, Develop, Share, and Copilot. A search bar at the top right includes a 'Comments' button. The notebook title is 'Notebook 1 | Saved'. The code cell [7] contains:

```

1 from delta.tables import DeltaTable
2 dbtable1=DeltaTable.forName(spark,target_path)
3 dbtable1.toDF().show()

```

Cell [7] output: 3 sec - Command executed in 3 sec 494 ms by drusya2025 on 4:12:47 PM, 5/06/25

The resulting DataFrame (Table view) is displayed below the code cell [7]. It has columns: account_id, customer_id, account_type, balance, hash_key, created_by, created_date, updated_by, updated_date. The data is:

account_id	customer_id	account_type	balance	hash_key	created_by	created_date	updated_by	updated_date
14	14	Checking	5200.5	3349b4 / 104				
15	15	Savings	700.75	1744154229				
16	16	Checking	1400.0	427159154				
17	17	Savings	600.25	1642826618				

The code cell [8] contains:

```

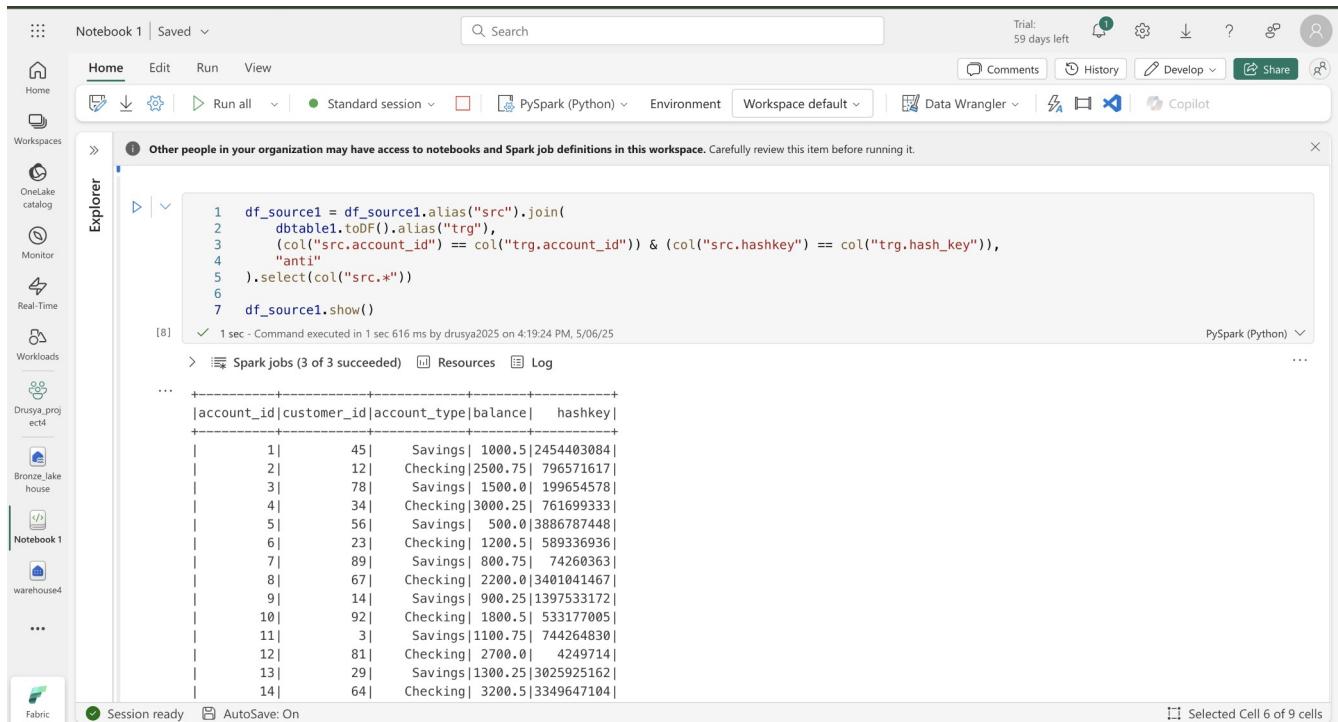
1 df_source1 = df_source1.alias("src").join(
2     dbtable1.toDF().alias("trg"),
3     (col("src.account_id") == col("trg.account_id")) & (col("src.hashkey") == col("trg.hash_key")),
4     "anti"
5 ).select(col("src.*"))
6
7 df_source1.show()

```

The resulting DataFrame (Table view) is displayed below the code cell [8]. It has columns: account_id, customer_id, account_type, balance, hashkey. The data is:

account_id	customer_id	account_type	balance	hashkey
14	14	Checking	5200.5	3349b4 / 104
15	15	Savings	700.75	1744154229
16	16	Checking	1400.0	427159154
17	17	Savings	600.25	1642826618

With the help of anti join command I joined my source and target data frame. The purpose of this command is to check the id and hashkey columns so that we can determine the data is an updated one or new.



The screenshot shows a Databricks notebook interface with the following details:

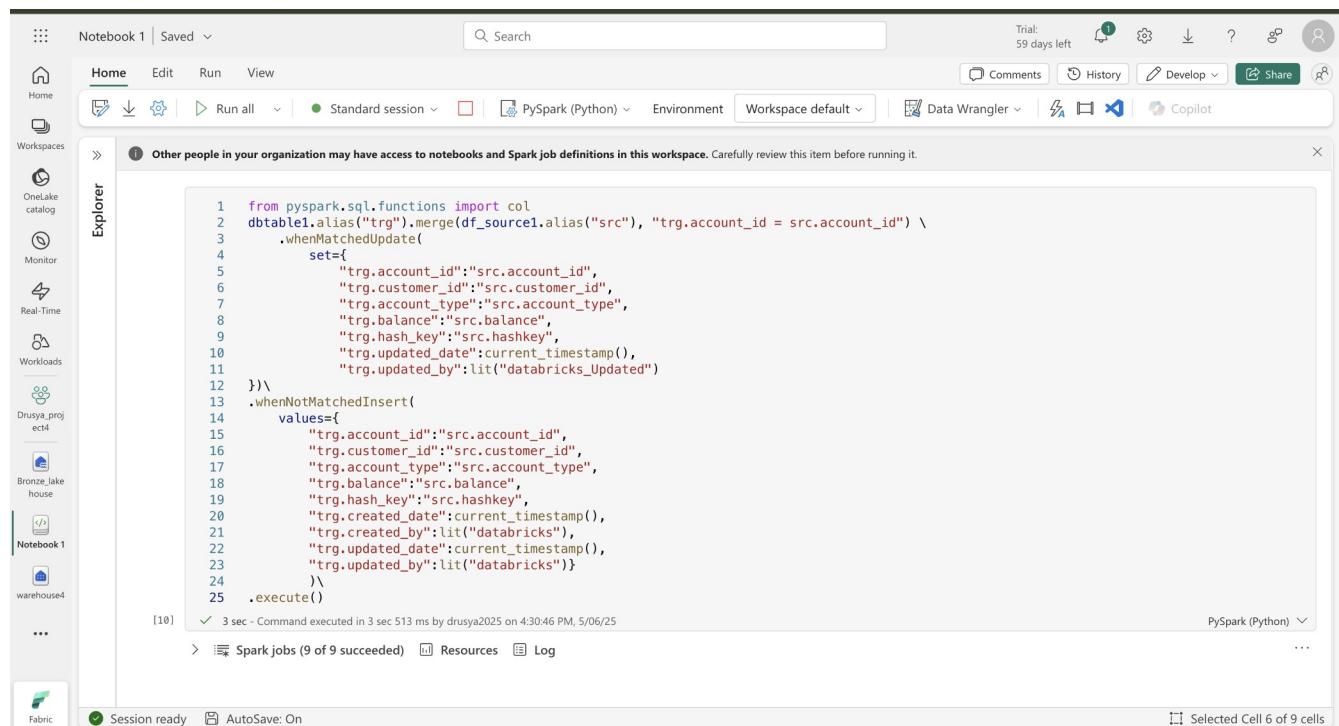
- Header:** Notebook 1 | Saved | Trial: 59 days left
- Toolbar:** Home, Edit, Run, View, Standard session, PySpark (Python), Environment, Workspace default, Data Wrangler, Develop, Share, Copilot.
- Left Sidebar (Workspaces):** OneLake catalog, Monitor, Real-Time, Workloads, Drusya_project4, Bronze_Lake_house, Notebook 1, warehouse4, Fabric.
- Central Area:**
 - Cell 8:** Shows the execution of an anti-join command and its output. The command is:

```
1 df_source1 = df_source1.alias("src").join(
2   dbtable1.toDF().alias("trg"),
3   (col("src.account_id") == col("trg.account_id")) & (col("src.hashkey") == col("trg.hash_key")),
4   "anti"
5 ).select(col("src.*"))
6
7 df_source1.show()
```

 - Output:** A table showing account data. The first few rows are:

account_id	customer_id	account_type	balance	hashkey
1	45	Savings	1000.5	2454403084
2	12	Checking	2500.75	796571617
3	78	Savings	1500.0	199654578
4	34	Checking	3000.25	761699333
5	56	Savings	500.0	3886787448
6	23	Checking	1200.5	589336936
7	89	Savings	800.75	74260363
8	67	Checking	2200.0	3401041467
9	14	Savings	900.25	1397533172
10	92	Checking	1800.5	533177005
11	3	Savings	1100.75	744264830
12	81	Checking	2700.0	4249714
13	29	Savings	1300.25	3025925162
14	64	Checking	3200.5	3349647104

Now with the help of merge command we are updating records based on update or new records into the table.



The screenshot shows a Databricks notebook interface with the following details:

- Header:** Notebook 1 | Saved | Trial: 59 days left
- Toolbar:** Home, Edit, Run, View, Standard session, PySpark (Python), Environment, Workspace default, Data Wrangler, Develop, Share, Copilot.
- Left Sidebar (Workspaces):** OneLake catalog, Monitor, Real-Time, Workloads, Drusya_project4, Bronze_Lake_house, Notebook 1, warehouse4, Fabric.
- Central Area:**
 - Cell 10:** Shows the execution of a merge command. The command is:

```
1 from pyspark.sql.functions import col
2 dbtable1.alias("trg").merge(df_source1.alias("src"), "trg.account_id = src.account_id") \
3   .whenMatchedUpdate(
4     set={
5       "trg.account_id": "src.account_id",
6       "trg.customer_id": "src.customer_id",
7       "trg.account_type": "src.account_type",
8       "trg.balance": "src.balance",
9       "trg.hash_key": "src.hashkey",
10      "trg.updated_date": current_timestamp(),
11      "trg.updated_by": lit("databricks_Updated")
12    })
13   .whenNotMatchedInsert(
14     values={
15       "trg.account_id": "src.account_id",
16       "trg.customer_id": "src.customer_id",
17       "trg.account_type": "src.account_type",
18       "trg.balance": "src.balance",
19       "trg.hash_key": "src.hashkey",
20       "trg.created_date": current_timestamp(),
21       "trg.created_by": lit("databricks"),
22       "trg.updated_date": current_timestamp(),
23       "trg.updated_by": lit("databricks")}
24     )
25   .execute()
```

 - Output:** A message indicating 9 successful spark jobs.

Finally I displayed the output table. Used the same logic on the rest of the files.

Accounts table:

The screenshot shows a Databricks notebook interface. The sidebar on the left lists workspaces: OneLake catalog, Monitor, Real-Time, Workloads, Drusya_proj ect4, Bronze_lake house, Notebook 1, warehouse4, and Fabric. The main area shows a code cell with the command `display(spark.read.format("delta").load(target_path))`. Below the code cell is a table titled "Table view" with 9 columns and 98 rows. The columns are: account_id, customer_id, account_type, balance, hash_key, created_by, created_date, updated_by, and updated_date. The data includes various account types like Savings and Checking, with balances ranging from 500.0 to 2500.75. The table is sorted by account_id. A tooltip on the right says "Selected Cell 6 of 9 cells".

Customers table:

The screenshot shows a Databricks notebook interface. The sidebar on the left lists workspaces: OneLake catalog, Monitor, Real-Time, Workloads, Drusya_proj ect4, Bronze_lake house, Notebook 1, warehouse4, and Fabric. The main area shows a code cell with the command `display(spark.read.format("delta").load(target_path2))`. Below the code cell is a table titled "Table view" with 11 columns and 87 rows. The columns are: customer_id, first_name, last_name, address, city, state, hash_key, created_by, created_date, updated_by, and updated_date. The data includes names like John Doe, Jane Smith, Michael Johnson, etc., from various locations across Canada. The table is sorted by customer_id. A tooltip on the right says "Selected Cell 37 of 38 cells".

Loan payments table:

The screenshot shows a Databricks notebook interface. The sidebar on the left lists workspaces: OneLake catalog, Drusya_project4, Bronze_Lake_house, Notebook1, warehouse4, and Fabric. The main area shows a code cell with the command `display(spark.read.format("delta").load(target_path3))`. Below the code cell is a table titled "Table view" with 9 columns and 100 rows. The columns are: payment_id, loan_id, payment_date, payment_amount, hash_key, created_by, created_date, updated_by, and updated_date. The data in the table represents loan payments.

	payment_id	loan_id	payment_date	payment_amount	hash_key	created_by	created_date	updated_by	updated_date
1	1	45	2024-01-01	100.0	4189237233	databricks	2025-05-06 21:3...	databricks	2025-05-06 21:30...
2	2	23	2024-01-02	150.0	1457204963	databricks	2025-05-06 21:3...	databricks	2025-05-06 21:30...
3	3	67	2024-01-03	200.0	2777852429	databricks	2025-05-06 21:3...	databricks	2025-05-06 21:30...
4	4	89	2024-01-04	250.0	264335070	databricks	2025-05-06 21:3...	databricks	2025-05-06 21:30...
5	5	12	2024-01-05	300.0	438074744	databricks	2025-05-06 21:3...	databricks	2025-05-06 21:30...
6	6	34	2024-01-06	350.0	3330136997	databricks	2025-05-06 21:3...	databricks	2025-05-06 21:30...
7	7	56	2024-01-07	400.0	1192820904	databricks	2025-05-06 21:3...	databricks	2025-05-06 21:30...
8	8	78	2024-01-08	450.0	2203091019	databricks	2025-05-06 21:3...	databricks	2025-05-06 21:30...
9	9	90	2024-01-09	500.0	2681690964	databricks	2025-05-06 21:3...	databricks	2025-05-06 21:30...
10	10	11	2024-01-10	550.0	3325951132	databricks	2025-05-06 21:3...	databricks	2025-05-06 21:30...
11	11	22	2024-01-11	600.0	3894410316	databricks	2025-05-06 21:3...	databricks	2025-05-06 21:30...
12	12	33	2024-01-12	650.0	2585655648	databricks	2025-05-06 21:3...	databricks	2025-05-06 21:30...
13	13	44	2024-01-13	700.0	1774633368	databricks	2025-05-06 21:3...	databricks	2025-05-06 21:30...
14	14	55	2024-01-14	750.0	2348634532	databricks	2025-05-06 21:3...	databricks	2025-05-06 21:30...
15	15	66	2024-01-15	800.0	1613741173	databricks	2025-05-06 21:3...	databricks	2025-05-06 21:30...
...	2024-01-16	850.0	3040864209	databricks	2025-05-06 21:3...	databricks	2025-05-06 21:30...

Loans table:

The screenshot shows a Databricks notebook interface. The sidebar on the left lists workspaces: OneLake catalog, Drusya_project4, Bronze_Lake_house, Notebook1, warehouse4, and Fabric. The main area shows a code cell with the command `display(spark.read.format("delta").load(target_path4))`. Below the code cell is a table titled "Table view" with 10 columns and 100 rows. The columns are: loan_id, customer_id, loan_amount, interest_rate, loan_term, hash_key, created_by, created_date, updated_by, and updated_date. The data in the table represents loans.

	loan_id	customer_id	loan_amount	interest_rate	loan_term	hash_key	created_by	created_date	updated_by	updated_date
1	1	45	10000.5	5.5	36	2502041691	databricks	2025-05-06 21:33...	databricks	2025-05-06 21:33...
2	2	12	20000.75	4.5	48	3355992476	databricks	2025-05-06 21:3...	databricks	2025-05-06 21:33...
3	3	78	15000.0	6.0	60	4060872917	databricks	2025-05-06 21:3...	databricks	2025-05-06 21:33...
4	4	34	30000.25	3.5	24	2847880128	databricks	2025-05-06 21:3...	databricks	2025-05-06 21:33...
5	5	56	25000.0	5.0	36	1867834730	databricks	2025-05-06 21:3...	databricks	2025-05-06 21:33...
6	6	23	17500.5	4.0	48	172242294	databricks	2025-05-06 21:3...	databricks	2025-05-06 21:33...
7	7	89	22500.75	6.5	60	1348082437	databricks	2025-05-06 21:3...	databricks	2025-05-06 21:33...
8	8	67	27500.0	3.0	24	4106075248	databricks	2025-05-06 21:3...	databricks	2025-05-06 21:33...
9	9	14	32500.25	5.5	36	3981072152	databricks	2025-05-06 21:3...	databricks	2025-05-06 21:33...
10	10	92	37500.5	4.5	48	888718168	databricks	2025-05-06 21:3...	databricks	2025-05-06 21:33...
11	11	3	10000.75	6.0	60	2179146733	databricks	2025-05-06 21:3...	databricks	2025-05-06 21:33...
12	12	81	20000.0	3.5	24	1426967511	databricks	2025-05-06 21:3...	databricks	2025-05-06 21:33...
13	13	29	15000.25	5.0	36	1193680727	databricks	2025-05-06 21:3...	databricks	2025-05-06 21:33...
14	14	64	30000.5	4.0	48	2167198012	databricks	2025-05-06 21:3...	databricks	2025-05-06 21:33...
15	15	47	25000.75	6.5	60	3344653795	databricks	2025-05-06 21:3...	databricks	2025-05-06 21:33...
...	2024-01-20	20000.0	20	1323054209	databricks	2025-05-06 21:3...	databricks	2025-05-06 21:33...

Transactions table:

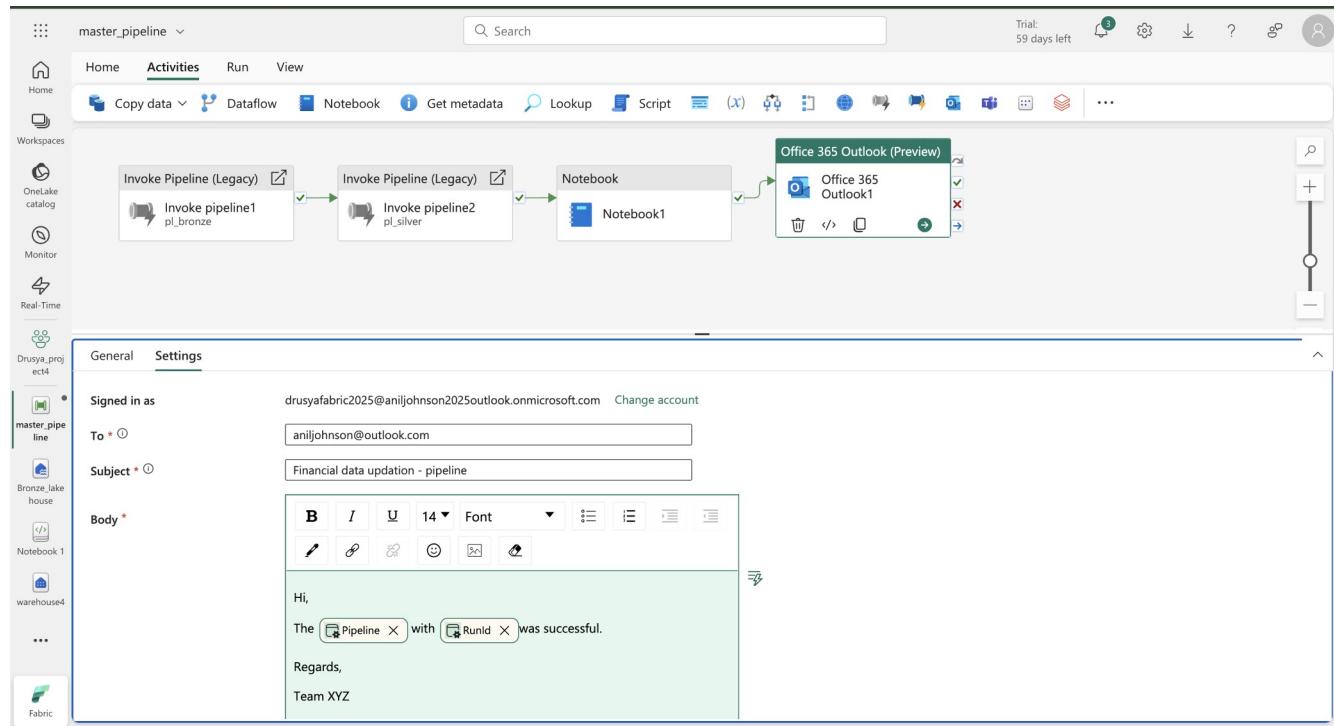
The screenshot shows a Databricks Notebook interface. The notebook title is "Notebook 1 | Saved". The code cell contains a single line of PySpark code: `display(spark.read.format("delta").load(target_path5))`. The output of the cell shows a table with 10 columns and 100 rows. The columns are labeled: transaction_id, account_id, transaction_date, transaction_amount, transaction_type, hash_key, created_by, created_date, updated_by. The table data represents various transactions, mostly deposits and withdrawals, across different dates and account IDs.

CREATING MASTER PIPELINE AND SCHEDULING THE PIPELINE WITH EMAIL NOTIFICATION TASK

Created a master pipeline and connected all the activities together.

The screenshot shows the Databricks Pipeline Designer interface. The pipeline name is "master_pipeline". The pipeline flowchart consists of four main components connected sequentially: "Invoke Pipeline (Legacy)" (Bronze), "Invoke Pipeline (Legacy)" (Silver), "Notebook" (Gold), and "Office 365 Outlook (Preview)". The "Notebook" component is highlighted with a green border. Below the flowchart, the "General" tab of the pipeline settings is visible, showing the pipeline name "Office 365 Outlook1" and the "Activated" activity state.

Created an email activity so that we will get an email notification upon successful run of the pipeline. Since the email id I have is not supporting in Fabric it will show an error if we use an enterprise or company email it will work due to limitations I am only showing the demo of the email activity process.



Now I triggered my pipeline to run at specific interval of time.

