



TRANSACTIONS AND LOAN DATA FOR A CUSTOMER

Bootcamp Project - 2



APRIL 24, 2025
BY HARJINDER SINGH

Table of Contents

Introduction.....	2
Service Principal Setup	3
Databricks Scope Creation.....	12
MyUtilityFunctions Notebook	14
Silver Layer Notebook	20
Gold Delta Table Creation Notebook.....	32
Gold Layer Notebook	38
Databricks Workflow Automation	50
Output Review	55
Power BI Report	63

Introduction

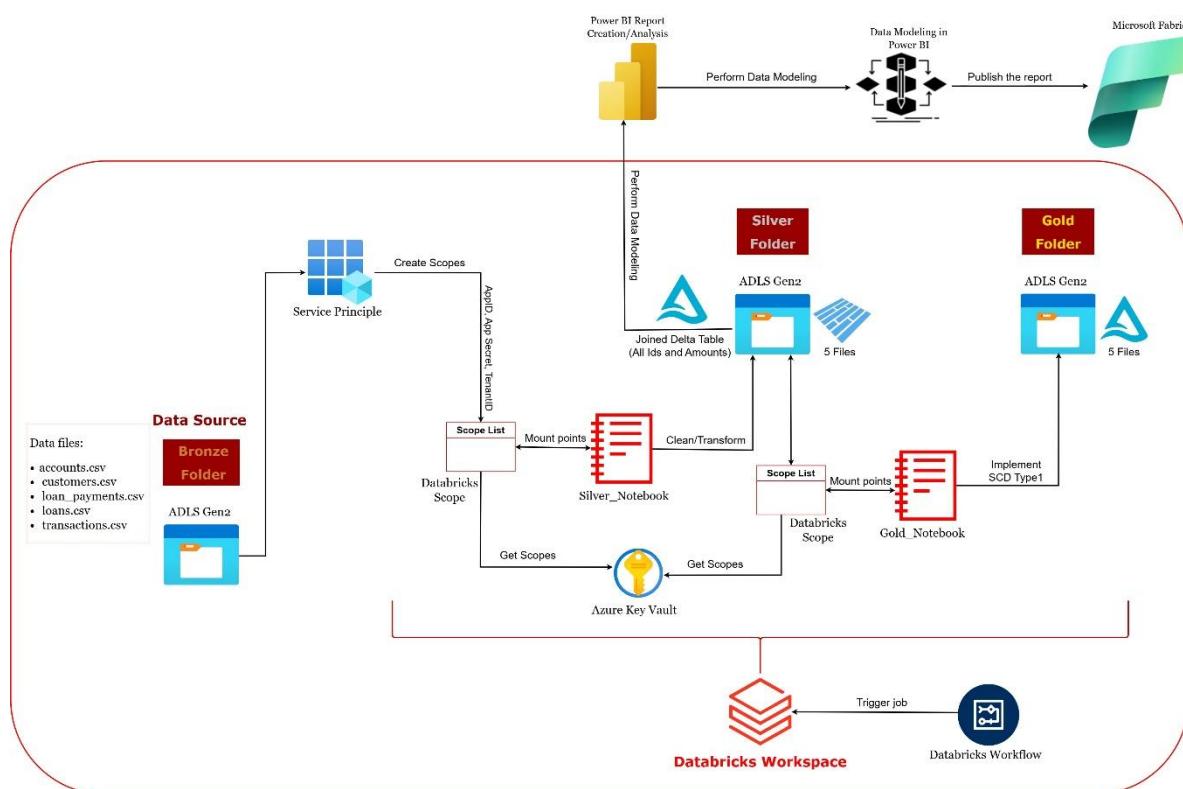
This project outlines the end-to-end implementation of a modern data pipeline using **Azure Data Lake Storage Gen2**, **Databricks**, and **Power BI**. The goal is to extract raw banking-related data files from a source location, clean and transform the data using **PySpark in Databricks**, apply **Slowly Changing Dimensions (SCD) Type 1** for managing dimensional data, and finally visualize insights using Power BI.

The process follows a structured multi-layered architecture:

- **Bronze Layer** for raw data ingestion,
- **Silver Layer** for cleaned and transformed data, and
- **Gold Layer** for final curated data in **Delta format**.

Dynamic parameters and **Azure Key Vault** integration ensure secure, scalable, and configurable pipelines. The deliverables include detailed documentation, code snippets, and an architecture diagram to illustrate the workflow clearly.

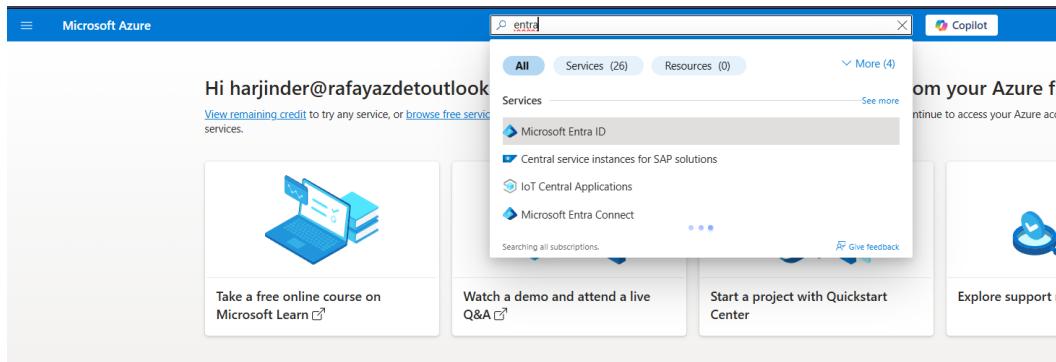
Architecture Followed:



Service Principal Setup

➤ Steps to connect ADLS gen2 storage with databricks using Service Principal:

- Open the Azure Account -> Search for the **Microsoft Entra ID** in the search bar.



- Select the **Microsoft Entra ID** option -> select the **App Registrations** Option under the **Manage** option -> click on **+ New registration** to register an app.

A screenshot of the 'Default Directory | App registrations' page in the Azure portal. The left sidebar shows navigation options like Overview, Preview features, and various management sections. The 'App registrations' section is highlighted with a yellow box. The main content area has a header with tabs for 'Endpoints', 'Troubleshoot', 'Refresh', 'Download', 'Preview features', and 'Got feedback?'. A prominent red box highlights the '+ New registration' button. Below it, a message states: 'Starting June 30th, 2020 we will no longer add any new features to Azure Active Directory Authentication Library (ADAL) and Azure Active Directory Graph. We will continue to provide feature updates. Applications will need to be upgraded to Microsoft Authentication Library (MSAL) and Microsoft Graph.' Below this, there are tabs for 'All applications', 'Owned applications' (which is selected), and 'Deleted applications'. A search bar says 'Start typing a display name or application (client) ID to filter these r...'. A blue button labeled '+ Add filters' is next to it. A message at the bottom right says 'There are no applications here.' and a blue button says 'Register an application'.

- Next, enter the **name** for new app registration and select any option for Supported account types options -> click on **Register** button.

* Name
The user-facing display name for this application (this can be changed later).
 ✓

Supported account types
Who can use this application or access this API?
 Accounts in this organizational directory only (Default Directory only - Single tenant)
 Accounts in any organizational directory (Any Microsoft Entra ID tenant - Multitenant)
 Accounts in any organizational directory (Any Microsoft Entra ID tenant - Multitenant) and personal Microsoft accounts (e.g. Skype, Xbox)
 Personal Microsoft accounts only
[Help me choose...](#)

Redirect URI (optional)
We'll return the authentication response to this URI after successfully authenticating the user. Providing this now is optional and it can be changed later, but a value is required for most authentication scenarios.

[By proceeding, you agree to the Microsoft Platform Policies](#) ↗

Register

- Now the registration is done, we can see the Essentials related to the registration.

So, **copy the Application ID & Directory ID and save it for later use.**

Next, click on **Add a certificate or secret** option of Client credentials in registered app essentials.

Home > Default Directory | App registrations >
spn-databricks ↗ ...

Search 🔍 Delete ✖ Endpoints 🌐 Preview features ➡

Overview Quickstart Integration assistant Diagnose and solve problems Manage Support + Troubleshooting

Essentials

Display name	: spn-databricks	Client credentials	Add a certificate or secret
Application (client) ID	: 041612e6-9f6b-409e-a89d-0a39c8cb7503	Redirect URLs	: Add a Redirect URI
Object ID	: c4f2695d-ad72-4301-89a6-5e70a603750e	Application ID URI	: Add an Application ID URI
Directory (tenant) ID	: 33d03180-c464-491c-83c0-b95da1d65d49	Managed application in I...	: spn-databricks
Supported account types : My organization only			

ℹ Welcome to the new and improved App registrations. Looking to learn how it's changed from App registrations (Legacy)? [Learn more](#)

ℹ Starting June 30th, 2020 we will no longer add any new features to Azure Active Directory Authentication Library (ADAL) and Azure Active Directory Graph. We will continue to provide technical updates but we will no longer provide feature updates. Applications will need to be upgraded to Microsoft Authentication Library (MSAL) and Microsoft Graph. [Learn more](#)

[Get Started](#) Documentation

- Next, click on **+ New client secret** option under the client secrets tab.

Credentials enable confidential applications to identify themselves to the authentication service when receiving tokens at a web addressable location (using an HTTPS scheme). For a higher level of assurance, we recommend using a certificate (instead of a client secret) as a credential.

Certificates & secrets

- Token configuration
- API permissions
- Expose an API
- App roles
- Owners
- Roles and administrators

Client secrets (0)

A secret string that the application uses to prove its identity when requesting a token. Also can be referred to as application password.

New client secret

Description	Expires	Value	Secret ID
No client secrets have been created for this application.			

- Now, enter the **description** and **expires** option for client secret. (In my case, I opted for **Custom**).

Add a client secret

Description	databricksconn
Expires	Custom
Start	Recommended: 180 days (6 months) 90 days (3 months) 365 days (12 months) 545 days (18 months) 730 days (24 months)
End	Custom

Add **Cancel**

- And, select the **Start** and **End** Date for the secret -> click on **Add** button.
As I opted for custom option for Expires field so I am giving the randomly about 1 week.
Usually, in the industry, they select for **90 days and 180 days** as Expires values.

Start	04/25/2025
End	05/23/2025

- The created secret is now visible, so **copy the secret value** from it and **store the value for the later use**. (As the value won't be available for longer time.)

The screenshot shows the Azure portal's 'Certificates & secrets' blade for the app registration 'spn-databricks'. The 'Client secrets' tab is selected, showing one entry:

Description	Expires	Value	Secret ID
databricksconn	4/5/2025	[REDACTED]	e106c9d6-d4e8-4a85-851a-3bdca6c43...

- Next, Move to **ALDS Gen2 storage** account -> go to **Access Control (IAM)** -> click on **+ Add** -> select **Add role assignment**.

The screenshot shows the Azure portal's 'Access Control (IAM)' blade for the storage account 'hsinghadls'. The 'Add role assignment' button is highlighted.

we need to give the Storage blob data contributor access to the Service principle so that we can perform the action using service principle to the ADLS gen2 storage account.

- So, search for **Storage Blob Data Contributor** access -> click on next.

Add role assignment ...

Role Members Conditions Review + assign

A role definition is a collection of permissions. You can use the built-in roles or you can create your own custom roles. [Learn more](#)

Job function roles Privileged administrator roles

Grant access to Azure resources based on job function, such as the ability to create virtual machines.

Name ↑↓	Description ↑↓	Type ↑↓	Category ↑↓	Details
Defender CSPM Storage Data Scanner	Grants access to read blobs and files. This role is used by the data scanner of Defender CSPM.	BuiltInRole	None	View
Defender for Storage Data Scanner	Grants access to read blobs and update index tags. This role is used by the data scanner of Defender for Storage.	BuiltInRole	None	View
Storage Blob Data Contributor	Allows for read, write and delete access to Azure Storage blob containers and data	BuiltInRole	Storage	View
Storage Blob Data Owner	Allows for full access to Azure Storage blob containers and data, including assigning POSIX access control.	BuiltInRole	Storage	View
Storage Blob Data Reader	Allows for read access to Azure Storage blob containers and data	BuiltInRole	Storage	View
Storage Blob Delegator	Allows for generation of a user delegation key which can be used to sign SAS tokens	BuiltInRole	Storage	View

Showing 1 - 6 of 6 results.

[Review + assign](#) [Previous](#) [Next](#) [Feedback](#)

- Next, select the **spn-databricks** as member -> click on Next. (here, spn-databricks is our created service principle)

Add role assignment ...

Role **Members** Conditions Review + assign

Selected role Storage Blob Data Contributor

Assign access to User, group, or service principal Managed identity

Members [+ Select members](#)

Name	Object ID	Type
No members selected		

Description Optional

[Review + assign](#) [Previous](#) [Next](#) [Select](#) [Close](#)

Add role assignment ...

Role **Members** Conditions Review + assign

Selected role Storage Blob Data Contributor

Assign access to User, group, or service principal Managed identity

Members [+ Select members](#)

Name	Object ID	Type
spn-databricks	159ee81c-3235-407a-999b-2e3c330d6...	App

Description Optional

[Review + assign](#) [Previous](#) [Next](#)

- Click on **Review + assign** button and complete the role assignment

Microsoft Azure

Search resources, services, and docs (G+)

Home > adlsgen2hsingh | Access Control (IAM) >

Add role assignment

[Role](#) [Members](#) [Conditions](#) [Review + assign](#)

Role	Storage Blob Data Contributor						
Scope	/subscriptions/91a74185-4d48-411e-bc73-bb2295b0a30c/resourceGroups/rg-ncpl-hsingh/providers/Microsoft.Storage/storageAccounts/adlsgen2hsingh						
Members	<table border="1"> <thead> <tr> <th>Name</th> <th>Object ID</th> <th>Type</th> </tr> </thead> <tbody> <tr> <td>spn-databricks</td> <td>159ee81c-3235-407a-999b-2e3c3330d682</td> <td>App</td> </tr> </tbody> </table>	Name	Object ID	Type	spn-databricks	159ee81c-3235-407a-999b-2e3c3330d682	App
Name	Object ID	Type					
spn-databricks	159ee81c-3235-407a-999b-2e3c3330d682	App					
Description	No description						
Condition	None						

[Review + assign](#) [Previous](#) [Next](#)

- Next, Move to Home Page of Azure account -> Search for the **Key Vault** in the search bar.

Microsoft Azure

Create a resource Microsoft Entra ID Key vault

Services (10) Marketplace (31) More (4)

Key vaults

SSH keys

Microsoft Entra authentication methods

BitLocker Keys

Key Vault

Language service

Document Intelligence (form recognizer)

Recent Favorite

adlsgen2hsingh blobstoragehsingh kvhsingh rg-ncpl-hsingh

Last Viewed

7 minutes ago
3 hours ago
3 days ago
3 days ago

- Here, the previously created Secret is available (with name as **kvhsingh**), the click on the available secret. (we can create new secret as well but here I am using the previous one).

Showing 1 to 1 of 1 records.

Name ↑	Type ↑↓	Resource group ↑↓	Location ↑↓
kvhsingh	Key vault	rg-ncpl-hsingh	Australia East

- Next, select the **Secrets** option under the Objects -> click on **+ Generate/Import** option.

- Name the secret (in my case, I named it as **appid**) and enter the **Application ID & Directory ID** in the **secret value** field -> click on **Create** button.

Microsoft Azure

Search resources, services, and docs (G+/)

Copilot

Home > Key vaults > kvhsingh | Secrets >

Create a secret

Upload options: Manual

Name * ⓘ: appid

Secret value * ⓘ:

Content type (optional):

Set activation date ⓘ:

Set expiration date ⓘ:

Enabled: Yes

Tags: 0 tags

Create **Cancel**

- Next again generate a new secret and name the secret (in my case, I named it as **appsecret**) and enter the **secret value** from Service Principle created secret in the **secret value** field -> click on **Create** button.

Microsoft Azure

Search resources, services, and docs (G+/)

Copilot

Home > Key vaults > kvhsingh | Secrets >

Create a secret

Upload options: Manual

Name * ⓘ: appsecret

Secret value * ⓘ:

Content type (optional):

Set activation date ⓘ:

Set expiration date ⓘ:

Enabled: Yes

Tags: 0 tags

Create **Cancel**

- Now, we can see the both newly created secrets.

Name	Type	Status	Expiration date
appid		✓ Enabled	
appsecret		✓ Enabled	
azure-sqldb-password		✓ Enabled	
azure-sqldb-username		✓ Enabled	
onprem-system-password		✓ Enabled	

- Next, Go to Key Vault and open the **Properties** for new created **Key Vault (kvhsingh)** and **copy the Vault URI and Resource ID**.

We can use these values to create new scope in databricks to access the storage account.

kvhsingh | Properties

Key vault

Search Save Discard changes Refresh

Keys Secrets Certificates

Settings

Access configuration Networking Microsoft Defender for Cloud

Properties

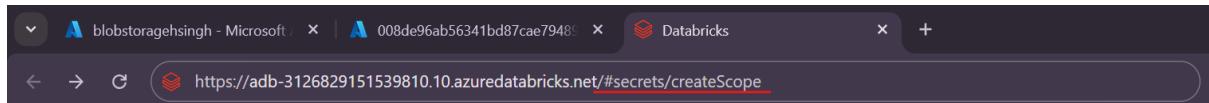
Locks Monitoring Alerts Metrics Diagnostic settings

Name	kvhsingh
Sku (Pricing tier)	Standard
Location	australiaeast
Vault URI	https://kvhsingh.vault.azure.net/
Resource ID	/subscriptions/91a74185-4d48-411e-bc73-bb2295b0a30c/resourceGroups/rg-ncpl-hs...
Subscription ID	91a74185-4d48-411e-bc73-bb2295b0a30c
Subscription Name	Azure subscription
Directory ID	33d03180-c464-491c-83c0-b95da1d65d49
Directory Name	Default Directory
Soft-delete	Soft delete has been enabled on this key vault
Days to retain deleted vaults	90

Databricks Scope Creation

➤ Steps to create a new scope in databricks using key vault properties:

- Now, move to the resources in Azure account and open the Data bricks workspace.
Go to the URL section of browser and remove all the content after **.net/** and add the below text:
#secrets/createScope



- Now, the new Page is opened in the Databrick workspace to create a **Secret scope**.
Enter the name of Scope (in my case, I named it as *adlsstgconnection*), and select the manage principle as **Creator**,

A screenshot of the "Create Secret Scope" dialog box. The title bar says "HomePage / Create Secret Scope" and "Create Secret Scope | Cancel Create".

A store for secrets that is identified by a name and backed by a specific store type. [Learn more](#)

Scope Name ?
adlsstgconnection

Manage Principal ?
Creator

Azure Key Vault ?

DNS Name
https://xxx.vault.azure.net/

Resource ID
/subscriptions/xxxxxx/...

- Next, Go to Key Vault and open the **Properties** for new created **Key Vault** (**kvhsingh**) and **copy** the **Vault URI** and **Resource ID**.

The screenshot shows the 'kvhsingh | Properties' page in the Azure portal. The left sidebar has 'Properties' selected. The main area shows key vault details: Name (kvhsingh), Sku (Standard), Location (australiaeast), and various configuration settings like Days to retain deleted vaults (90). The 'Vault URI' field contains `https://kvhsingh.vault.azure.net/`, and the 'Resource ID' field contains `/subscriptions/91a74185-4d48-411e-bc73-bb2295b0a30c/resourceGroups/rg-ncpl-hsingh/providers/Microsoft.KeyVault/vaults/kvhsingh`. A yellow box highlights the 'Vault URI' field, and another yellow box highlights the 'Resource ID' field. A 'Copy to clipboard' button is located to the right of the 'Vault URI' field.

- After copy, again Move to the Create scope page of databricks and **paste** the copied key vault **Vault URI** into **DNS Name** and **Resource ID** into **Resource ID** fields.

Then, click on **Create button**.

- After clicking, verify the Scope creation via message popup, and click on OK.

The screenshot shows the 'Create Secret Scope' dialog. It includes fields for 'Scope Name' (adlsstgconnection), 'Manage Principal' (Creator), 'Azure Key Vault' (DNS Name: `https://kvhsingh.vault.azure.net/`, Resource ID: `/subscriptions/ca6fef37-a187-4dc8-b638-ab3267c398e1/resourceGroups/hsingh-rg`), and a 'Create' button which is highlighted with a red box.

MyUtilityFunctions Notebook

– List Available Secret Scopes

- Retrieves a list of all secret scopes available in the Databricks workspace using dbutils.secrets.listScopes().
- Code:

```
dbutils.secrets.listScopes()
```



A screenshot of a Databricks notebook cell. The cell contains the following code:

```
# List all secret scopes available in the Databricks workspace
dbutils.secrets.listScopes()
```

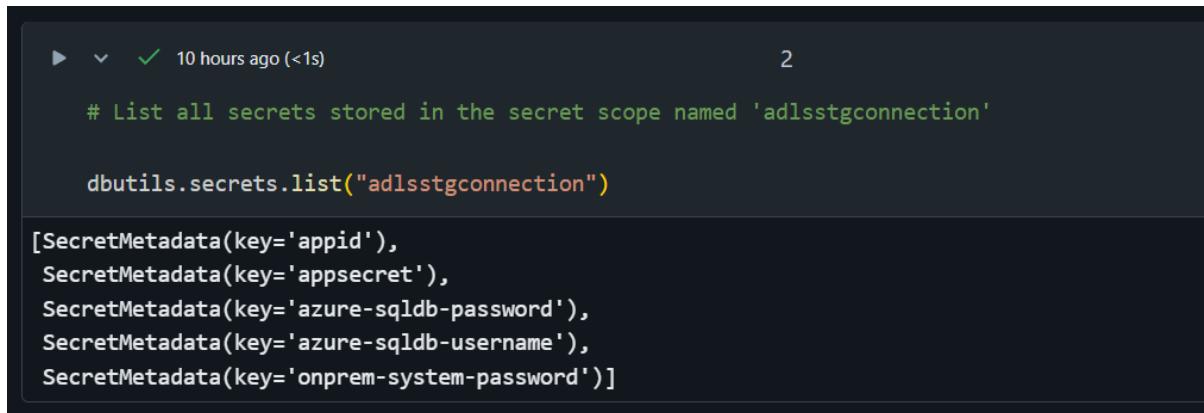
The output of the cell is:

```
[SecretScope(name='adlsstgconnection')]
```

– List Secrets in a Specific Scope

- Lists all the secrets stored under the secret scope named "adlsstgconnection".
- Code:

```
dbutils.secrets.list("adlsstgconnection")
```



A screenshot of a Databricks notebook cell. The cell contains the following code:

```
# List all secrets stored in the secret scope named 'adlsstgconnection'
dbutils.secrets.list("adlsstgconnection")
```

The output of the cell is:

```
[SecretMetadata(key='appid'),
SecretMetadata(key='appsecret'),
SecretMetadata(key='azure-sqldb-password'),
SecretMetadata(key='azure-sqldb-username'),
SecretMetadata(key='onprem-system-password')]
```

– Mount ADLS Storage to Databricks

- Checks if the given mount point /mnt/project2-container already exists.
- If not, it mounts the Azure Data Lake Storage (ADLS) container using OAuth credentials securely retrieved from Databricks secret scope.
- The configuration ensures secure, token-based authentication.

- Code:

```

mount_point = "/mnt/project2-container"

if not any(mount.mountPoint == mount_point for mount in
dbutils.fs.mounts()):
    configs = {
        "fs.azure.account.auth.type": "OAuth",
        "fs.azure.account.oauth.provider.type": "org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider",
        "fs.azure.account.oauth2.client.id": dbutils.secrets.get(scope="adlsstgconnection", key="appid"),
        "fs.azure.account.oauth2.client.secret": dbutils.secrets.get(scope="adlsstgconnection", key="appsecret"),
        "fs.azure.account.oauth2.client.endpoint": "https://login.microsoftonline.com/78be0424-72cf-4e7f-8a71-966bd89c103b/oauth2/token"
    }

    dbutils.fs.mount(
        source="abfss://project2-
container@hsinghadls.dfs.core.windows.net/",
        mount_point=mount_point,
        extra_configs=configs
    )
    print(f"Mounted successfully at {mount_point}")
else:
    print(f"{mount_point} is already mounted.")

```

```

1 # Check if mount point exists or not if not then create new
2 # Mount Azure Data Lake Storage (ADLS) container to Databricks File System (DBFS)
3 # using OAuth authentication and credentials from secret scope
4
5 mount_point = "/mnt/project2-container"
6
7 if not any(mount.mountPoint == mount_point for mount in dbutils.fs.mounts()):
8     configs = {
9         "fs.azure.account.auth.type": "OAuth",
10        "fs.azure.account.oauth.provider.type": "org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider",
11        "fs.azure.account.oauth2.client.id": dbutils.secrets.get(scope="adlsstgconnection", key="appid"),
12        "fs.azure.account.oauth2.client.secret": dbutils.secrets.get(scope="adlsstgconnection", key="appsecret"),
13        "fs.azure.account.oauth2.client.endpoint": "https://login.microsoftonline.com/78be0424-72cf-4e7f-8a71-966bd89c103b/oauth2/token"
14    }
15
16    dbutils.fs.mount(
17        source="abfss://project2-container@hsinghadls.dfs.core.windows.net/",
18        mount_point=mount_point,
19        extra_configs=configs
20    )
21    print(f"Mounted successfully at {mount_point}")
22 else:
23     print(f"{mount_point} is already mounted.")
24

```

Output Terminal Debug console

/mnt/project2-container is already mounted.

– List all files to check mount point

- Retrieves and displays a list of files and directories located at the root level of the mounted ADLS container /mnt/project2-container.
- Code:

```
dbutils.fs.ls("/mnt/project2-container")
```

```

# List all files and directories in the root of the mounted container '/mnt/project2-container'
dbutils.fs.ls("/mnt/project2-container")

[FileInfo(path='dbfs:/mnt/project2-container/Bronze_Layer/', name='Bronze_Layer/', size=0, modificationTime=1745394195000),
 FileInfo(path='dbfs:/mnt/project2-container/Gold_Layer/', name='Gold_Layer/', size=0, modificationTime=1745524456000),
 FileInfo(path='dbfs:/mnt/project2-container/Silver_Layer/', name='Silver_Layer/', size=0, modificationTime=1745523623000)]

```

+ Code + Text

– Function to Generate File Path Based on Current Date

- Creates a dynamic file path based on the current date and a given folder name (like Bronze_Layer), organizing data by year/month/day for easy partitioning.
- Code:

```
from datetime import datetime
```

```
def get_file_path(folder_name):
    current_date = datetime.now().strftime("%Y-%m-%d")
```

```

year, month, day = current_date.split("-")
file_path = f"/mnt/project2-
container/{folder_name}/{year}/{month}/{day}/"

return file_path

```

```

# Function to generate a dynamic file path based on folder name and current date

from datetime import datetime

def get_file_path(folder_name):
    current_date = datetime.now().strftime("%Y-%m-%d")
    year, month, day = current_date.split("-")
    file_path = f"/mnt/project2-container/{folder_name}/{year}/{month}/{day}/"

    return file_path

```

– Function to Read Data from ADLS

- Reads data from a specified path in ADLS using Spark, allowing flexibility in file format (like CSV or Parquet).
- Code:

```

def get_adls_data(path, file_format, file_name):
    df = spark.read.format(file_format).option("header", True).load(path +
file_name)

return df

```

```

# Function to read data from ADLS at the given path and format using Spark

def get_adls_data(path, file_format, file_name):
    df = spark.read.format(file_format).option("header", True).load(path + file_name)

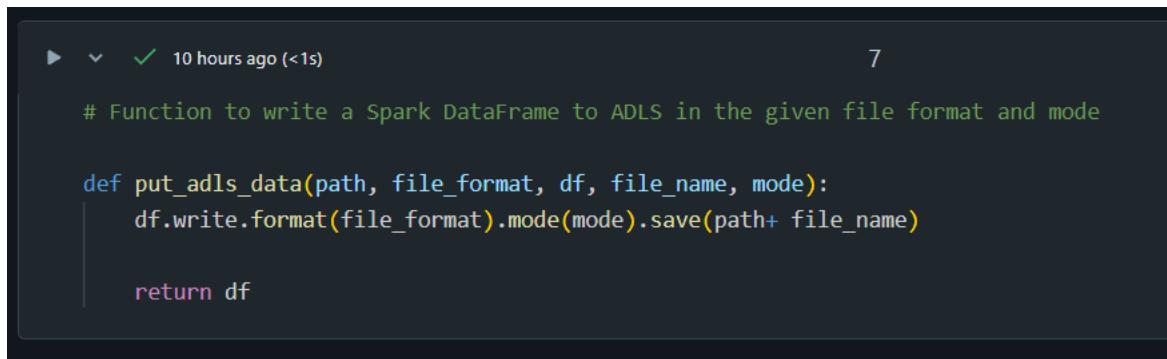
    return df

```

– Function to Write Data to ADLS

- Saves a DataFrame to a specified path and file format in ADLS, supporting different write modes such as append or overwrite.
- Code:

```
def put_adls_data(path, file_format, df, file_name, mode):  
    df.write.format(file_format).mode(mode).save(path+ file_name)  
  
    return df
```



A screenshot of a Jupyter Notebook cell. The cell contains Python code for writing a DataFrame to ADLS. The code defines a function `put_adls_data` that takes four parameters: `path`, `file_format`, `df`, and `mode`. It uses the `df.write.format(file_format).mode(mode).save(path+ file_name)` method to save the DataFrame. The cell has a green checkmark icon and the text "10 hours ago (<1s)". The cell number "7" is in the top right corner.

```
# Function to write a Spark DataFrame to ADLS in the given file format and mode  
  
def put_adls_data(path, file_format, df, file_name, mode):  
    df.write.format(file_format).mode(mode).save(path+ file_name)  
  
    return df
```

– List CSV Files in Bronze Layer

- Lists all files in the Bronze layer folder for the current date.
- Filters the results to extract only file names and prints them.
- Code:

```
def list_csv_files_in_bronze(bronze_path):  
    files = dbutils.fs.ls(bronze_path)  
    file_names = [file.name for file in files if file.name]  
    return file_names  
  
files = list_csv_files_in_bronze(get_file_path("Bronze_Layer"))  
print(files)
```

▶ ✓ 10 hours ago (<1s) 8

```
# Function to list all file names in the Bronze layer for the current date
# List and print all files in the dynamically generated Bronze layer path

def list_csv_files_in_bronze(bronze_path):
    files = dbutils.fs.ls(bronze_path)
    file_names = [file.name for file in files if file.name]
    return file_names

files = list_csv_files_in_bronze(get_file_path("Bronze_Layer"))
print(files)

['accounts.csv', 'customers.csv', 'loan_payments.csv', 'loans.csv', 'transactions.csv']
```

----- + Code + Text -----

Silver Layer Notebook

– Utility Functions

- Utilizes pre-defined utility functions for modular code reuse and simplifies ADLS interactions.
- Code:

```
%run "/Workspace/Project2/MyUtilityFunctions"
```



```
06:30 AM (1s) 1 Python :: 1 :: 1
%run "/Workspace/Project2/MyUtilityFunctions"

[SecretScope(name='adlssstgconnection')]

[SecretMetadata(key='appid'),
 SecretMetadata(key='appsecret'),
 SecretMetadata(key='azure-sqldb-password'),
 SecretMetadata(key='azure-sqldb-username'),
 SecretMetadata(key='onprem-system-password')]

/mnt/project2-container is already mounted.

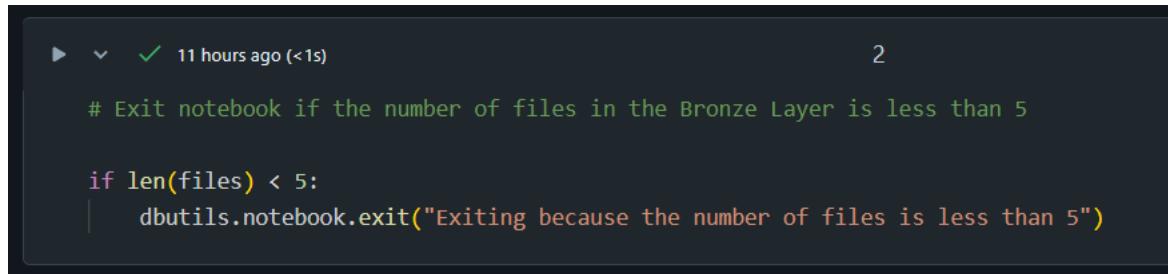
[FileInfo(path='dbfs:/mnt/project2-container/Bronze_Layer/', name='Bronze_Layer/', size=0, modificationTime=1745394195000),
 FileInfo(path='dbfs:/mnt/project2-container/Gold_Layer/', name='Gold_Layer/', size=0, modificationTime=1745524456000),
 FileInfo(path='dbfs:/mnt/project2-container/Silver_Layer/', name='Silver_Layer/', size=0, modificationTime=1745523623000)]

['accounts.csv', 'customers.csv', 'loan_payments.csv', 'loans.csv', 'transactions.csv']
```

– File Count Check

- Validates the presence of at least 5 files in the Bronze Layer before proceeding with data processing.
- Code:

```
if len(files) < 5:
    dbutils.notebook.exit("Exiting because the number of files is less than 5")
```



```
11 hours ago (<1s) 2
# Exit notebook if the number of files in the Bronze Layer is less than 5

if len(files) < 5:
    dbutils.notebook.exit("Exiting because the number of files is less than 5")
```

– CSV Data Ingestion

- Loads each file from the Bronze Layer into a separate DataFrame using utility functions.
- Code:

```

raw_data_dfs=[]

for i in range(0,len(files)):
    raw_data_dfs.append(get_adls_data(get_file_path("Bronze_Layer"), "csv",
files[i]))

display(raw_data_dfs[4])

```

```

# Load all CSV files from Bronze Layer into a list of DataFrames

raw_data_dfs=[]

for i in range(0,len(files)):
    raw_data_dfs.append(get_adls_data(get_file_path("Bronze_Layer"), "csv", files[i]))

display(raw_data_dfs[4])

```

▶ (6) Spark Jobs

Table +

	Δ transaction_id	Δ account_id	Δ transaction_date	Δ transaction_amount	Δ transaction_type
1	1	45	01-01-2024	100.5	Deposit
2	2	12	02-01-2024	200.75	Withdrawal
3	3	78	03-01-2024	150	Deposit
4	4	34	04-01-2024	300.25	Withdrawal
5	5	56	05-01-2024	250	Deposit

– Null ID Filtering

- Filters out rows with null or blank primary key fields to ensure data quality before transformations.
- Code:

```

from pyspark.sql.functions import trim

def get_filtered_df(df, id_column):
    return df.filter(trim(id_column).isNotNull())

filtered_dfs = []

filtered_dfs.append(get_filtered_df(raw_data_dfs[0],
raw_data_dfs[0].account_id))
filtered_dfs.append(get_filtered_df(raw_data_dfs[1],
raw_data_dfs[1].customer_id))
filtered_dfs.append(get_filtered_df(raw_data_dfs[2],
raw_data_dfs[2].payment_id))

```

```

filtered_dfs.append(get_filtered_df(raw_data_dfs[3],
raw_data_dfs[3].loan_id))
filtered_dfs.append(get_filtered_df(raw_data_dfs[4],
raw_data_dfs[4].transaction_id))

display(filtered_dfs[4])

```

The screenshot shows a Jupyter Notebook cell with the following content:

```

1 # Import trim function and define function to filter out rows with null or blank ID columns
2 # Apply filters on specific ID columns from each dataset
3
4 from pyspark.sql.functions import trim
5
6 def get_filtered_df(df, id_column):
7     return df.filter(trim(id_column).isNotNull())
8
9 filtered_dfs = []
10
11 filtered_dfs.append(get_filtered_df(raw_data_dfs[0], raw_data_dfs[0].account_id))
12 filtered_dfs.append(get_filtered_df(raw_data_dfs[1], raw_data_dfs[1].customer_id))
13 filtered_dfs.append(get_filtered_df(raw_data_dfs[2], raw_data_dfs[2].payment_id))
14 filtered_dfs.append(get_filtered_df(raw_data_dfs[3], raw_data_dfs[3].loan_id))
15 filtered_dfs.append(get_filtered_df(raw_data_dfs[4], raw_data_dfs[4].transaction_id))
16
17 display(filtered_dfs[4])
18

```

Below the code, there is a table output:

Table +

	transaction_id	account_id	transaction_date	transaction_amount	transaction_type
1	1	45	01-01-2024	100.5	Deposit
2	2	12	02-01-2024	200.75	Withdrawal
3	3	78	03-01-2024	150	Deposit

↓ 101 rows | 0.40s runtime

– Remove Duplicate Records

- Applies the `.distinct()` transformation to eliminate duplicate rows.
- Code:

```

def get_distinct_df(df):
    return df.distinct()

distinct_dfs = []

for i in range(0, len(filtered_dfs)):
    distinct_dfs.append(get_distinct_df(filtered_dfs[i]))

display(distinct_dfs[4])

```

11 hours ago (1s) 5

```

1 # Define function to return distinct records
2 # Create a list of distinct (deduplicated) DataFrames
3
4 def get_distinct_df(df):
5     return df.distinct()
6
7 distinct_dfs = []
8
9 for i in range(0, len(filtered_dfs)):
10    distinct_dfs.append(get_distinct_df(filtered_dfs[i]))
11
12 display(distinct_dfs[4])

```

Table +

	A _C transaction_id	A _C account_id	A _C transaction_date	A _C transaction_amount	A _C transaction_type
1	83	82	23-03-2024	150	Deposit
2	81	70	21-03-2024	100.5	Deposit
3	90	38	30-03-2024	375.25	Withdrawal
4	23	88	23-01-2024	150	Deposit
5	93	79	02-04-2024	150	Deposit
6	69	59	09-03-2024	325	Deposit
7	61	52	01-03-2024	100.5	Deposit

– Define Fill Values

- Uses dictionary-based default values to replace nulls with domain-appropriate placeholders.
- Code:

```

fillna_values = {
    "accounts.csv": {
        "customer_id": "-1",
        "account_type": "N/A",
        "balance": "-1.00"
    },
    "customers.csv": {
        "first_name": "Unknown",
        "last_name": "Unknown",
        "address": "Unknown",
        "city": "Unknown",
        "state": "Unknown",
        "zip": "Unknown"
    }
},

```

```

"loan_payments.csv":
{
    "loan_id": "-1",
    "payment_date": "01-01-1900",
    "payment_amount": "-1.00"
},

"loans.csv":
{
    "customer_id": "-1",
    "loan_amount": "-1.00",
    "interest_rate": "-1.00",
    "loan_term": "-1"
},

"transactions.csv":
{
    "account_id": "-1",
    "transaction_date": "01-01-1900",
    "transaction_amount": "0.00",
    "transaction_type": "N/A"
}
}

```



The screenshot shows a Jupyter Notebook cell with the following code:

```

1 # Define dictionary to fill missing values with default placeholders
2
3 fillna_values = [
4     "accounts.csv": {
5         "customer_id": "-1", "account_type": "N/A", "balance": "-1.00"
6     },
7     "customers.csv": {
8         "first_name": "Unknown", "last_name": "Unknown", "address": "Unknown", "city": "Unknown", "state": "Unknown", "zip": "Unknown"
9     },
10    "loan_payments.csv": {
11        "loan_id": "-1", "payment_date": "01-01-1900", "payment_amount": "-1.00"
12    },
13    "loans.csv": {
14        "customer_id": "-1", "loan_amount": "-1.00", "interest_rate": "-1.00", "loan_term": "-1"
15    },
16    "transactions.csv": {
17        "account_id": "-1", "transaction_date": "01-01-1900", "transaction_amount": "0.00", "transaction_type": "N/A"
18    }
19]
20
21
22
23
24 ]

```

The code defines a dictionary `fillna_values` where each key is a CSV file name and its value is a dictionary of column names and their default placeholder values. The code is annotated with line numbers from 1 to 24.

– Fill Missing Values

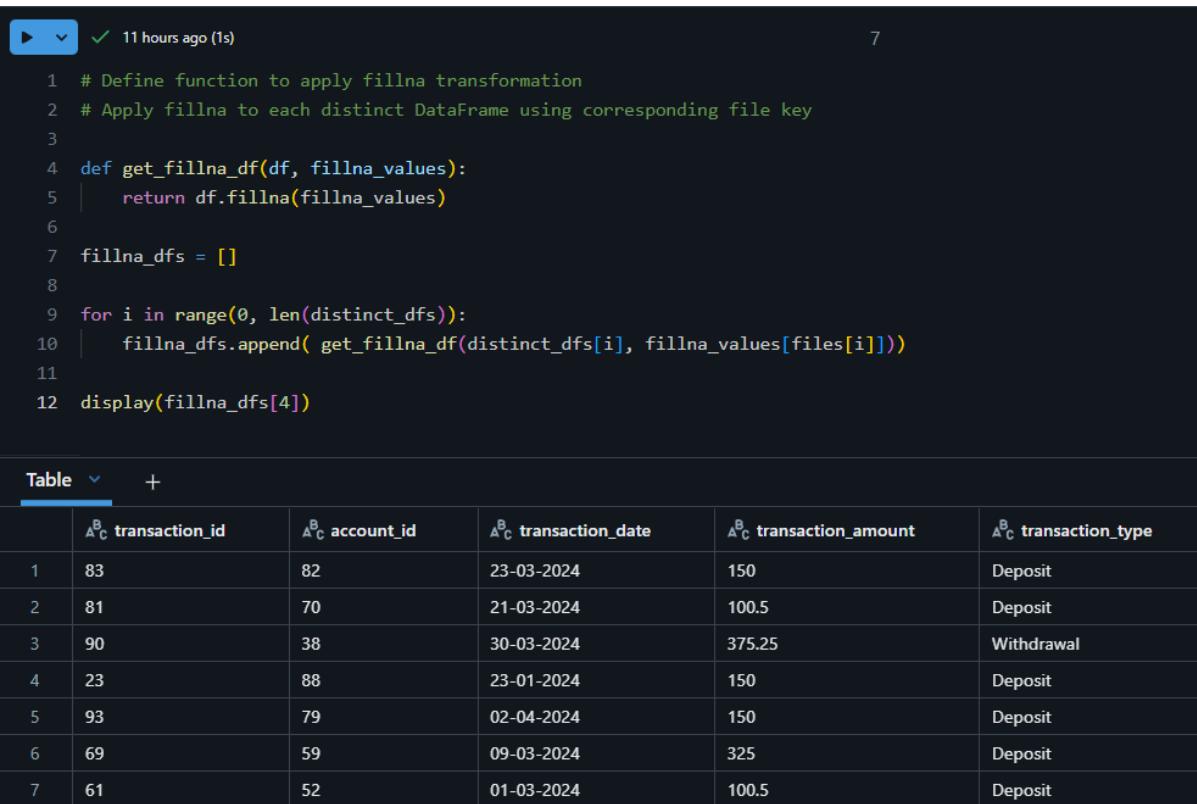
- Applies .fillna() method using predefined values to prepare data for type conversion.
- Code:

```
def get_fillna_df(df, fillna_values):
    return df.fillna(fillna_values)

fillna_dfs = []

for i in range(0, len(distinct_dfs)):
    fillna_dfs.append(get_fillna_df(distinct_dfs[i], fillna_values[files[i]]))

display(fillna_dfs[4])
```



The screenshot shows a Jupyter Notebook cell with the following details:

- Timestamp: 11 hours ago (1s)
- Cell ID: 7
- Code content:

```
1 # Define function to apply fillna transformation
2 # Apply fillna to each distinct DataFrame using corresponding file key
3
4 def get_fillna_df(df, fillna_values):
5     return df.fillna(fillna_values)
6
7 fillna_dfs = []
8
9 for i in range(0, len(distinct_dfs)):
10    fillna_dfs.append(get_fillna_df(distinct_dfs[i], fillna_values[files[i]]))
11
12 display(fillna_dfs[4])
```

Below the code cell is a table titled "Table" showing the result of the displayed DataFrame:

	transaction_id	account_id	transaction_date	transaction_amount	transaction_type
1	83	82	23-03-2024	150	Deposit
2	81	70	21-03-2024	100.5	Deposit
3	90	38	30-03-2024	375.25	Withdrawal
4	23	88	23-01-2024	150	Deposit
5	93	79	02-04-2024	150	Deposit
6	69	59	09-03-2024	325	Deposit
7	61	52	01-03-2024	100.5	Deposit

– Data Type Casting

- Converts fields into proper data types (e.g., Integer, Double, Date) required for analytical operations.
- Code:

```
from pyspark.sql.types import IntegerType, DoubleType, StringType,
    DateType
from pyspark.sql.functions import to_date
```

```

cast_accounts_df = fillna_dfs[0].select(
    fillna_dfs[0].account_id.cast(IntegerType()),
    fillna_dfs[0].customer_id.cast(IntegerType()),
    fillna_dfs[0].account_type.cast(StringType()),
    fillna_dfs[0].balance.cast(DoubleType())
)
cast_customers_df = fillna_dfs[1].select(
    fillna_dfs[1].customer_id.cast(IntegerType()),
    fillna_dfs[1].first_name.cast(StringType()),
    fillna_dfs[1].last_name.cast(StringType()),
    fillna_dfs[1].address.cast(StringType()),
    fillna_dfs[1].city.cast(StringType()),
    fillna_dfs[1].state.cast(StringType()),
    fillna_dfs[1].zip.cast(StringType())
)
cast_loan_payments_df = fillna_dfs[2].select(
    fillna_dfs[2].payment_id.cast(IntegerType()),
    fillna_dfs[2].loan_id.cast(IntegerType()),
    to_date(fillna_dfs[2].payment_date, "dd-MM-yyyy").alias("payment_date"),
    fillna_dfs[2].payment_amount.cast(DoubleType())
)
cast_loans_df = fillna_dfs[3].select(
    fillna_dfs[3].loan_id.cast(IntegerType()),
    fillna_dfs[3].customer_id.cast(IntegerType()),
    fillna_dfs[3].loan_amount.cast(DoubleType()),
    fillna_dfs[3].interest_rate.cast(DoubleType()),
    fillna_dfs[3].loan_term.cast(IntegerType())
)
cast_transactions_df = fillna_dfs[4].select(
    fillna_dfs[4].transaction_id.cast(IntegerType()),
    fillna_dfs[4].account_id.cast(IntegerType()),
    to_date(fillna_dfs[4].transaction_date, "dd-MM-
    yyyy").alias("transaction_date"),
    fillna_dfs[4].transaction_amount.cast(DoubleType()),
    fillna_dfs[4].transaction_type.cast(StringType())
)
display(cast_transactions_df)

```

11 hours ago (1s)

8

```
1 # Import data types and transformation functions
2 # Cast columns in all dataframes to appropriate types using Spark data types
3 # Example shown for one dataset (cast_accounts_df)
4
5 from pyspark.sql.types import IntegerType, DoubleType, StringType, DateType
6 from pyspark.sql.functions import to_date
7
8
9 cast_accounts_df = fillna_dfs[0].select(
10     fillna_dfs[0].account_id.cast(IntegerType()),
11     fillna_dfs[0].customer_id.cast(IntegerType()),
12     fillna_dfs[0].account_type.cast(StringType()),
13     fillna_dfs[0].balance.cast(DoubleType())
14 )
15
16 cast_customers_df = fillna_dfs[1].select(
17     fillna_dfs[1].customer_id.cast(IntegerType()),
18     fillna_dfs[1].first_name.cast(StringType()),
19     fillna_dfs[1].last_name.cast(StringType()),
20     fillna_dfs[1].address.cast(StringType()),
21     fillna_dfs[1].city.cast(StringType()),
22     fillna_dfs[1].state.cast(StringType()),
23     fillna_dfs[1].zip.cast(StringType())
24 )
25
26 cast_loan_payments_df = fillna_dfs[2].select(
27     fillna_dfs[2].payment_id.cast(IntegerType()),
28     fillna_dfs[2].loan_id.cast(IntegerType()),
29     to_date(fillna_dfs[2].payment_date, "dd-MM-yyyy").alias("payment_date"),
30     fillna_dfs[2].payment_amount.cast(DoubleType())
31 )
32
33 cast_loans_df = fillna_dfs[3].select(
34     fillna_dfs[3].loan_id.cast(IntegerType()),
35     fillna_dfs[3].customer_id.cast(IntegerType()),
36     fillna_dfs[3].loan_amount.cast(DoubleType()),
37     fillna_dfs[3].interest_rate.cast(DoubleType()),
38     fillna_dfs[3].loan_term.cast(IntegerType())
39 )
40
41 cast_transactions_df = fillna_dfs[4].select(
42     fillna_dfs[4].transaction_id.cast(IntegerType()),
43     fillna_dfs[4].account_id.cast(IntegerType()),
44     to_date(fillna_dfs[4].transaction_date, "dd-MM-yyyy").alias("transaction_date"),
45     fillna_dfs[4].transaction_amount.cast(DoubleType()),
46     fillna_dfs[4].transaction_type.cast(StringType())
47 )
48
49 display(cast_transactions_df)
```

Table +

1 transaction_id	1 account_id	transaction_date	1.2 transaction_amount	A transaction_type
1	83	82	2024-03-23	150 Deposit
2	81	70	2024-03-21	100.5 Deposit
3	90	38	2024-03-30	375.25 Withdrawal
4	23	88	2024-01-23	150 Deposit
5	93	79	2024-04-02	150 Deposit
6	69	59	2024-03-09	325 Deposit
7	61	52	2024-03-01	100.5 Deposit
8	87	93	2024-03-27	225.5 Deposit
9	17	99	2024-01-17	225.5 Deposit

- Column Renaming

- Renames columns across all datasets into consistent CamelCase format for clarity and schema alignment.
- Code:

```
accounts_df = cast_accounts_df.withColumnRenamed("account_id",  
"AccountId")\  
    .withColumnRenamed("customer_id", "CustomerId")\  
    .withColumnRenamed("account_type", "AccountType")\  
    .withColumnRenamed("balance", "Balance")  
  
customers_df = cast_customers_df.withColumnRenamed("customer_id",  
"CustomerId")\  
    .withColumnRenamed("first_name", "FirstName")\  
    .withColumnRenamed("last_name", "LastName")\  
    .withColumnRenamed("address", "Address")\  
    .withColumnRenamed("city", "City")\  
    .withColumnRenamed("state", "State")\  
    .withColumnRenamed("zip", "Zip")  
  
loan_payments_df =  
    cast_loan_payments_df.withColumnRenamed("payment_id", "PaymentId")\  
        .withColumnRenamed("loan_id", "LoanId")\  
        .withColumnRenamed("payment_date", "PaymentDate")\  
        .withColumnRenamed("payment_amount", "PaymentAmount")  
  
loans_df = cast_loans_df.withColumnRenamed("loan_id", "LoanId")\  
    .withColumnRenamed("customer_id", "CustomerId")\  
    .withColumnRenamed("loan_amount", "LoanAmount")\  
    .withColumnRenamed("interest_rate", "InterestRate")\  
    .withColumnRenamed("loan_term", "LoanTerm")  
  
transactions_df =  
    cast_transactions_df.withColumnRenamed("transaction_id",  
"TransactionId")\  
        .withColumnRenamed("account_id", "AccountId")\  
        .withColumnRenamed("transaction_date", "TransactionDate")\  
        .withColumnRenamed("transaction_amount", "TransactionAmount")\  
        .withColumnRenamed("transaction_type", "TransactionType")  
  
display(transactions_df)
```

✓ 11 hours ago (<1s)

9

```
1 # Rename columns for better readability and consistency using CamelCase
2
3 accounts_df = cast_accounts_df.withColumnRenamed("account_id", "AccountId")\
4     .withColumnRenamed("customer_id", "CustomerId")\
5     .withColumnRenamed("account_type", "AccountType")\
6     .withColumnRenamed("balance", "Balance")
7
8 customers_df = cast_customers_df.withColumnRenamed("customer_id", "CustomerId")\
9     .withColumnRenamed("first_name", "FirstName")\
10    .withColumnRenamed("last_name", "LastName")\
11    .withColumnRenamed("address", "Address")\
12    .withColumnRenamed("city", "City")\
13    .withColumnRenamed("state", "State")\
14    .withColumnRenamed("zip", "Zip")
15
16 loan_payments_df = cast_loan_payments_df.withColumnRenamed("payment_id", "PaymentId")\
17     .withColumnRenamed("loan_id", "LoanId")\
18     .withColumnRenamed("payment_date", "PaymentDate")\
19     .withColumnRenamed("payment_amount", "PaymentAmount")
20
21 loans_df = cast_loans_df.withColumnRenamed("loan_id", "LoanId")\
22     .withColumnRenamed("customer_id", "CustomerId")\
23     .withColumnRenamed("loan_amount", "LoanAmount")\
24     .withColumnRenamed("interest_rate", "InterestRate")\
25     .withColumnRenamed("loan_term", "LoanTerm")
26
27 transactions_df = cast_transactions_df.withColumnRenamed("transaction_id", "TransactionId")\
28     .withColumnRenamed("account_id", "AccountId")\
29     .withColumnRenamed("transaction_date", "TransactionDate")\
30     .withColumnRenamed("transaction_amount", "TransactionAmount")\
31     .withColumnRenamed("transaction_type", "TransactionType")
32
33 display(transactions_df)
```

Table +

	TransactionId	AccountId	TransactionDate	TransactionAmount	TransactionType
1	83	82	2024-03-23	150	Deposit
2	81	70	2024-03-21	100.5	Deposit
3	90	38	2024-03-30	375.25	Withdrawal
4	23	88	2024-01-23	150	Deposit
5	93	79	2024-04-02	150	Deposit

– Table Joins

- Joins all five processed datasets into one unified view using shared keys (CustomerId, LoanId, AccountId).
- Code:

```
joined_df = accounts_df.join(customers_df, "CustomerId", "inner")\
    .join(loans_df, "CustomerId", "inner")\
    .join(loan_payments_df, "LoanId", "inner")\
```

```

    .join(transactions_df, "AccountId", "inner")\
    .distinct()\
    .select("AccountId", "CustomerId", "Balance", "LoanId", "LoanAmount",
"PaymentId", "PaymentAmount", "PaymentDate", "TransactionId",
"TransactionAmount", "TransactionDate")\

```

```
display(joined_df)
```

```

1 # Join all processed DataFrames to create a unified view for analysis and to generate a PowerBI report using it.
2
3 joined_df = accounts_df.join(customers_df, "CustomerId", "inner")\
4     .join(loans_df, "CustomerId", "inner")\
5     .join(loan_payments_df, "LoanId", "inner")\
6     .join(transactions_df, "AccountId", "inner")\
7     .distinct()\
8     .select("AccountId", "CustomerId", "Balance", "LoanId", "LoanAmount", "PaymentId",\
9     | "PaymentAmount", "PaymentDate", "TransactionId", "TransactionAmount", "TransactionDate")\
10
11
12 display(joined_df)
13

```

Table +

	1 ² ₃ AccountId	1 ² ₃ CustomerId	1.2 Balance	1 ² ₃ LoanId	1.2 LoanAmount	1 ² ₃ PaymentId	1.2 PaymentAmount
1	55	63	425.75	55	25000.75	14	750
2	33	85	150.25	33	15000.25	100	1000
3	24	11	2600	24	30000	93	4700
4	10	5	1600.5	10	27500.5	47	2400

– Write to Silver Layer

- Writes all clean, typed, and joined DataFrames to the Silver Layer in Parquet format for further analysis.
- Code:

```

put_adls_data("/mnt/project2-container/Silver_Layer/", "parquet",
accounts_df, "accounts", "overwrite")
put_adls_data("/mnt/project2-container/Silver_Layer/", "parquet",
customers_df, "customers", "overwrite")
put_adls_data("/mnt/project2-container/Silver_Layer/", "parquet",
loan_payments_df, "loan_payments", "overwrite")
put_adls_data("/mnt/project2-container/Silver_Layer/", "parquet", loans_df,
"loans", "overwrite")
put_adls_data("/mnt/project2-container/Silver_Layer/", "parquet",
transactions_df, "transactions", "overwrite")

put_adls_data("/mnt/project2-container/Silver_Layer/", "parquet", joined_df,
"joined_data", "append")

```

The screenshot shows a Jupyter Notebook cell with the following content:

```
1 # Save cleaned and joined DataFrames to Silver Layer in Parquet format
2
3 put_adls_data("/mnt/project2-container/Silver_Layer/", "parquet", accounts_df, "accounts", "overwrite")
4 put_adls_data("/mnt/project2-container/Silver_Layer/", "parquet", customers_df, "customers", "overwrite")
5 put_adls_data("/mnt/project2-container/Silver_Layer/", "parquet", loan_payments_df, "loan_payments", "overwrite")
6 put_adls_data("/mnt/project2-container/Silver_Layer/", "parquet", loans_df, "loans", "overwrite")
7 put_adls_data("/mnt/project2-container/Silver_Layer/", "parquet", transactions_df, "transactions", "overwrite")
8
9 put_adls_data("/mnt/project2-container/Silver_Layer/", "parquet", joined_df, "joined_data", "append")
10
```

Below the code, there is an output section:

Output Terminal Debug console

```
DataFrame[AccountId: int, CustomerId: int, Balance: double, LoanId: int, LoanAmount: double, PaymentId: int, PaymentAme: date, TransactionId: int, TransactionAmount: double, TransactionDate: date]
```

Gold Delta Table Creation Notebook

– Set Catalog

- Sets the SQL context to use the hive_metastore catalog, which is required before creating tables.
- Code:

```
%sql  
use catalog hive_metastore
```



A screenshot of a Jupyter Notebook cell. The cell has a green checkmark icon and the text "12 hours ago (<1s)". The cell content is as follows:

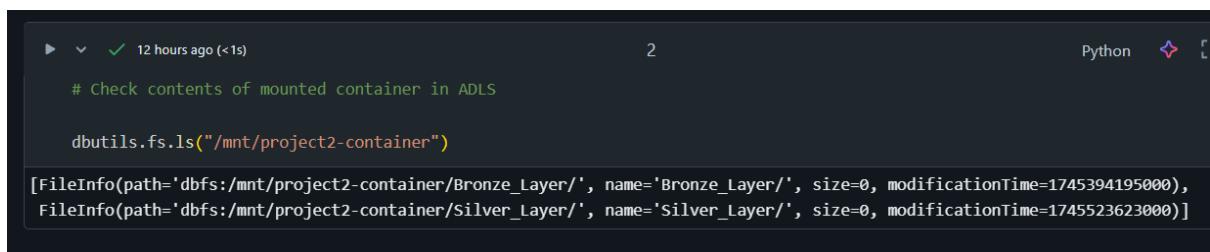
```
%sql  
  
-- Use the Hive Metastore catalog (required for Delta table creation)  
use catalog hive_metastore
```

The cell ends with an "OK" message.

– Filesystem Check

- Lists the contents of the ADLS path /mnt/project2-container to verify that storage is mounted and accessible.
- Code:

```
dbutils.fs.ls("/mnt/project2-container")
```



A screenshot of a Jupyter Notebook cell. The cell has a green checkmark icon and the text "12 hours ago (<1s)". The cell content is as follows:

```
# Check contents of mounted container in ADLS  
dbutils.fs.ls("/mnt/project2-container")
```

The cell output shows two FileInfo objects:

```
[FileInfo(path='dbfs:/mnt/project2-container/Bronze_Layer/', name='Bronze_Layer/', size=0, modificationTime=1745394195000),  
 FileInfo(path='dbfs:/mnt/project2-container/Silver_Layer/', name='Silver_Layer/', size=0, modificationTime=1745523623000)]
```

– Gold Layer Table Creation

- Creates external Delta tables in the Gold Layer for each entity (accounts, customers, loan_payments, loans, and transactions), specifying schema, data types, and file locations.
- Code:

For **accounts** table:

```
%sql
CREATE TABLE IF NOT EXISTS accounts (
    AccountId INT,
    CustomerId INT,
    AccountType STRING,
    Balance DECIMAL(18,2),
    CreatedBy STRING,
    CreatedDate TIMESTAMP,
    UpdatedBy STRING,
    UpdatedDate TIMESTAMP,
    Hashkey BIGINT
)
LOCATION '/mnt/project2-container/Gold_Layer/accounts/'
```

```
%sql

-- Create Gold Layer: Accounts Table
CREATE TABLE IF NOT EXISTS accounts (
    AccountId INT,
    CustomerId INT,
    AccountType STRING,
    Balance DECIMAL(18,2),
    CreatedBy STRING,
    CreatedDate TIMESTAMP,
    UpdatedBy STRING,
    UpdatedDate TIMESTAMP,
    Hashkey BIGINT
)
LOCATION '/mnt/project2-container/Gold_Layer/accounts/'

OK
```

For **customers** table:

```
%sql
CREATE TABLE IF NOT EXISTS customers (
    CustomerId INT,
    FirstName STRING,
    LastName STRING,
    Address STRING,
    City STRING,
    State STRING,
    Zip STRING,
```

```

    CreatedBy STRING,
    CreatedDate TIMESTAMP,
    UpdatedBy STRING,
    UpdatedDate TIMESTAMP,
    Hashkey BIGINT
)
LOCATION "/mnt/project2-container/Gold_Layer/customers/"

```

```

▶ ▾ ✓ 12 hours ago (1s) 4
%sql

-- Create Gold Layer: Customers Table
CREATE TABLE IF NOT EXISTS customers (
    CustomerId INT,
    FirstName STRING,
    LastName STRING,
    Address STRING,
    City STRING,
    State STRING,
    Zip STRING,
    CreatedBy STRING,
    CreatedDate TIMESTAMP,
    UpdatedBy STRING,
    UpdatedDate TIMESTAMP,
    Hashkey BIGINT
)
LOCATION "/mnt/project2-container/Gold_Layer/customers/"

OK

```

For **loan_payments** table:

```

%sql
CREATE TABLE IF NOT EXISTS loan_payments (
    PaymentId INT,
    LoanId INT,
    PaymentDate DATE,
    PaymentAmount DECIMAL(18,2),
    CreatedBy STRING,
    CreatedDate TIMESTAMP,

```

```

    UpdatedBy STRING,
    UpdatedDate TIMESTAMP,
    Hashkey BIGINT
)
LOCATION "/mnt/project2-container/Gold_Layer/loan_payments/"

```

```

▶ ⏴ ✓ 12 hours ago (1s) 5
%sql

-- Create Gold Layer: Loan Payments Table
CREATE TABLE IF NOT EXISTS loan_payments (
    PaymentId INT,
    LoanId INT,
    PaymentDate DATE,
    PaymentAmount DECIMAL(18,2),
    CreatedBy STRING,
    CreatedDate TIMESTAMP,
    UpdatedBy STRING,
    UpdatedDate TIMESTAMP,
    Hashkey BIGINT
)
LOCATION "/mnt/project2-container/Gold_Layer/loan_payments/"

OK

```

For **loans** table:

```

%sql
CREATE TABLE IF NOT EXISTS loans (
    LoanId INT,
    CustomerId INT,
    LoanAmount DECIMAL(10,2),
    InterestRate DECIMAL(5,2),
    LoanTerm INT,
    CreatedBy STRING,
    CreatedDate TIMESTAMP,
    UpdatedBy STRING,
    UpdatedDate TIMESTAMP,
    Hashkey BIGINT
)
LOCATION "/mnt/project2-container/Gold_Layer/loans/"

```

```
▶ ▼ ✓ 12 hours ago (1s) 6
%sql

-- Create Gold Layer: Loans Table
CREATE TABLE IF NOT EXISTS loans (
    LoanId INT,
    CustomerId INT,
    LoanAmount DECIMAL(10,2),
    InterestRate DECIMAL(5,2),
    LoanTerm INT,
    CreatedBy STRING,
    CreatedDate TIMESTAMP,
    UpdatedBy STRING,
    UpdatedDate TIMESTAMP,
    Hashkey BIGINT
)
LOCATION "/mnt/project2-container/Gold_Layer/loans/"

OK
```

For **transactions** table:

```
%sql
CREATE TABLE IF NOT EXISTS transactions (
    TransactionId INT,
    AccountId INT,
    TransactionDate DATE,
    TransactionAmount DECIMAL(10,2),
    TransactionType STRING,
    CreatedBy STRING,
    CreatedDate TIMESTAMP,
    UpdatedBy STRING,
    UpdatedDate TIMESTAMP,
    Hashkey BIGINT
)
LOCATION "/mnt/project2-container/Gold_Layer/transactions/"
```

12 hours ago (1s) 7

```
%sql

-- Create Gold Layer: Transactions Table
CREATE TABLE IF NOT EXISTS transactions (
    TransactionId INT,
    AccountId INT,
    TransactionDate DATE,
    TransactionAmount DECIMAL(10,2),
    TransactionType STRING,
    CreatedBy STRING,
    CreatedDate TIMESTAMP,
    UpdatedBy STRING,
    UpdatedDate TIMESTAMP,
    Hashkey BIGINT
)
LOCATION "/mnt/project2-container/Gold_Layer/transactions/"

OK

This result is stored as _sqldf and can be used in other Python cells.
```

Gold Layer Notebook

– Utility Function Inclusion

- Uses %run to load predefined utility functions (MyUtilityFunctions) to load the user defined function in current Notebook.
- Code:

```
%run "/Workspace/Project2/MyUtilityFunctions"
```

```
06:30 AM (1s) 1 Python :: ::

%run "/Workspace/Project2/MyUtilityFunctions"

[SecretScope(name='adlsstgconnection')]

[SecretMetadata(key='appid'),
 SecretMetadata(key='appsecret'),
 SecretMetadata(key='azure-sqldb-password'),
 SecretMetadata(key='azure-sqldb-username'),
 SecretMetadata(key='onprem-system-password')]

/mnt/project2-container is already mounted.

[FileInfo(path='dbfs:/mnt/project2-container/Bronze_Layer/', name='Bronze_Layer/', size=0, modificationTime=1745394195000),
 FileInfo(path='dbfs:/mnt/project2-container/Gold_Layer/', name='Gold_Layer/', size=0, modificationTime=1745524456000),
 FileInfo(path='dbfs:/mnt/project2-container/Silver_Layer/', name='Silver_Layer/', size=0, modificationTime=1745523623000)]

['accounts.csv', 'customers.csv', 'loan_payments.csv', 'loans.csv', 'transactions.csv']
```

– Read Silver Layer Data

- Loops through all files in the Silver Layer, loads them as DataFrames, and appends to a list.
- Code:

```
all_data_df=[]

for i in range(0,len(files)):
    all_data_df.append(get_adls_data("/mnt/project2-
container/Silver_Layer/", "parquet", files[i]\
.replace(".csv", "")))

print(len(all_data_df))
```

```
▶ ▾ ✓ 12 hours ago (5s) 2

# Initialize list to store DataFrames
# Load Silver Layer Parquet data for all files

all_data_df=[]

for i in range(0,len(files)):
    all_data_df.append(get_adls_data("/mnt/project2-container/Silver_Layer/", "parquet", files[i]\n        .replace(".csv", "")))

print(len(all_data_df))

▶ (5) Spark Jobs
```

– Gold Delta table creation sql command Inclusion

- Uses %run to create external Gold Delta Table DDL (GoldDeltaTableCreation).
 - Code:

```
%run "/Workspace/Project2/GoldDeltaTableCreation"
```

```
▶ ▾ ✓ 07:07 AM (9s) 3 Py

%run "/Workspace/Project2/GoldDeltaTableCreation"

OK
[FileInfo(path='dbfs:/mnt/project2-container/Bronze_Layer/', name='Bronze_Layer/', size=0, modificationTime=1745394195000),
 FileInfo(path='dbfs:/mnt/project2-container/Gold_Layer/', name='Gold_Layer/', size=0, modificationTime=1745524456000),
 FileInfo(path='dbfs:/mnt/project2-container/Silver_Layer/', name='Silver_Layer/', size=0, modificationTime=1745523623000)]
OK
OK
OK
OK
OK
```

– Add Hashkey Column

- Adds a Hashkey to each DataFrame using `crc32(concat(*columns))` to track changes for deduplication and slowly changing dimensions.
 - Code:

```
from pyspark.sql.functions import crc32, concat
```

```
for i in range(0,len(files)):  
    all_data_df[i] = all_data_df[i].withColumn("Hashkey",  
    crc32(concat(*all_data_df[i].columns)))
```

```
▶ ▾ ✓ 12 hours ago (1s) 4
# Generate Hashkey column for SCD (Slowly Changing Dimensions) tracking

from pyspark.sql.functions import crc32, concat

for i in range(0,len(files)):
    all_data_df[i] = all_data_df[i].withColumn("Hashkey", crc32(concat(*all_data_df[i].columns)))
```

– Initialize Delta Tables

- Reads Delta tables from the Gold Layer path using DeltaTable.forPath() and converts them into DataFrames.
- Code:

```
from delta.tables import DeltaTable
```

```
delta_tables = []
delta_table_dfs = []

for i in range(0,len(files)):
    fn = files[i].replace(".csv", "")
    delta_tables.append(DeltaTable.forPath(spark, f"/mnt/project2-
container/Gold_Layer/{fn}/"))
    delta_table_dfs.append(delta_tables[i].toDF())
```

```
▶ ✓ 12 hours ago (< 1s) 5
# Read existing Gold Layer Delta Tables

from delta.tables import DeltaTable

delta_tables = []
delta_table_dfs = []

for i in range(0,len(files)):
    fn = files[i].replace(".csv", "")
    delta_tables.append(DeltaTable.forPath(spark, f"/mnt/project2-container/Gold_Layer/{fn}/"))
    delta_table_dfs.append(delta_tables[i].toDF())
```

– Define Update & Insert Logic

- Stores update and insert column mappings for each table in a dictionary named values.
- Code:

```
from pyspark.sql.functions import col, lit, current_timestamp

values={

    "accounts.csv":
        {
            "set_values":
                {
                    "tgt.AccountId": "src.AccountId",
                    "tgt.CustomerId": "src.CustomerId",
                    "tgt.AccountType": "src.AccountType",
                    "tgt.Balance": "src.Balance",
                    "tgt.UpdatedBy": lit("Gold-Databricks-Update"),
                    "tgt.UpdatedDate": current_timestamp(),
                    "tgt.Hashkey": "src.Hashkey"
                },
            "insert_values":
                {
                    "tgt.AccountId": "src.AccountId",
                    "tgt.CustomerId": "src.CustomerId",
                    "tgt.AccountType": "src.AccountType",
                    "tgt.Balance": "src.Balance",
                    "tgt.CreatedBy": lit("Gold-Databricks"),
                    "tgt.CreatedDate": current_timestamp(),
                    "tgt.UpdatedBy": lit("Gold-Databricks"),
                    "tgt.UpdatedDate": current_timestamp(),
                    "tgt.Hashkey": "src.Hashkey"
                }
        },
    "customers.csv":
        {
            "set_values":
                {
                    "tgt.CustomerId": "src.CustomerId",
                    "tgt.FirstName": "src.FirstName",
                    "tgt.LastName": "src.LastName",
                    "tgt.Address": "src.Address",
                    "tgt.City": "src.City",
                    "tgt.State": "src.State",
                    "tgt.Zip": "src.Zip",
                }
        }
},
```

```

    "tgt.UpdatedBy": lit("Gold-Databricks-Update"),
    "tgt.UpdatedDate": current_timestamp(),
    "tgt.Hashkey": "src.Hashkey"
},
"insert_values" :
{
    "tgt.CustomerId": "src.CustomerId",
    "tgt.FirstName": "src.FirstName",
    "tgt.LastName": "src.LastName",
    "tgt.Address": "src.Address",
    "tgt.City": "src.City",
    "tgt.State": "src.State",
    "tgt.Zip": "src.Zip",
    "tgt.CreatedBy": lit("Gold-Databricks"),
    "tgt.CreatedDate": current_timestamp(),
    "tgt.UpdatedBy": lit("Gold-Databricks"),
    "tgt.UpdatedDate": current_timestamp(),
    "tgt.Hashkey": "src.Hashkey"
},
},
"loan_payments.csv":
{
    "set_values": {
        "tgt.PaymentId": "src.PaymentId",
        "tgt.LoanId": "src.LoanId",
        "tgt.PaymentDate": "src.PaymentDate",
        "tgt.PaymentAmount": "src.PaymentAmount",
        "tgt.UpdatedBy": lit("Gold-Databricks-Update"),
        "tgt.UpdatedDate": current_timestamp(),
        "tgt.Hashkey": "src.Hashkey"
    },
    "insert_values": {
        "tgt.PaymentId": "src.PaymentId",
        "tgt.LoanId": "src.LoanId",
        "tgt.PaymentDate": "src.PaymentDate",
        "tgt.PaymentAmount": "src.PaymentAmount",
        "tgt.CreatedBy": lit("Gold-Databricks"),
        "tgt.CreatedDate": current_timestamp(),
        "tgt.UpdatedBy": lit("Gold-Databricks"),
        "tgt.UpdatedDate": current_timestamp(),
        "tgt.Hashkey": "src.Hashkey"
    }
}
},

```

```

"loans.csv":
{
  "set_values": {
    "tgt.LoanId": "src.LoanId",
    "tgt.CustomerId": "src.CustomerId",
    "tgt.LoanAmount": "src.LoanAmount",
    "tgt.InterestRate": "src.InterestRate",
    "tgt.LoanTerm": "src.LoanTerm",
    "tgt.UpdatedBy": lit("Gold-Databricks-Update"),
    "tgt.UpdatedDate": current_timestamp(),
    "tgt.Hashkey": "src.Hashkey"
  },
  "insert_values": {
    "tgt.LoanId": "src.LoanId",
    "tgt.CustomerId": "src.CustomerId",
    "tgt.LoanAmount": "src.LoanAmount",
    "tgt.InterestRate": "src.InterestRate",
    "tgt.LoanTerm": "src.LoanTerm",
    "tgt.CreatedBy": lit("Gold-Databricks"),
    "tgt.CreatedDate": current_timestamp(),
    "tgt.UpdatedBy": lit("Gold-Databricks"),
    "tgt.UpdatedDate": current_timestamp(),
    "tgt.Hashkey": "src.Hashkey"
  }
},
"transactions.csv":
{
  "set_values": {
    "tgt.TransactionId": "src.TransactionId",
    "tgt.AccountId": "src.AccountId",
    "tgt.TransactionDate": "src.TransactionDate",
    "tgt.TransactionAmount": "src.TransactionAmount",
    "tgt.TransactionType": "src.TransactionType",
    "tgt.UpdatedBy": lit("Gold-Databricks-Update"),
    "tgt.UpdatedDate": current_timestamp(),
    "tgt.Hashkey": "src.Hashkey"
  },
  "insert_values": {
    "tgt.TransactionId": "src.TransactionId",
    "tgt.AccountId": "src.AccountId",
    "tgt.TransactionDate": "src.TransactionDate",
    "tgt.TransactionAmount": "src.TransactionAmount",

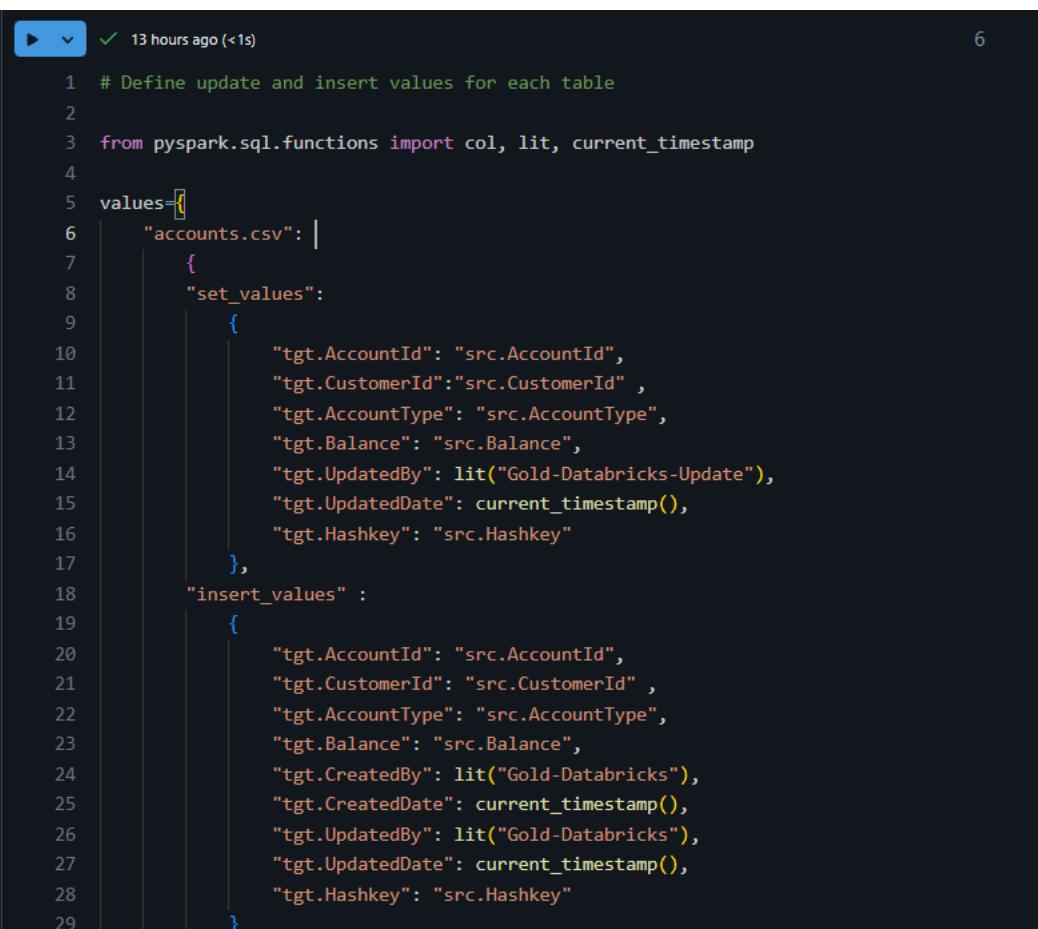
```

```

    "tgt.TransactionType": "src.TransactionType",
    "tgt.CreatedBy": lit("Gold-Databricks"),
    "tgt.CreatedDate": current_timestamp(),
    "tgt.UpdatedBy": lit("Gold-Databricks"),
    "tgt.UpdatedDate": current_timestamp(),
    "tgt.Hashkey": "src.Hashkey"
}
}

}

```



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

- Timestamp:** 13 hours ago (<1s)
- Row Count:** 6
- Code Content:**

```

1 # Define update and insert values for each table
2
3 from pyspark.sql.functions import col, lit, current_timestamp
4
5 values=[{
6     "accounts.csv": {
7         "set_values": {
8             "tgt.AccountId": "src.AccountId",
9             "tgt.CustomerId": "src.CustomerId",
10            "tgt.AccountType": "src.AccountType",
11            "tgt.Balance": "src.Balance",
12            "tgt.UpdatedBy": lit("Gold-Databricks-Update"),
13            "tgt.UpdatedDate": current_timestamp(),
14            "tgt.Hashkey": "src.Hashkey"
15        },
16        "insert_values" :
17        {
18            "tgt.AccountId": "src.AccountId",
19            "tgt.CustomerId": "src.CustomerId",
20            "tgt.AccountType": "src.AccountType",
21            "tgt.Balance": "src.Balance",
22            "tgt.CreatedBy": lit("Gold-Databricks"),
23            "tgt.CreatedDate": current_timestamp(),
24            "tgt.UpdatedBy": lit("Gold-Databricks"),
25            "tgt.UpdatedDate": current_timestamp(),
26            "tgt.Hashkey": "src.Hashkey"
27        }
28    }
29 }

```

```

30 },
31
32 "customers.csv":
33 {
34     "set_values":
35     {
36         "tgt.CustomerId": "src.CustomerId",
37         "tgt.FirstName": "src.FirstName",
38         "tgt.LastName": "src.LastName",
39         "tgt.Address": "src.Address",
40         "tgt.City": "src.City",
41         "tgt.State": "src.State",
42         "tgt.Zip": "src.Zip",
43         "tgt.UpdatedBy": lit("Gold-Databricks-Update"),
44         "tgt.UpdatedDate": current_timestamp(),
45         "tgt.Hashkey": "src.Hashkey"
46     },
47     "insert_values" :
48     {
49         "tgt.CustomerId": "src.CustomerId",
50         "tgt.FirstName": "src.FirstName",
51         "tgt.LastName": "src.LastName",
52         "tgt.Address": "src.Address",
53         "tgt.City": "src.City",
54         "tgt.State": "src.State",
55         "tgt.Zip": "src.Zip",
56         "tgt.CreatedBy": lit("Gold-Databricks"),
57         "tgt.CreatedDate": current_timestamp(),
58         "tgt.UpdatedBy": lit("Gold-Databricks"),
59         "tgt.UpdatedDate": current_timestamp(),
60         "tgt.Hashkey": "src.Hashkey"
61     },
62 },
63
64 "loan_payments.csv":
65 {
66     "set_values": {
67         "tgt.PaymentId": "src.PaymentId",
68         "tgt.LoanId": "src.LoanId",
69         "tgt.PaymentDate": "src.PaymentDate",
70         "tgt.PaymentAmount": "src.PaymentAmount",
71         "tgt.UpdatedBy": lit("Gold-Databricks-Update"),
72         "tgt.UpdatedDate": current_timestamp(),
73         "tgt.Hashkey": "src.Hashkey"
74     },
75     "insert_values": {
76         "tgt.PaymentId": "src.PaymentId",
77         "tgt.LoanId": "src.LoanId",
78         "tgt.PaymentDate": "src.PaymentDate",
79         "tgt.PaymentAmount": "src.PaymentAmount",
80         "tgt.CreatedBy": lit("Gold-Databricks"),
81         "tgt.CreatedDate": current_timestamp(),
82         "tgt.UpdatedBy": lit("Gold-Databricks"),
83         "tgt.UpdatedDate": current_timestamp(),
84         "tgt.Hashkey": "src.Hashkey"
85     }
86 }

```

```

86     },
87
88     "loans.csv":
89     {
90         "set_values": {
91             "tgt.LeanId": "src.LeanId",
92             "tgt.CustomerId": "src.CustomerId",
93             "tgt.LeanAmount": "src.LeanAmount",
94             "tgt.InterestRate": "src.InterestRate",
95             "tgt.LeanTerm": "src.LeanTerm",
96             "tgt.UpdatedBy": lit("Gold-Databricks-Update"),
97             "tgt.UpdatedDate": current_timestamp(),
98             "tgt.Hashkey": "src.Hashkey"
99         },
100        "insert_values": {
101            "tgt.LeanId": "src.LeanId",
102            "tgt.CustomerId": "src.CustomerId",
103            "tgt.LeanAmount": "src.LeanAmount",
104            "tgt.InterestRate": "src.InterestRate",
105            "tgt.LeanTerm": "src.LeanTerm",
106            "tgt.CreatedBy": lit("Gold-Databricks"),
107            "tgt.CreatedDate": current_timestamp(),
108            "tgt.UpdatedBy": lit("Gold-Databricks"),
109            "tgt.UpdatedDate": current_timestamp(),
110            "tgt.Hashkey": "src.Hashkey"
111        }
112    },
113
114    "transactions.csv":
115    {
116        "set_values": {
117            "tgt.TransactionId": "src.TransactionId",
118            "tgt.AccountId": "src.AccountId",
119            "tgt.TransactionDate": "src.TransactionDate",
120            "tgt.TransactionAmount": "src.TransactionAmount",
121            "tgt.TransactionType": "src.TransactionType",
122            "tgt.UpdatedBy": lit("Gold-Databricks-Update"),
123            "tgt.UpdatedDate": current_timestamp(),
124            "tgt.Hashkey": "src.Hashkey"
125        },
126        "insert_values": {
127            "tgt.TransactionId": "src.TransactionId",
128            "tgt.AccountId": "src.AccountId",
129            "tgt.TransactionDate": "src.TransactionDate",
130            "tgt.TransactionAmount": "src.TransactionAmount",
131            "tgt.TransactionType": "src.TransactionType",
132            "tgt.CreatedBy": lit("Gold-Databricks"),
133            "tgt.CreatedDate": current_timestamp(),
134            "tgt.UpdatedBy": lit("Gold-Databricks"),
135            "tgt.UpdatedDate": current_timestamp(),
136            "tgt.Hashkey": "src.Hashkey"
137        }
138    }
139
140 }

```

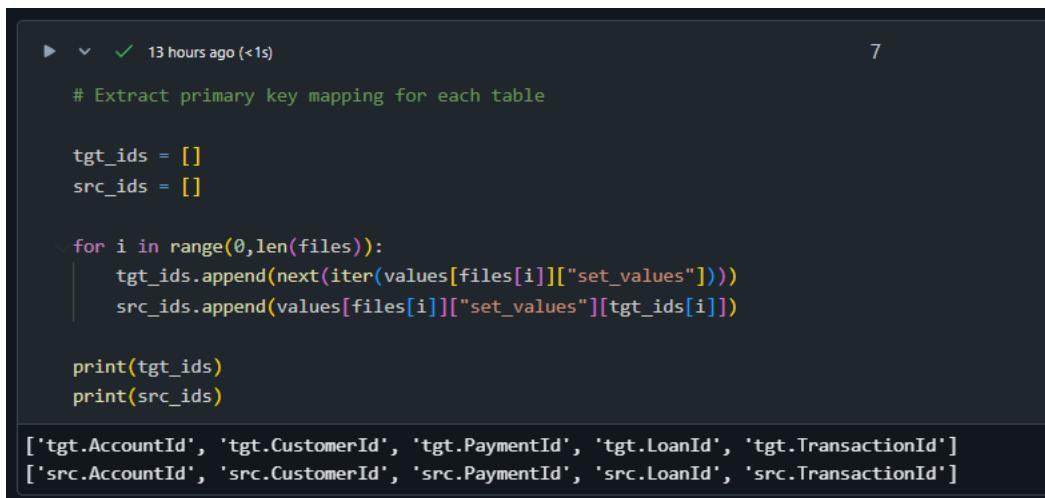
– Generate Matching Columns

- Extracts target and source ID columns (e.g., primary keys) to be used for matching records.
- Code:

```
tgt_ids = []
src_ids = []

for i in range(0,len(files)):
    tgt_ids.append(next(iter(values[files[i]]["set_values"])))
    src_ids.append(values[files[i]]["set_values"][tgt_ids[i]])

print(tgt_ids)
print(src_ids)
```



The screenshot shows a Jupyter Notebook cell with the following content:

```
# Extract primary key mapping for each table

tgt_ids = []
src_ids = []

for i in range(0,len(files)):
    tgt_ids.append(next(iter(values[files[i]]["set_values"])))
    src_ids.append(values[files[i]]["set_values"][tgt_ids[i]])

print(tgt_ids)
print(src_ids)
```

Output:

```
['tgt.AccountId', 'tgt.CustomerId', 'tgt.PaymentId', 'tgt.LoanId', 'tgt.TransactionId']
['src.AccountId', 'src.CustomerId', 'src.PaymentId', 'src.LoanId', 'src.TransactionId']
```

– Anti Join for New Records

- Performs anti-joins between Silver (source) and Gold (target) on primary_key and Hashkey to get only new or changed records.
- Code:

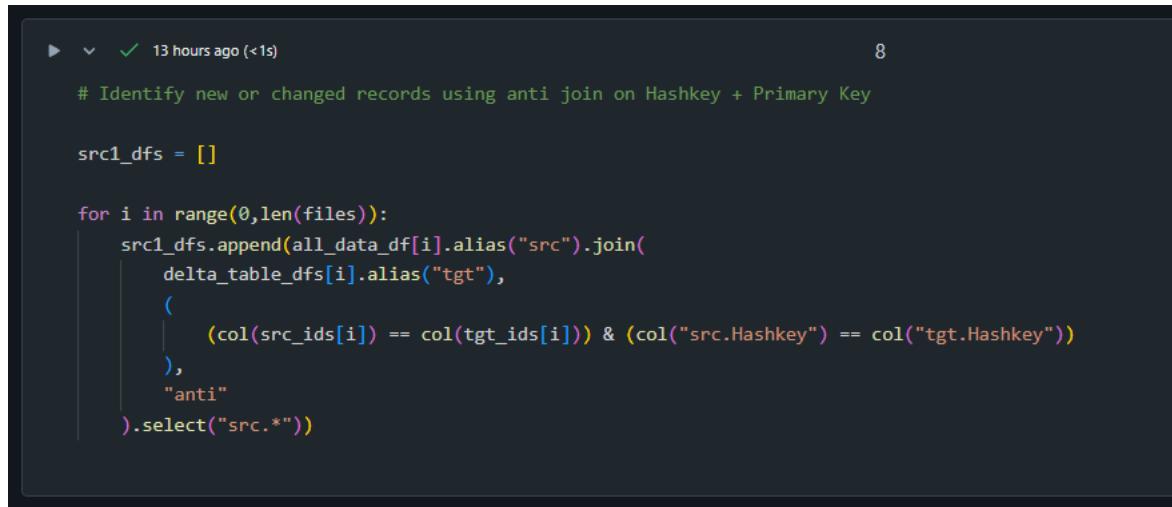
```
src1_dfs = []

for i in range(0,len(files)):
    src1_dfs.append(all_data_df[i].alias("src").join(
        delta_table_dfs[i].alias("tgt"),
        (
            (col(src_ids[i]) == col(tgt_ids[i])) & (col("src.Hashkey") ==
            col("tgt.Hashkey"))
```

```

),
"anti"
).select("src.*"))

```



```

▶ 13 hours ago (<1s) 8
# Identify new or changed records using anti join on Hashkey + Primary Key

src1_dfs = []

for i in range(0,len(files)):
    src1_dfs.append(all_data_df[i].alias("src").join(
        delta_table_dfs[i].alias("tgt"),
        (
            (col(src_ids[i]) == col(tgt_ids[i])) & (col("src.Hashkey") == col("tgt.Hashkey"))
        ),
        "anti"
    ).select("src.*"))

```

– Merge into Delta Tables

- Executes a merge for each table using:
 - whenMatchedUpdate for updated records
 - whenNotMatchedInsert for new records
- Code:

```

for i in range(0,len(files)):
    delta_tables[i].alias("tgt").merge(src1_dfs[i].alias("src"),((col(tgt_ids[i]) ==
col(src_ids[i]))))\
        .whenMatchedUpdate(
            set = values[files[i]]["set_values"]
        )\
        .whenNotMatchedInsert(
            values = values[files[i]]["insert_values"]
        ).execute()

```

▶ ✓ 13 hours ago (17s) 9

```
# Perform MERGE to update or insert records into Gold Layer Delta tables

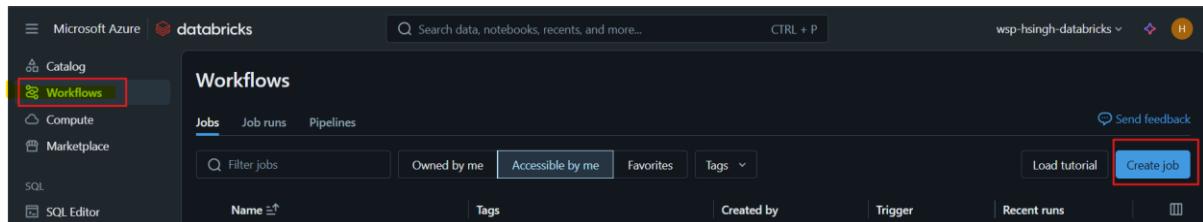
for i in range(0,len(files)):
    delta_tables[i].alias("tgt").merge(src1_dfs[i].alias("src"),((col(tgt_ids[i]) == col(src_ids[i]))))\
        .whenMatchedUpdate(
            set = values[files[i]]["set_values"]
        )\
        .whenNotMatchedInsert(
            values = values[files[i]]["insert_values"]
        ).execute()

▶ (35) Spark Jobs
```

Databricks Workflow Automation

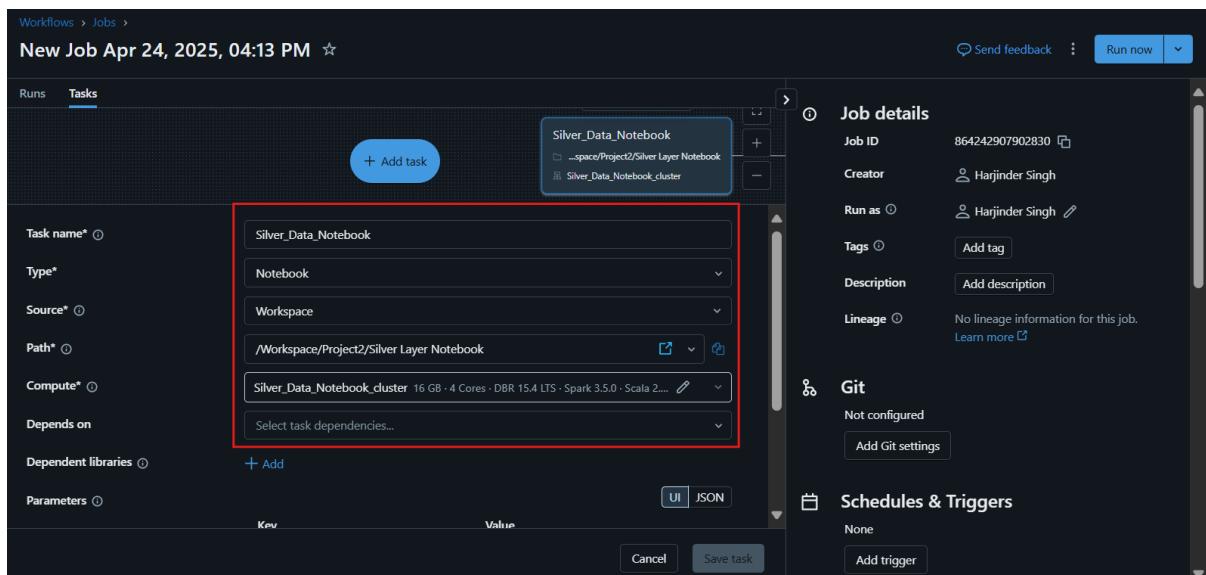
Here, I have used the **scheduled trigger** to automate the execution of workflow job which contains 2 tasks to execute Silver and Gold Layer Notebooks in sequence **every morning at 4AM**.

Step 1: Go to Azure Databricks -> click on **Workflows** -> Click on **Create Job**.



The screenshot shows the Databricks Workflows interface. On the left sidebar, 'Workflows' is selected and highlighted with a red box. At the top right, there is a 'Create job' button, which is also highlighted with a red box.

Step 2: Enter the basic details such as task name, source, path etc. -> click on computed to create new job cluster.



The screenshot shows the 'New Job' creation dialog in Databricks. The 'Tasks' tab is selected. In the 'Task name*' field, 'Silver_Data_Notebook' is entered. Below it, the 'Type*' dropdown is set to 'Notebook'. The 'Source*' dropdown is set to 'Workspace', and the 'Path*' dropdown shows '/Workspace/Project2/Silver Layer Notebook'. The 'Compute*' dropdown shows 'Silver_Data_Notebook_cluster'. A red box highlights the 'Compute*' dropdown and its associated settings. To the right, the 'Job details' and 'Schedules & Triggers' sections are visible.

Step 3: In the compute, Click on Add New Job Cluster option.

Workflows > Jobs >

New Job Apr 24, 2025, 04:13 PM ☆

Runs Tasks

Jobs Compute

Serverless

Silver_Data_Notebook_cluster
16 GB · 4 Cores · DBR 15.4 LTS · Spark 3.5.0 · Scala 2.12

Add new job cluster

Task name* ⓘ

Type* ⓘ

Source* ⓘ

Path* ⓘ

Compute* ⓘ

Depends on

Dependent libraries ⓘ

Parameters ⓘ

UI JSON

Cancel Save task

Step 4: Select the single Node and basic node type as shown below.

Silver_Data_Notebook_cluster Simple form: OFF

Cluster name

Silver_Data_Notebook_cluster

Policy ⓘ

Unrestricted

Single node

Multi node

Access mode ⓘ

Dedicated (formerly: Single user)

Performance

Databricks runtime version ⓘ

Runtime: 15.4 LTS (Scala 2.12, Spark 3.5.0)

Use Photon Acceleration ⓘ

Node type ⓘ

Standard_D4ds_v5 16 GB Memory, 4 Cores

Summary

1 Driver 16 GB Memory, 4 Cores

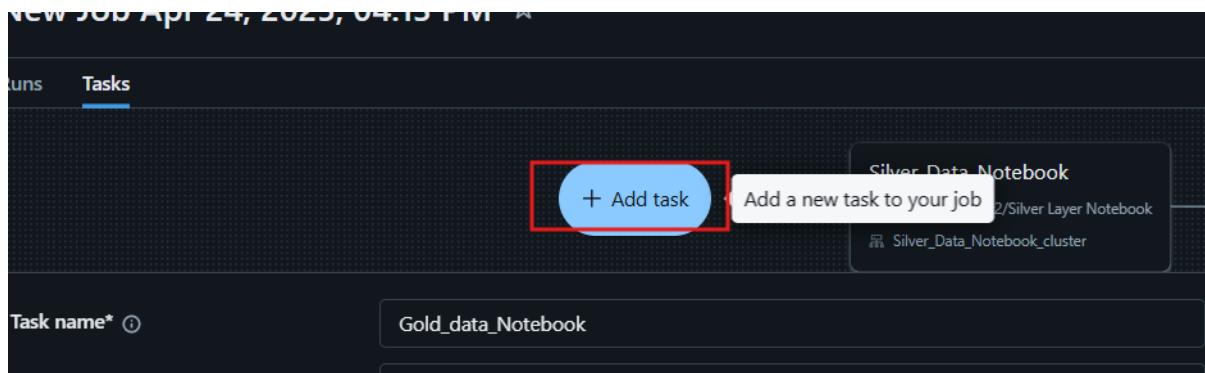
Runtime 15.4.x-scala2.12

Unity Catalog Standard_D4ds_v5

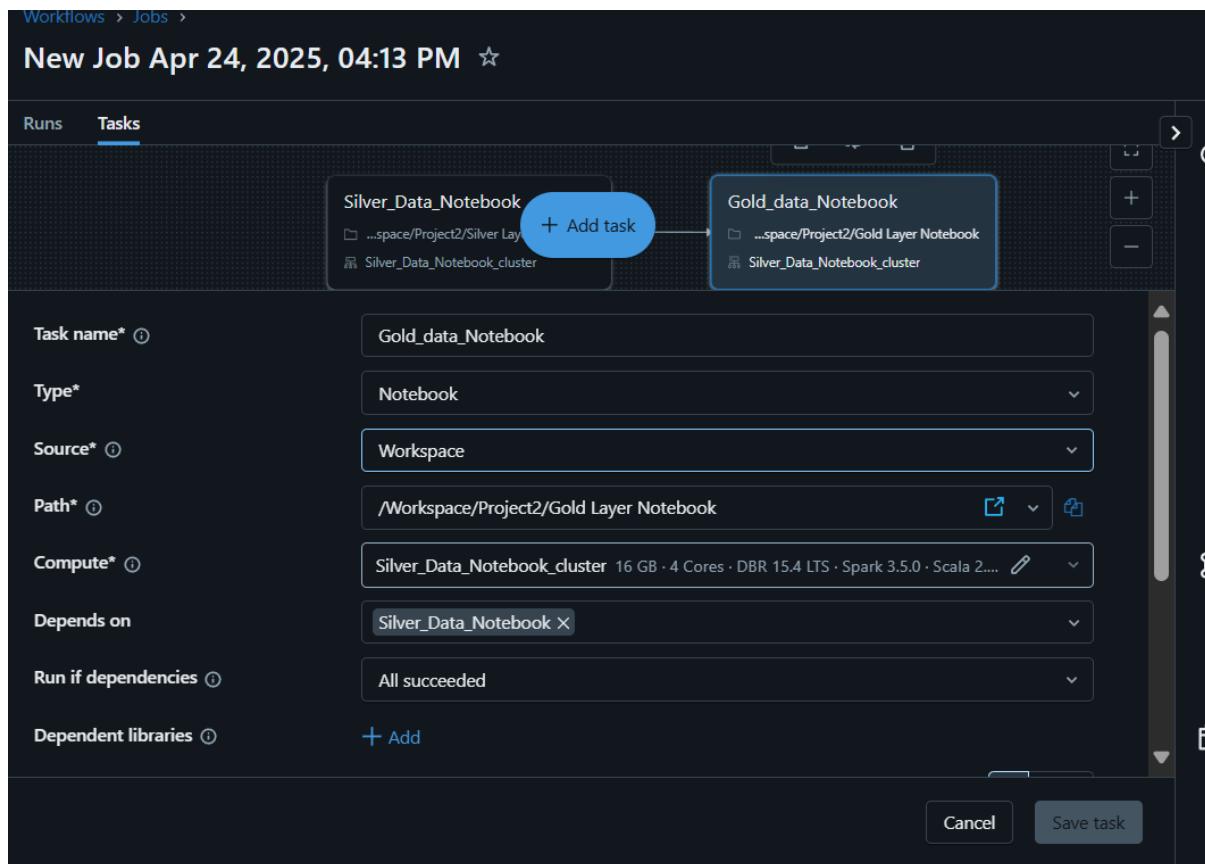
1 DBU/h

Cancel Confirm

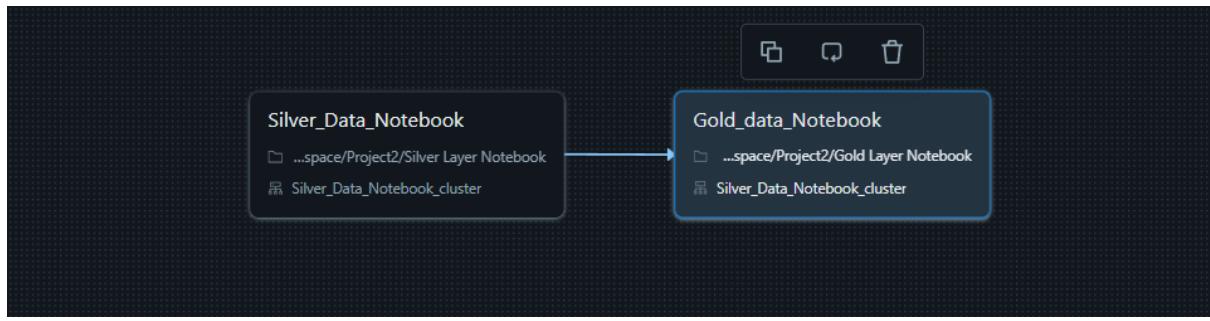
Step 5: Next, Click on Add task to add Glod Layer Notebook as dependent task to Silver Layer Notebook.



Step 6: Enter basic details such as task name, type etc., and select the previously created compute job cluster. -> click on save task.



Step 7: Review the workflow tasks.



Step 8: To add trigger to automate the workflow execution, click on trigger.

No task selected

Choose a task from the graph to edit its properties

Lineage (No lineage information for this job. Learn more)

Git (Not configured. Add Git settings)

Schedules & Triggers (None. Add trigger)

Compute (Silver_Data_Notebook_cluster. Single node: Standard_D4ds_v5 - DBR: 15.4 LTS (includes Apache Spark 3.5.0, Scala 2.12). Configure Swap)

Job parameters

Step 9: select the trigger type as Scheduled.

A

Schedules & Triggers

Trigger Status

Active

Paused

Trigger type

Scheduled

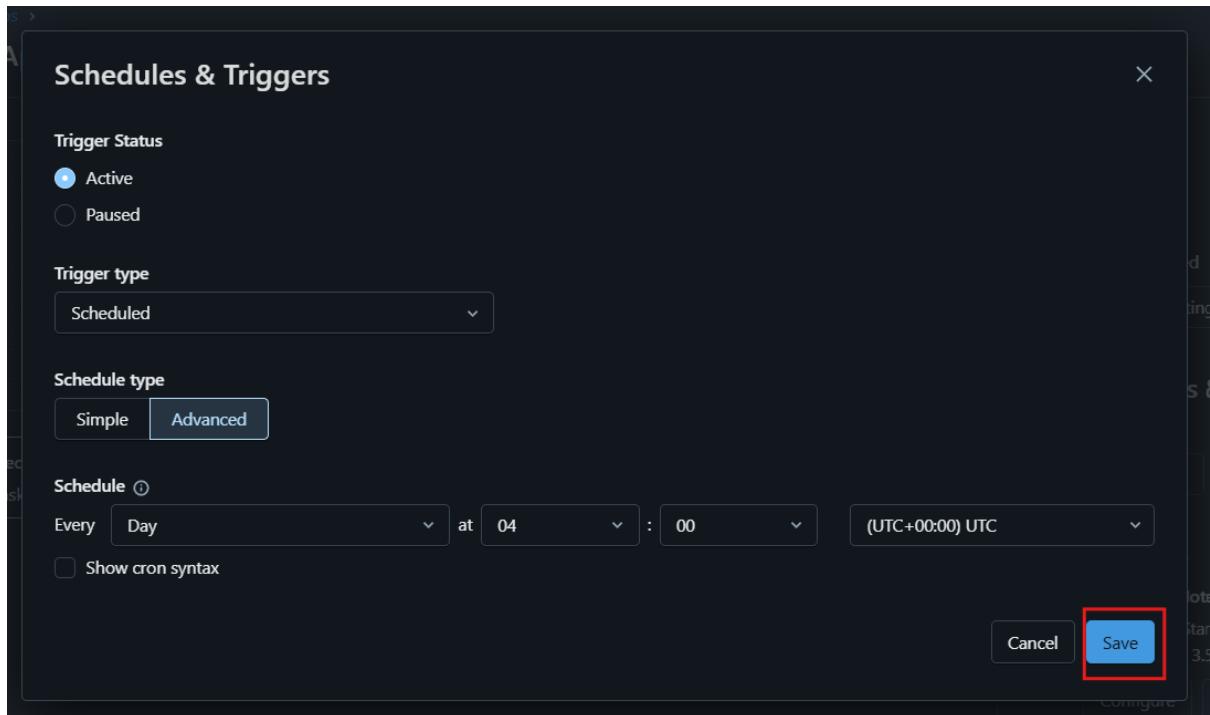
- Scheduled
- File arrival
- Continuous ⓘ

Periodic ⓘ

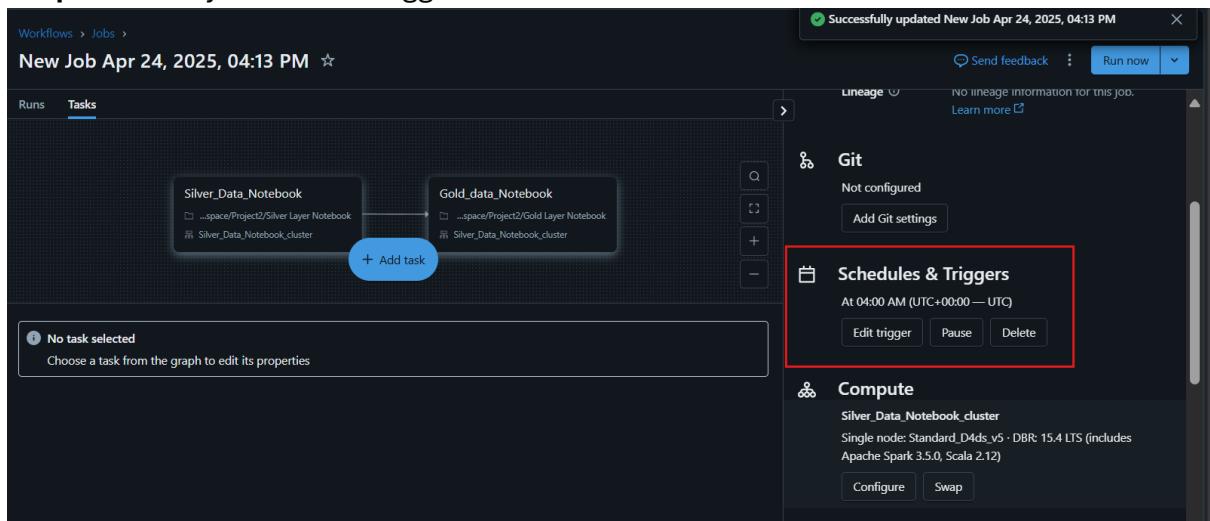
Every 1 Day

Cancel Save

Step 10: select Scheduled type as Advanced and Enter the Every day and time (this will run every day in the morning at 4AM) -> click on Save.



Step 11: Finally review the trigger.



Output Review

✓ First Run Outputs

- Result of Silver Layer Notebook
 - Access all 5 created parquet files from Silver_Layer folder in ADLS Gen2 and Displays the **five DataFrames** (one for each table like accounts, customers, etc.) for visual inspection in the Databricks notebook.

```
▶ Just now (4s) 12
all_data_df=[]

for i in range(0,len(files)):
    all_data_df.append(get_adls_data("/mnt/project2-container/Silver_Layer/", "parquet", files[i]\n        .replace(".csv", "")))

▶ (5) Spark Jobs
```



```
▶ 1 minute ago (5s)
display(all_data_df[0])
display(all_data_df[1])
display(all_data_df[2])
display(all_data_df[3])
display(all_data_df[4])

▶ (5) Spark Jobs
```

For accounts:

	Accountid	CustomerId	AccountType	Balance
1	54	42	Checking	5500.5
2	88	1	Checking	8900
3	82	2	Checking	8300.5
4	92	44	Checking	9300
5	21	53	Savings	300.25
6	29	58	Savings	75.25
7	35	62	Savings	175.75
8	38	15	Checking	3900.5
9	72	17	Checking	7300
10	33	85	Savings	150.25
11	45	68	Savings	300.25
12	86	21	Checking	8700.5
13	51	72	Savings	375.75
14	84	40	Checking	8500
15	5	56	Savings	500

For customers:

Table +

	i^2_3 CustomerId	A ^B C FirstName	A ^B C LastName	A ^B C Address	A ^B C City	A ^B C State	A ^B C Zip
1	45	Christopher	Ward	4444 Maple Ave	Keswick	ON	L4P0A1
2	68	Charlotte	Griffin	6767 Poplar St	Victoria Harbour	ON	L0K0A1
3	86	Olivia	Gibson	8585 Elm St	New Liskeard	ON	P0J0A1
4	80	Emily	Jordan	8888 Airport rd	North Bay	ON	P1B0A1
5	37	Alexander	Bell	3636 Redwood Dr	Stratford	ON	N5A0A1
6	83	David	Fisher	8282 Ash Blvd	Verner	ON	P0H0A1
7	73	Andrew	Hamilton	7272 Maple Ave	Gravenhurst	ON	P1P0A1
8	34	Olivia	Reed	3333 Birch Blvd	Orillia	ON	L3V0A1
9	3	Michael	Johnson	789 Oak Dr	Montreal	QC	H1A1A1
10	9	William	Lopez	808 Redwood Dr	Victoria	BC	V8W0A1
11	28	Emily	Edwards	2727 Beech Dr	Brantford	ON	N3T0A1
12	43	Joseph	Cox	4242 Cedar Ln	Aurora	ON	L4G0A1
13	50	Evelyn	Barnes	4949 Fir St	Sutton	ON	L0E0A1
14	69	Joseph	Diaz	6868 Ash Blvd	Port McNicoll	ON	L0K0A1
15	56	Elizabeth	Long	5555 Beech Dr	East Gwillimbury	ON	L9N0A1

For loan_payments:

Table +

	i^2_3 PaymentId	i^2_3 LoanId	PaymentDate	1.2 PaymentAmount
1	4	89	2024-01-04	250
2	23	54	2024-01-23	1200
3	92	13	2024-04-01	4650
4	6	34	2024-01-06	350
5	21	32	2024-01-21	1100
6	39	30	2024-02-08	2000
7	76	37	2024-03-16	3850
8	1	45	2024-01-01	100
9	45	96	2024-02-14	2300
10	34	75	2024-02-03	1750
11	65	16	2024-03-05	3300
12	96	57	2024-04-05	4850
13	14	55	2024-01-14	750
14	3	67	2024-01-03	200
15	36	97	2024-02-05	1850

For loans:

Table +

	i^2_3 LoanId	i^2_3 CustomerId	1.2 LoanAmount	1.2 InterestRate	i^2_3 LoanTerm
1	54	42	30000.5	4	48
2	29	58	32500.25	5	36
3	72	17	20000	3	24
4	88	1	27500	3	24
5	78	4	27500.5	4	48
6	18	5	27500.5	4.5	48
7	70	33	37500.5	4	48
8	17	99	22500.25	5.5	36
9	46	24	17500.5	4	48
10	8	67	27500	3	24
11	79	55	32500.75	6.5	60
12	23	88	15000.75	6.5	60
13	53	86	15000.25	5	36
14	59	75	32500.75	6	60
15	1	45	10000.5	5.5	36

For transactions:

Table +

	i^2_3 TransactionId	i^2_3 AccountId	📅 TransactionDate	1.2 TransactionAmount	A^B_C TransactionType
1	83	82	2024-03-23	150	Deposit
2	81	70	2024-03-21	100.5	Deposit
3	90	38	2024-03-30	375.25	Withdrawal
4	23	88	2024-01-23	150	Deposit
5	93	79	2024-04-02	150	Deposit
6	69	59	2024-03-09	325	Deposit
7	61	52	2024-03-01	100.5	Deposit
8	87	93	2024-03-27	225.5	Deposit
9	17	99	2024-01-17	225.5	Deposit
10	71	73	2024-03-11	100.5	Deposit
11	91	77	2024-03-31	100.5	Deposit
12	42	36	2024-02-11	200.75	Withdrawal
13	38	15	2024-02-07	275.75	Withdrawal
14	89	54	2024-03-29	325	Deposit
15	15	47	2024-01-15	250	Deposit

- Result of Gold Layer Notebook
 - Displays the **five Delta tables** (one for each table like accounts, customers, etc.) from Gold Layer Location path in ADLS Gen2.

2 minutes ago (7s) 10

```
%sql
select * from accounts@v1
```

(1) Spark Jobs

_sqldf: pyspark.sql.dataframe.DataFrame = [Accountid: integer, CustomerId: integer ... 7 more fields]

Table +

	Accountid	CustomerId	AccountType	Balance	CreatedBy	CreatedDate	UpdatedBy	Up
11	45	68	Savings	300.25	Gold-Databricks	2025-04-24T19:57:13.858+00:00	Gold-Databricks	2025-04-24T19:57:13.858+00:00
12	86	21	Checking	8700.50	Gold-Databricks	2025-04-24T19:57:13.858+00:00	Gold-Databricks	2025-04-24T19:57:13.858+00:00
13	51	72	Savings	375.75	Gold-Databricks	2025-04-24T19:57:13.858+00:00	Gold-Databricks	2025-04-24T19:57:13.858+00:00
14	84	40	Checking	8500.00	Gold-Databricks	2025-04-24T19:57:13.858+00:00	Gold-Databricks	2025-04-24T19:57:13.858+00:00
15	5	56	Savings	500.00	Gold-Databricks	2025-04-24T19:57:13.858+00:00	Gold-Databricks	2025-04-24T19:57:13.858+00:00
16	15	47	Savings	700.75	Gold-Databricks	2025-04-24T19:57:13.858+00:00	Gold-Databricks	2025-04-24T19:57:13.858+00:00
17	98	49	Checking	9900.50	Gold-Databricks	2025-04-24T19:57:13.858+00:00	Gold-Databricks	2025-04-24T19:57:13.858+00:00
18								

↓ 100 rows | 6.92s runtime Refreshed 1 minute ago

This result is stored as _sqldf and can be used in other Python and SQL cells.

1 minute ago (1s) 11

```
%sql
select * from customers@v1
```

(1) Spark Jobs

_sqldf: pyspark.sql.dataframe.DataFrame = [CustomerId: integer, FirstName: string ... 10 more fields]

Table +

	CustomerId	FirstName	LastName	Address	City	State	Zip	CreatedBy	CreatedD
1	45	Christopher	Ward	4444 Maple Ave	Keswick	ON	L4P0A1	Gold-Databricks	2025-04-24T19:57:13.858+00:00
2	68	Charlotte	Griffin	6767 Poplar St	Victoria Harbour	ON	L0K0A1	Gold-Databricks	2025-04-24T19:57:13.858+00:00
3	86	Olivia	Gibson	8585 Elm St	New Liskeard	ON	P0J0A1	Gold-Databricks	2025-04-24T19:57:13.858+00:00
4	37	Alexander	Bell	3636 Redwood Dr	Stratford	ON	N5A0A1	Gold-Databricks	2025-04-24T19:57:13.858+00:00
5	83	David	Fisher	8282 Ash Blvd	Verner	ON	P0H0A1	Gold-Databricks	2025-04-24T19:57:13.858+00:00
6	73	Andrew	Hamilton	7272 Maple Ave	Gravenhurst	ON	P1P0A1	Gold-Databricks	2025-04-24T19:57:13.858+00:00
7	34	Olivia	Reed	3333 Birch Blvd	Orillia	ON	L3V0A1	Gold-Databricks	2025-04-24T19:57:13.858+00:00
8									

↓ 87 rows | 0.69s runtime Refreshed 1 minute ago

This result is stored as _sqldf and can be used in other Python and SQL cells.

2 minutes ago (1s) 12 SQL

```
%sql
select * from loan_payments@v1
```

(1) Spark Jobs

_sqldf: pyspark.sql.dataframe.DataFrame = [PaymentId: integer, LoanId: integer ... 7 more fields]

Table +

	PaymentId	LoanId	PaymentDate	PaymentAmount	CreatedBy	CreatedDate	UpdatedBy
1	4	89	2024-01-04	250.00	Gold-Databricks	2025-04-24T19:57:24.006+00:00	Gold-Databricks
2	23	54	2024-01-23	1200.00	Gold-Databricks	2025-04-24T19:57:24.006+00:00	Gold-Databricks
3	92	13	2024-04-01	4650.00	Gold-Databricks	2025-04-24T19:57:24.006+00:00	Gold-Databricks
4	6	34	2024-01-06	350.00	Gold-Databricks	2025-04-24T19:57:24.006+00:00	Gold-Databricks
5	21	32	2024-01-21	1100.00	Gold-Databricks	2025-04-24T19:57:24.006+00:00	Gold-Databricks
6	39	30	2024-02-08	2000.00	Gold-Databricks	2025-04-24T19:57:24.006+00:00	Gold-Databricks
7	76	37	2024-03-16	3850.00	Gold-Databricks	2025-04-24T19:57:24.006+00:00	Gold-Databricks
8	1	45	2024-01-01	100.00	Gold-Databricks	2025-04-24T19:57:24.006+00:00	Gold-Databricks

101 rows | 0.73s runtime Refreshed 2 minutes ago

This result is stored as _sqldf and can be used in other Python and SQL cells.

3 minutes ago (1s) 13 SQL

```
%sql
select * from loans@v1
```

(1) Spark Jobs

_sqldf: pyspark.sql.dataframe.DataFrame = [LoanId: integer, CustomerId: integer ... 8 more fields]

Table +

	LoanId	CustomerId	LoanAmount	InterestRate	LoanTerm	CreatedBy	CreatedDate	UpdatedBy
1	54	42	30000.50	4.00	48	Gold-Databricks	2025-04-24T19:57:26.575+00:00	Gold-Databricks
2	29	58	32500.25	5.00	36	Gold-Databricks	2025-04-24T19:57:26.575+00:00	Gold-Databricks
3	72	17	20000.00	3.00	24	Gold-Databricks	2025-04-24T19:57:26.575+00:00	Gold-Databricks
4	88	1	27500.00	3.00	24	Gold-Databricks	2025-04-24T19:57:26.575+00:00	Gold-Databricks
5	78	4	27500.50	4.00	48	Gold-Databricks	2025-04-24T19:57:26.575+00:00	Gold-Databricks
6	18	5	27500.50	4.50	48	Gold-Databricks	2025-04-24T19:57:26.575+00:00	Gold-Databricks

100 rows | 0.83s runtime Refreshed 3 minutes ago

This result is stored as _sqldf and can be used in other Python and SQL cells.

3 minutes ago (1s) 14 SQL

```
%sql
select * from transactions@v1
```

(1) Spark Jobs

_sqldf: pyspark.sql.dataframe.DataFrame = [TransactionId: integer, AccountId: integer ... 8 more fields]

Table +

	TransactionId	AccountId	TransactionDate	TransactionAmount	TransactionType	CreatedBy	CreatedDate
1		83	2024-03-23	150.00	Deposit	Gold-Databricks	2025-04-24T19:57:26.575+00:00
2		81	2024-03-21	100.50	Deposit	Gold-Databricks	2025-04-24T19:57:26.575+00:00
3		90	2024-03-30	375.25	Withdrawal	Gold-Databricks	2025-04-24T19:57:26.575+00:00
4		23	2024-01-23	150.00	Deposit	Gold-Databricks	2025-04-24T19:57:26.575+00:00
5		93	2024-04-02	150.00	Deposit	Gold-Databricks	2025-04-24T19:57:26.575+00:00
6		69	2024-03-09	325.00	Deposit	Gold-Databricks	2025-04-24T19:57:26.575+00:00
7		61	2024-03-01	100.50	Deposit	Gold-Databricks	2025-04-24T19:57:26.575+00:00

100 rows | 1.41s runtime Refreshed 4 minutes ago

This result is stored as _sqldf and can be used in other Python and SQL cells.

✓ **Second Run Outputs**

- Suppose Changes made in files (For Next Day data): **customers** and **transaction**.

Customers data before changes at Bronze:

76	75	Joshua	Sullivan	7474 Pine Rd	Bracebridge	ON	P1L0A1
77	76	Evelyn	Wallace	7575 Birch Blvd	Huntsville	ON	P1H0A1
78	77	Daniel	Woods	7676 Spruce Ln	Burks Falls	ON	P0A0A1
79	78	Abigail	Cole	7777 Fir St	Sundridge	ON	P0A0A1
80	79	James	West	7878 Redwood Dr	South River	ON	P0A0A1
81	80	Emily	Jordan	8888 Airport rd	North Bay	ON	P1B0A1
82	81	Michael	Owens	8080 Willow Rd	Mattawa	ON	P0H0A1
83	82	Elizabeth	Reynolds	8181 Poplar St	Sturgeon Falls	ON	P2B0A1
84	83	David	Fisher	8282 Ash Blvd	Verner	ON	P0H0A1
85	84	Sophia	Ellis	8383 Beech Dr	Field	ON	P0H0A1
86	85	John	Harrison	8484 Cedar Ln	Temagami	ON	P0H0A1
87	86	Olivia	Gibson	8585 Elm St	New Liskeard	ON	P0J0A1
88	87	William	McDonald	8686 Maple Ave	Haileybury	ON	P0J0A1
89							

Customers data after changes at Bronze:

78	77	Daniel	Woods	7676 Spruce Ln	Burks Falls	ON	P0A0A1
79	78	Abigail	Cole	7777 Fir St	Sundridge	ON	P0A0A1
80	79	James	West	7878 Redwood Dr	South River	ON	P0A0A1
81	80	Emily	Jordan	8888 Airport rd	North Bay	ON	P1B0A1
82	81	Michael	Owens	8080 Willow Rd	Mattawa	ON	P0H0A1
83	82	Elizabeth	Reynolds	8181 Poplar St	Sturgeon Falls	ON	P2B0A1
84	83	David	Fisher	8282 Ash Blvd	Verner	ON	P0H0A1
85	84	Sophia	Ellis	8383 Beech Dr	Field	ON	P0H0A1
86	85	John	Harrison	8484 Cedar Ln	Temagami	ON	P0H0A1
87	86	Olivia	Gibson	8585 Elm St	New Liskeard	ON	P0J0A1
88	87	William	McDonald	8686 Maple Ave	Haileybury	ON	P0J0A1
89	88	Harjinder	Singh	86 Sunny Ave	Brampton	ON	P0J0A1
90							

transactions data before changes at Bronze:

95	94	39	03-04-2024	300.25	Withdrawal
96	95	60	04-04-2024	250	Deposit
97	96	48	05-04-2024	175	Withdrawal
98	97	90	06-04-2024	225.5	Deposit
99	98	49	07-04-2024	275.75	Withdrawal
100	99	80	08-04-2024	325	Deposit
101	100	50	09-04-2024	510.25	Deposit
102					

transactions data after changes at Bronze:

97	96	48	05-04-2024	175	Withdrawal
98	97	90	06-04-2024	225.5	Deposit
99	98	49	07-04-2024	275.75	Withdrawal
100	99	80	08-04-2024	325	Deposit
101	100	50	09-04-2024	550.25	Deposit
102	101	50	09-04-2024	400.25	Withdrawal
103					

- Result of Gold Layer Notebook After Execution.

For customers:

```
%sql
select * from customers
```

88 rows | 1.12s runtime Refreshed now

This result is stored as `_sql1df` and can be used in other Python and SQL cells.

```
1 %sql
2
3 select * from customers where CustomerId in [80,88]
```

CustomerID	FirstName	LastName	Address	City	State	Zip	CreatedBy	CreatedDate	UpdatedBy
80	Emily	Jordan	8888 Airport rd	North Bay	ON	P1B0A1	Gold-Databricks	2025-04-24T19:57:21.635+00:00	Gold-Databricks
88	Harjinder	Singh	86 Sunny Ave	Brampton	ON	P0J0A1	Gold-Databricks	2025-04-24T20:21:44.421+00:00	Gold-Databricks

For customers:

Just now (1s) 14 SQL ⚡ ⌂ ⋮

```
%sql  
  
select * from transactions  
▶ (2) Spark Jobs  
▶ _sqldf: pyspark.sql.dataframe.DataFrame = [TransactionId: integer, AccountId: integer ... 8 more fields]
```

Table +

	TransactionId	AccountId	TransactionDate	TransactionAmount	TransactionType	CreatedBy	CreatedDate
10	71	73	2024-03-11	100.50	Deposit	Gold-Databricks	2025-04-24T19:5
11	91	77	2024-03-31	100.50	Deposit	Gold-Databricks	2025-04-24T19:5
12	42	36	2024-02-11	200.75	Withdrawal	Gold-Databricks	2025-04-24T19:5
13	38	15	2024-02-07	275.75	Withdrawal	Gold-Databricks	2025-04-24T19:5
14	89	54	2024-03-29	325.00	Deposit	Gold-Databricks	2025-04-24T19:5
15	15	47	2024-01-15	250.00	Deposit	Gold-Databricks	2025-04-24T19:5
16	57	97	2024-02-26	225.50	Deposit	Gold-Databricks	2025-04-24T19:5

101 rows | 1.48s runtime Refreshed now

This result is stored as `_sqldf` and can be used in other Python and SQL cells.

Just now (<1s) 14 SQL ⚡ ⌂ ⋮ ⏵

```
1 %sql  
2  
3 select * from transactions where TransactionId in [100, 101]
```

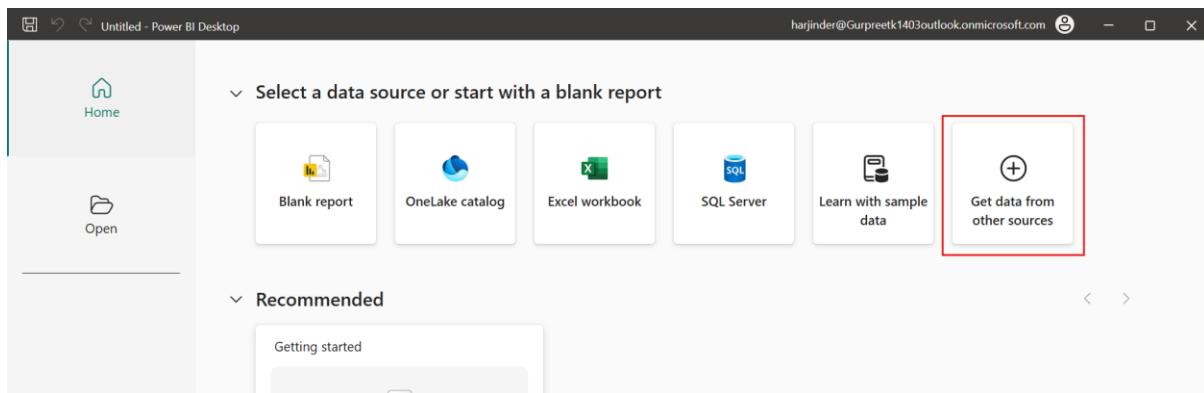
Table +

tionId	AccountId	TransactionDate	TransactionAmount	TransactionType	CreatedBy	CreatedDate	UpdatedBy	UpdatedDate	
1	101	50	2024-04-09	400.25	Withdrawal	Gold-Databricks	2025-04-24T20:22:00.655+00... 2025-04-24T20:22:00.655+00...	Gold-Databricks	2025-04-24T20:22:00.655+00... 2025-04-24T20:22:00.655+00...
2	100	50	2024-04-09	550.25	Deposit	Gold-Databricks	2025-04-24T19:57:28.613+00... 2025-04-24T19:57:28.613+00...	Gold-Databricks-Update	2025-04-24T19:57:28.613+00... 2025-04-24T19:57:28.613+00...

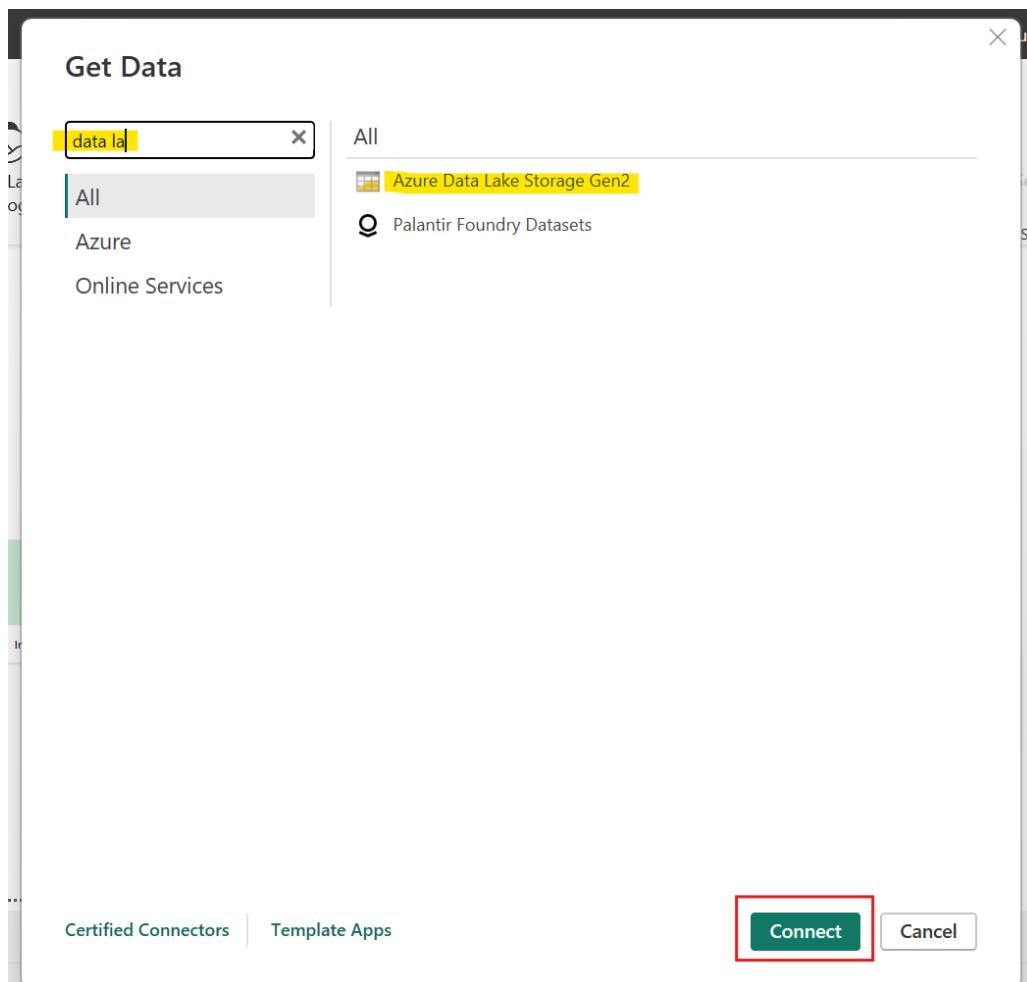
Power BI Report

Step 1: Install and open the Power BI desktop applications -> login with your current working azure portal credentials. (make sure the same credentials are used to login in Fabric account in order to access workspace).

Next, select Get Data From other source as data Source.



- Search and select Azure Data Lake Storage Gen2 option -> click on Connect.



Step 2: Next, enter the URL of Azure Data Lake storage Gen2 account with folder path (find the path in Endpoint tab of Setting in ADLS in DataLake storage section)-> select File System View as Data View. -> click on Ok.

Home > hsinghadls

hsinghadls | Endpoints ⚡ ...

Storage account

Search Refresh Give feedback

Queue service

Resource ID: /subscriptions/ca6fef37-a187-4dcb-b638-ab3267c398e1/resourceGroups/hsingh-rg/providers/Microsoft.Storage/storageAccounts... [Copy](#)

Queue service: https://hsinghadls.queue.core.windows.net/ [Copy](#)

Table service

Resource ID: /subscriptions/ca6fef37-a187-4dcb-b638-ab3267c398e1/resourceGroups/hsingh-rg/providers/Microsoft.Storage/storageAccounts... [Copy](#)

Table service: https://hsinghadls.table.core.windows.net/ [Copy](#)

Data Lake Storage

Resource ID: /subscriptions/ca6fef37-a187-4dcb-b638-ab3267c398e1/resourceGroups/hsingh-rg/providers/Microsoft.Storage/storageAccounts... [Copy](#)

Data Lake Storage: https://hsinghadls.dfs.core.windows.net/ [Copy](#)

Endpoints

Locks

Monitoring

Azure Data Lake Storage Gen2

URL: https://hsinghadls.dfs.core.windows.net/project2-container/Silver_Layer/joined_data

Data View

File System View

CDM Folder View (Beta)

OK Cancel

- Go to ADLS -> Access Key tab -> copy key1.

Microsoft Azure Upgrade Search resources, services, and docs (G+) Copilot ⚙️ 🌐 ⓘ

Home > hsinghadls

hsinghadls | Access keys

Storage account

Search Set rotation reminder Refresh Give feedback

Access keys authenticate your applications' requests to this storage account. Keep your keys in a secure location like Azure Key Vault, and replace them often with new keys. The two keys allow you to replace one while still using the other.

Remember to update the keys with any Azure resources and apps that use this storage account. [Learn more about managing storage account access keys](#)

Storage account name: hsinghadls

key1 Rotate key Last rotated: 16/4/2025 (8 days ago) Copied Hide

Key: OcbachuWzBhpDltvoYmrVSRBbaDc+p4e1v+UcNoXsnyl0KPCH+a/XtnZGn13nFR4...

Connection string: ***** Show

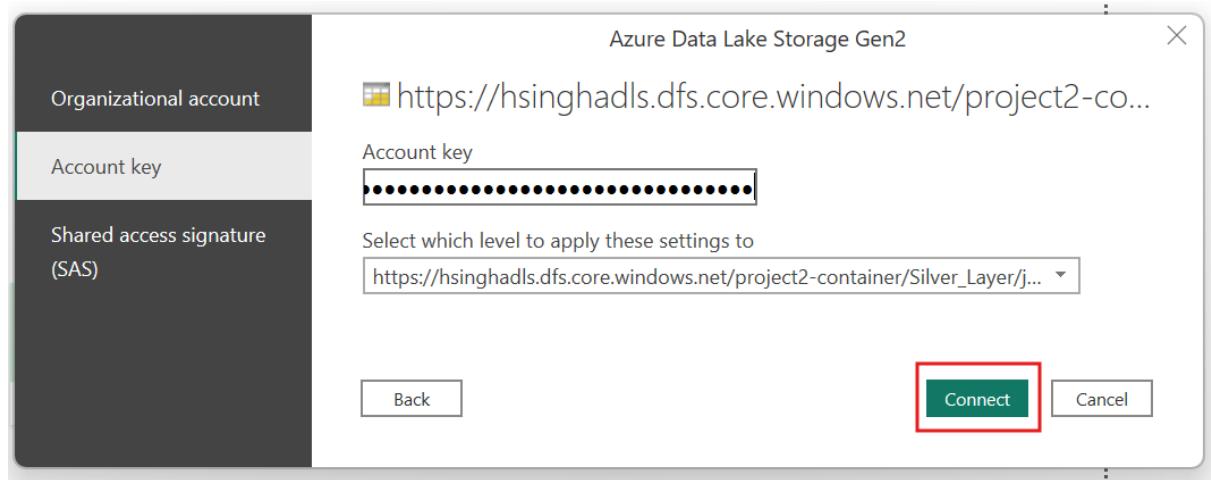
key2 Rotate key

Data storage: Containers, File shares, Queues, Tables

Security + networking: Networking, Access keys (selected), Shared access signature, Encryption, Microsoft Defender for Cloud

Data management

- Paste the copies key1 in the Account Key section -> click on connect.



Step 3: Review the files from selected path -> click on Tranform data.

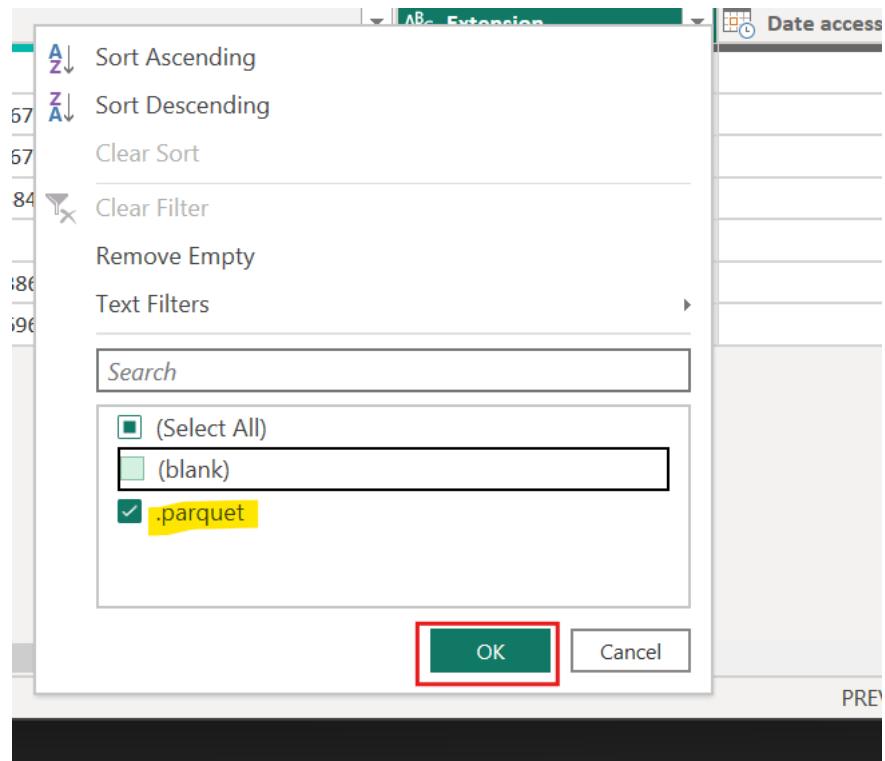
Content	Name	Extension	Date accessed	Date modified	Date created	Attributes
Binary	_SUCCESS		null	24-04-2025 20:21:04	null	Record
Binary	_committed_2922349333291338670		null	24-04-2025 20:21:04	null	Record
Binary	_committed_3547736633615669679		null	24-04-2025 19:40:27	null	Record
Binary	_committed_vacuum252665066484393270		null	24-04-2025 20:21:04	null	Record
Binary	_started_2922349333291338670		null	24-04-2025 20:21:04	null	Record
Binary	part-00000-tid-2922349333291338670-8ac14fad-2ef2... .parquet	.parquet	null	24-04-2025 20:21:04	null	Record
Binary	part-00000-tid-3547736633615669679-b9dc9f50-e434... .parquet	.parquet	null	24-04-2025 19:40:27	null	Record

Combine Load **Transform Data** Cancel

Step 4: Go to Extension column -> select parquet format -> click on ok. (To apply the filter to show all parquet files only).

Content	Name	Extension	Date accessed	Date modified
Binary	_SUCCESS		null	24-04-2025 20:21:04
Binary	_committed_2922349333291338670		null	24-04-2025 20:21:04
Binary	_committed_3547736633615669679		null	24-04-2025 19:40:27
Binary	_committed_vacuum252665066484393270		null	24-04-2025 20:21:04
Binary	_started_2922349333291338670		null	24-04-2025 20:21:04
Binary	part-00000-tid-2922349333291338670-8ac14fad-2ef2-451e-9d71-d55... .parquet	.parquet	null	24-04-2025 20:21:04
Binary	part-00000-tid-3547736633615669679-b9dc9f50-e434-40e2-aa4b-99... .parquet	.parquet	null	24-04-2025 19:40:27

8 COLUMNS, 7 ROWS Column profiling based on top 1000 rows PREVIEW DOWNLOADED AT 04:37 PM



Step 5: Go to Content column click on double arrow to combine the data. -> review the data -> click on OK -> click on Close and apply button.

	Content	Name	Extension
1	Binary	part-00000-tid-2922349333291338670-8ac14fad-2ef2-451e-9d71-d55...	.parquet
2	Binary	part-00000-tid-3547736633615669679-b9dc9f50-e434-40e2-aa4b-99...	.parquet

Parquet

AccountId	CustomerId	Balance	LoanId	LoanAmount	PaymentId	PaymentAmount	PaymentDate	TransactionId	TransactionAmount
54	42	5500.5	54	30000.5	23	1200	23-01-2024	89	
88	1	8900	88	27500	17	900	17-01-2024	23	
82	2	8300.5	82	20000.5	71	3600	11-03-2024	83	
92	44	9300	92	20000	81	4100	21-03-2024	10	
21	53	300.25	21	10000.25	20	1050	20-01-2024	20	
21	53	300.25	21	10000.25	20	1050	20-01-2024	86	
29	58	75.25	29	32500.25	48	2450	17-02-2024	13	
35	62	175.75	35	25000.75	94	4750	03-04-2024	62	
38	15	3900.5	38	27500.5	67	3400	07-03-2024	90	
72	17	7300	72	20000	61	3100	01-03-2024	51	
33	85	150.25	33	15000.25	12	650	12-01-2024	70	
33	85	150.25	33	15000.25	100	1000	10-04-2024	70	
33	85	150.25	33	15000.25	101	1000	01-01-1900	70	
45	68	300.25	45	25000.25	1	100	01-01-2024	1	
86	21	8700.5	20	37500	29	1500	29-01-2024	53	
86	21	8700.5	86	17500.5	35	1800	04-02-2024	53	
51	72	375.75	51	10000.75	50	2550	19-02-2024	41	
84	40	8500	84	30000	53	2700	22-02-2024	63	
5	56	500	5	25000	64	3250	04-03-2024	18	
15	47	700.75	15	25000.75	74	3750	14-03-2024	38	

The data in the preview has been truncated due to size limits.

OK Cancel

Untitled - Power Query Editor

File Home Transform Add Column View Tools Help

Close & Apply (highlighted with a red box)

New Source Recent Sources Enter Data Data source settings Manage Parameters Refresh Preview Advanced E Manage ▾

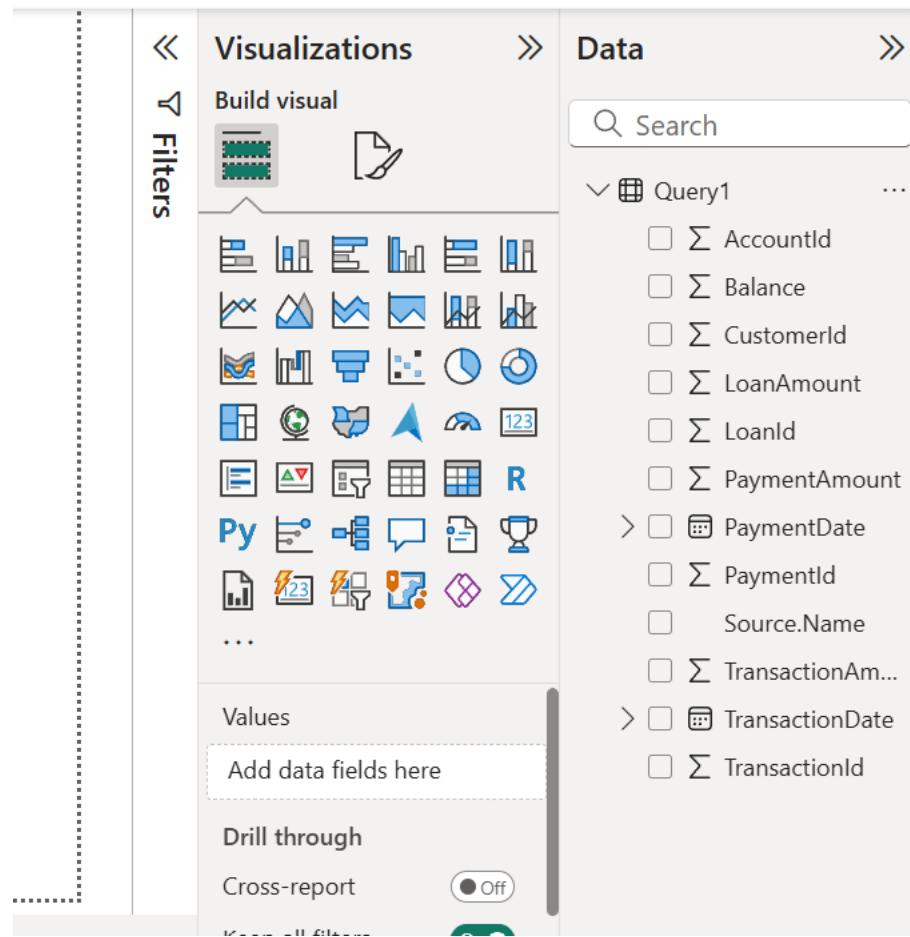
Queries [9]

- Transform File from Q...
- Helper Queries [3]
- Sample File

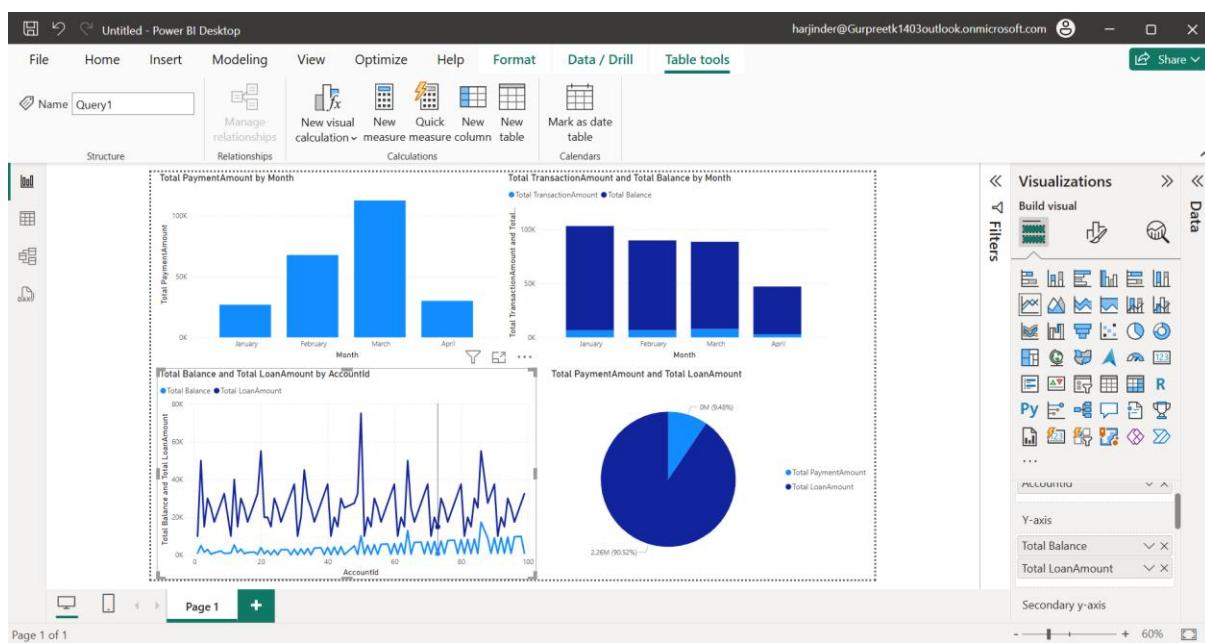
= Table.TransformColumnT

source.Name
0000-tid-2922349333291338670-8ac14fad-2ef2-451
0000-tid-2922349333291338670-8ac14fad-2ef2-451

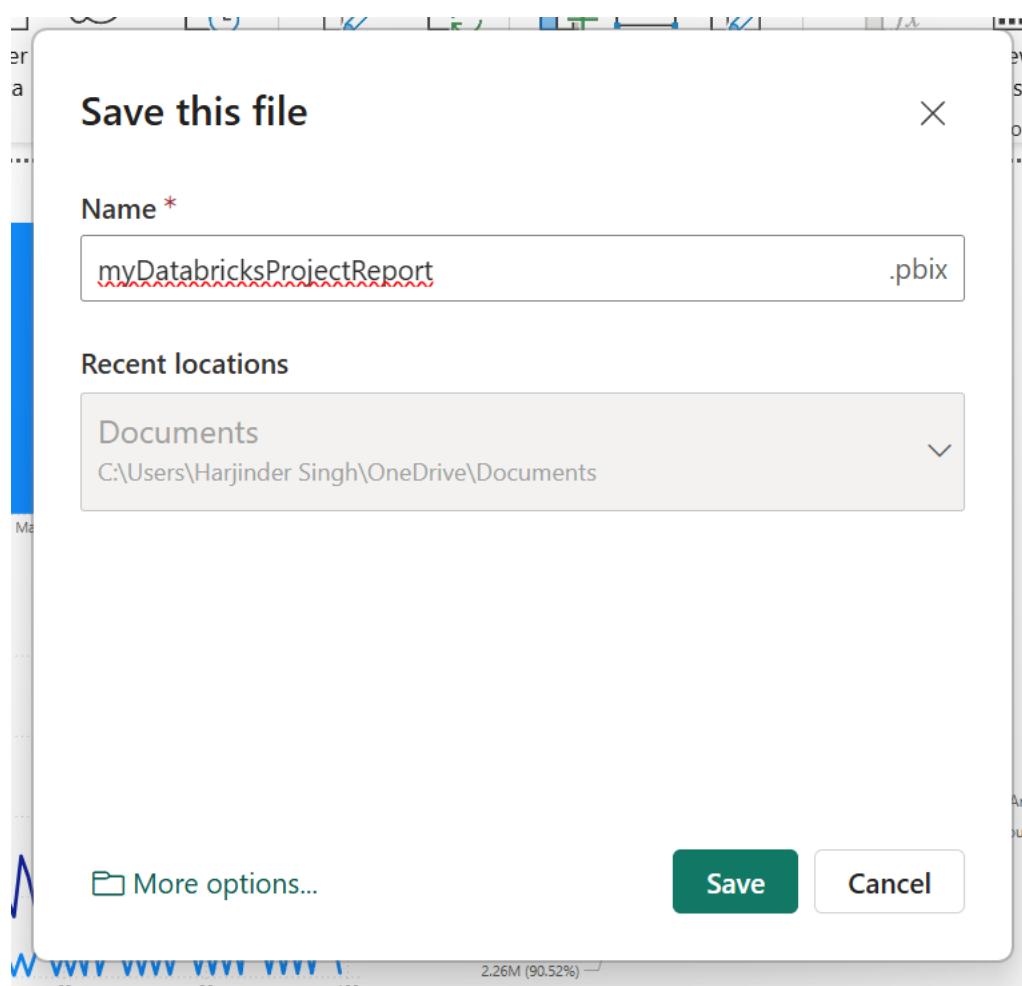
Step 6: Next, select the data columns to visualize the data from data section.



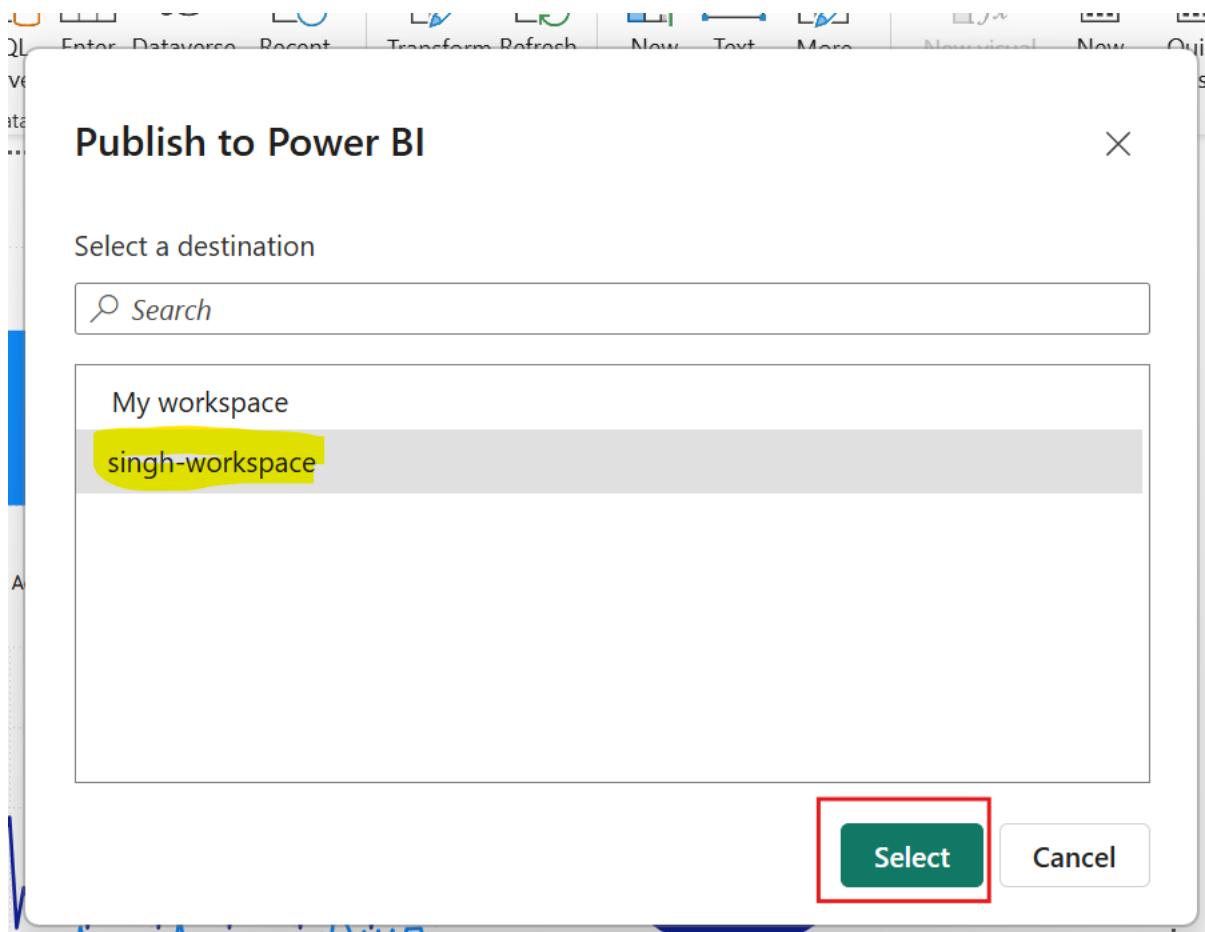
Step 7: Next, create some visuals and report by using data columns.



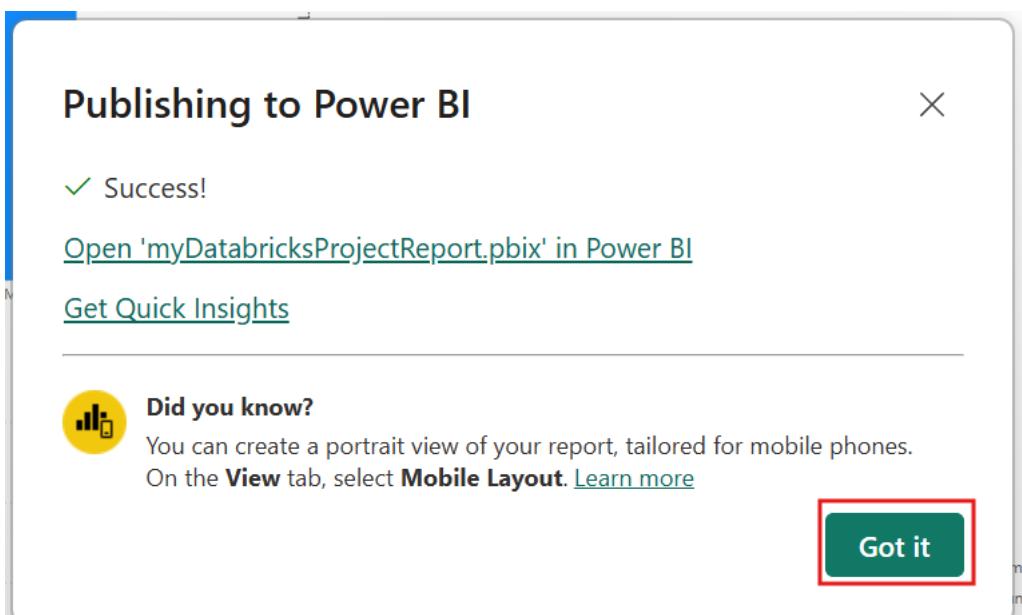
Step 8: Next, save the file -> select the fabric workspace to publish the power BI report -> click on select.



A screenshot of the Power BI service interface. The top navigation bar shows the user's email 'harjinder@Gurpreetk1403outlook.onmicrosoft.com'. The ribbon menu has sections like 'Insert', 'Calculations', 'Sensitivity', 'Visualizations', 'Build visual', and 'Filters'. The 'Visualizations' section is highlighted with a red box. A tooltip above the 'Publish' button says 'Publish this report online in the Power BI service.' The main workspace shows a report titled 'Total Balance by Month' with two blue squares below it.



Step 9: Finally, review the published report in the fabric workspace.



Fabric singh-workspace

Search

Trial: 54 days left

Create deployment pipeline Create app Manage access Workspace setting

New item New folder Import Migrate

Choose from predesigned task flows or add a task to build one (preview)

Select from one of Microsoft's predesigned task flows or add a task to start building one yourself.

Select a predesigned task flow Add a task

Name	Type	Task	Owner	Refreshed	Next refresh	Endorsement	Sensitivity	Included in app
myDatabricksProjectReport	Report	—	singh-work...	4/24/2025, 5:4...	—	—	—	No
myDatabricksProjectReport	Semantic ...	—	singh-work...	4/24/2025, 5...	N/A	—	—	
MyProjectReport	Report	—	singh-work...	4/22/2025, 4:4...	—	—	—	No
MyProjectReport	Semantic ...	—	singh-work...	4/22/2025,...	▲ N/A	—	—	
singhLakehouse	Lakehouse	—	Harjinder S...	—	—	—	—	
singhLakehouse	Semantic ...	—	singh-work...	4/19/2025, 3...	N/A	—	—	
singhLakehouse	SQL analyti...	—	Harjinder S...	—	—	—	—	

