TRANSACTIONS AND LOAN DATA FOR A CUSTOMER

Bootcamp Project - 2

APRIL 24, 2025

BY HARJINDER SINGH

Table of Contents

Introduction

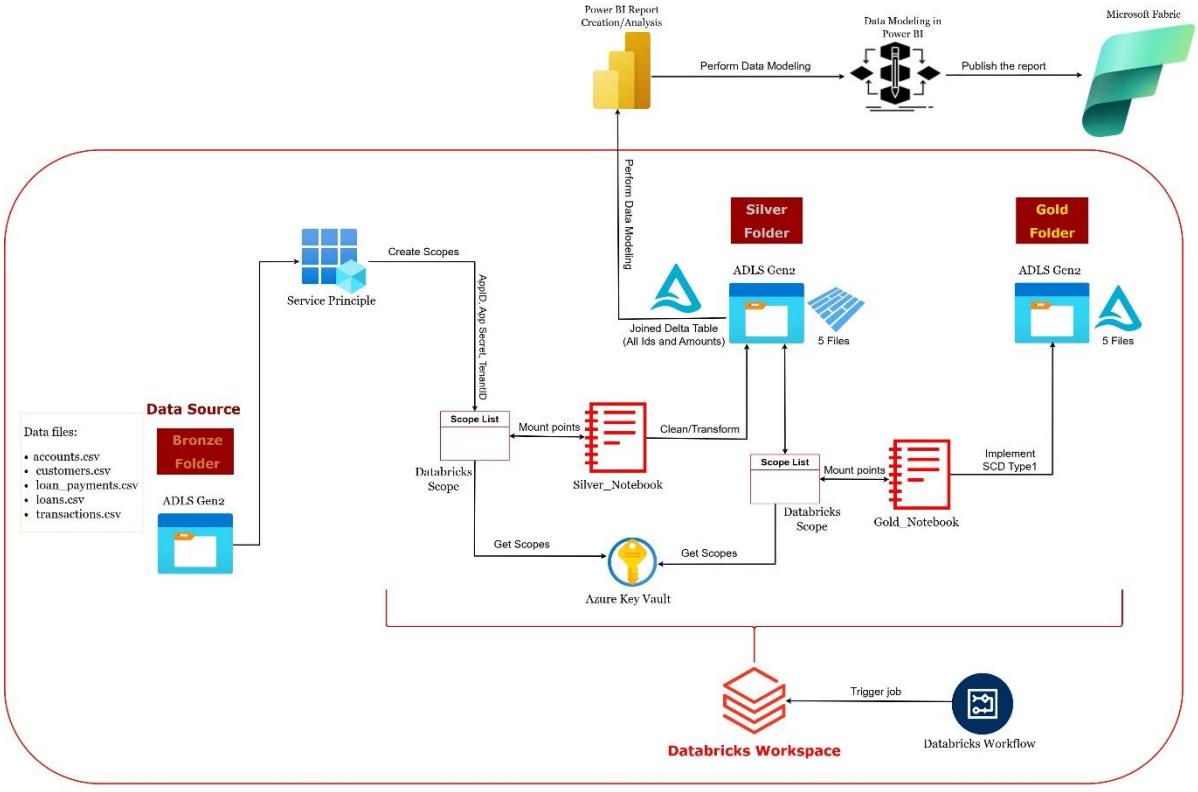
This project outlines the end-to-end implementation of a modern data pipeline using **Azure Data Lake Storage Gen2**, **Databricks**, and **Power BI**. The goal is to extract raw banking-related data files from a source location, clean and transform the data using **PySpark in Databricks**, apply **Slowly 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.

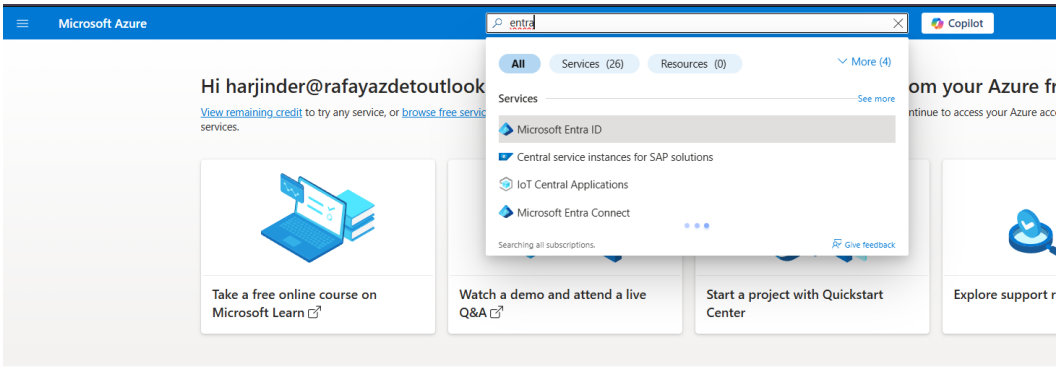
Architecture Followed:



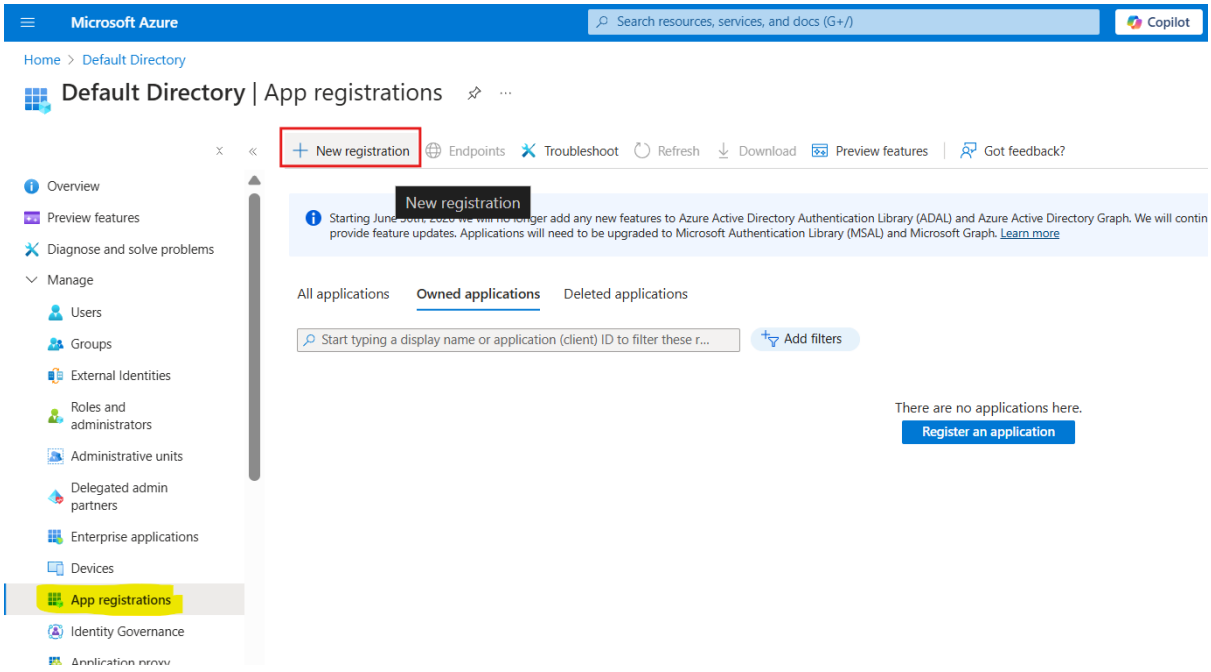
Service Principal Setup

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

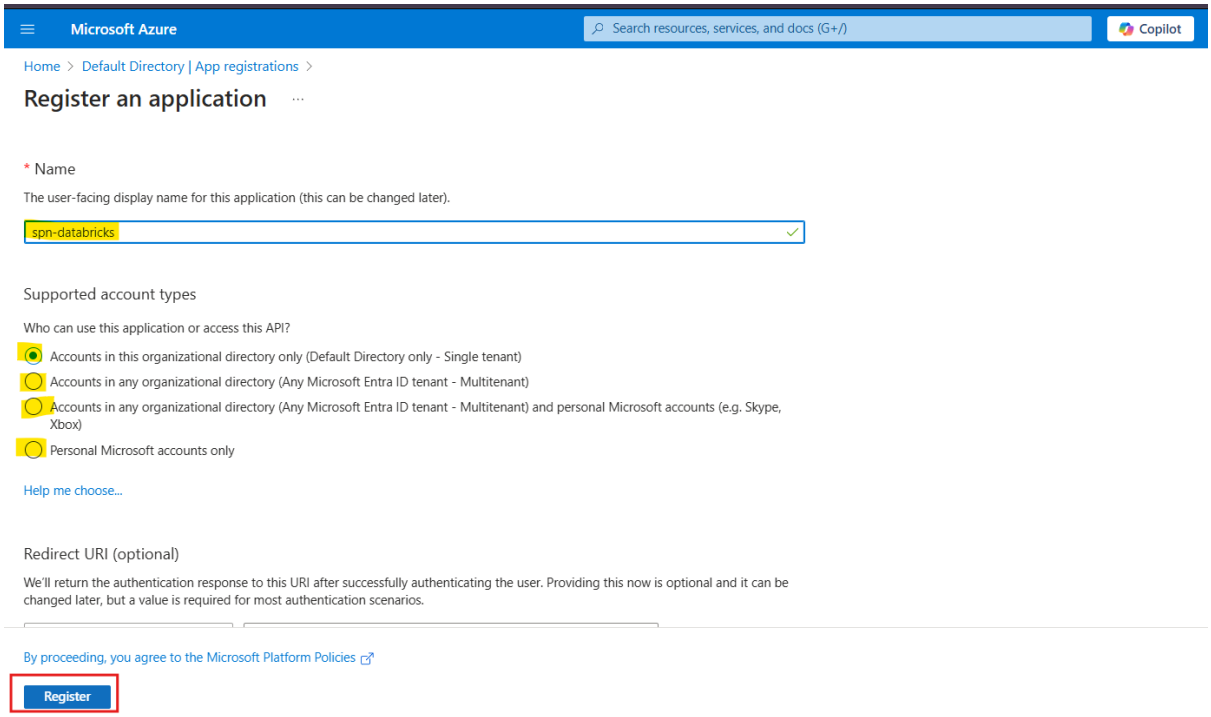
– Open the Azure Account -> Search for the **Microsoft Entra ID** in the search bar.



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



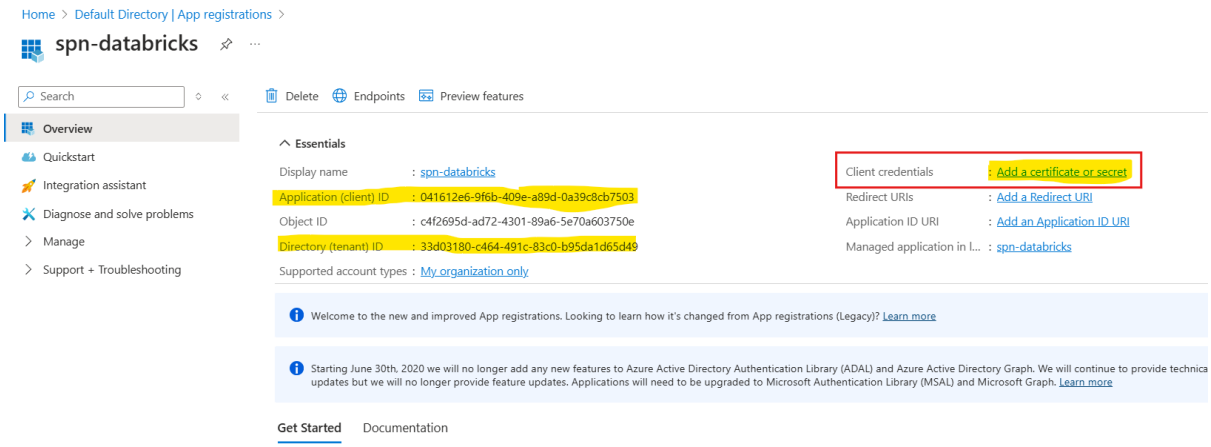
– Next, enter the **name** for new app registration and select any option for Supported account types options -> click on **Register** button.



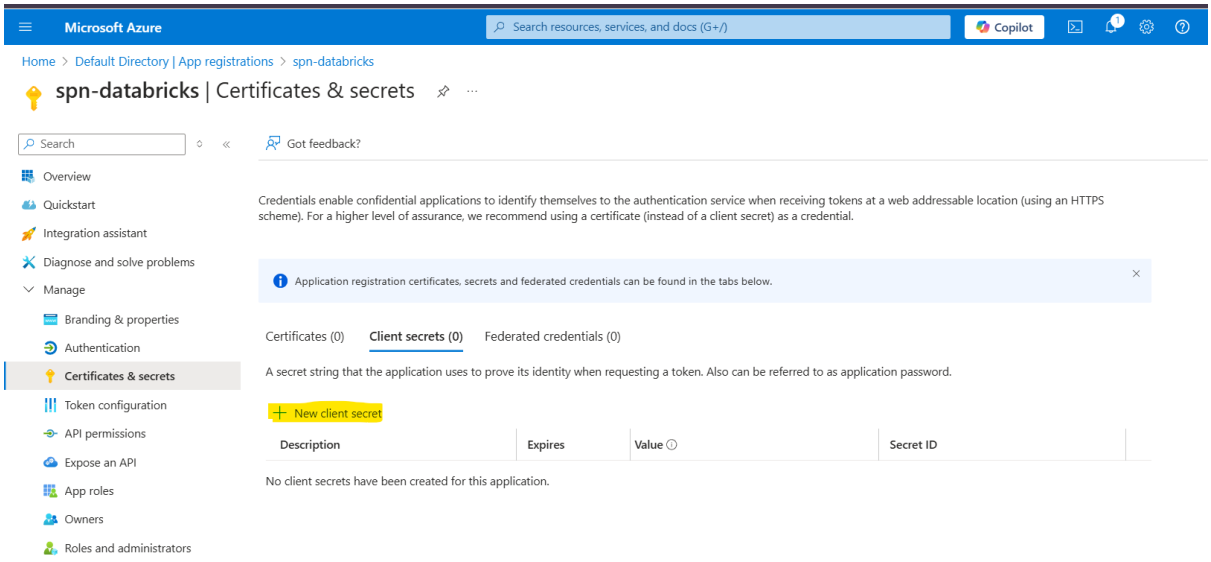
– Now the registration is done, we can see the Essentials related to the registration.

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

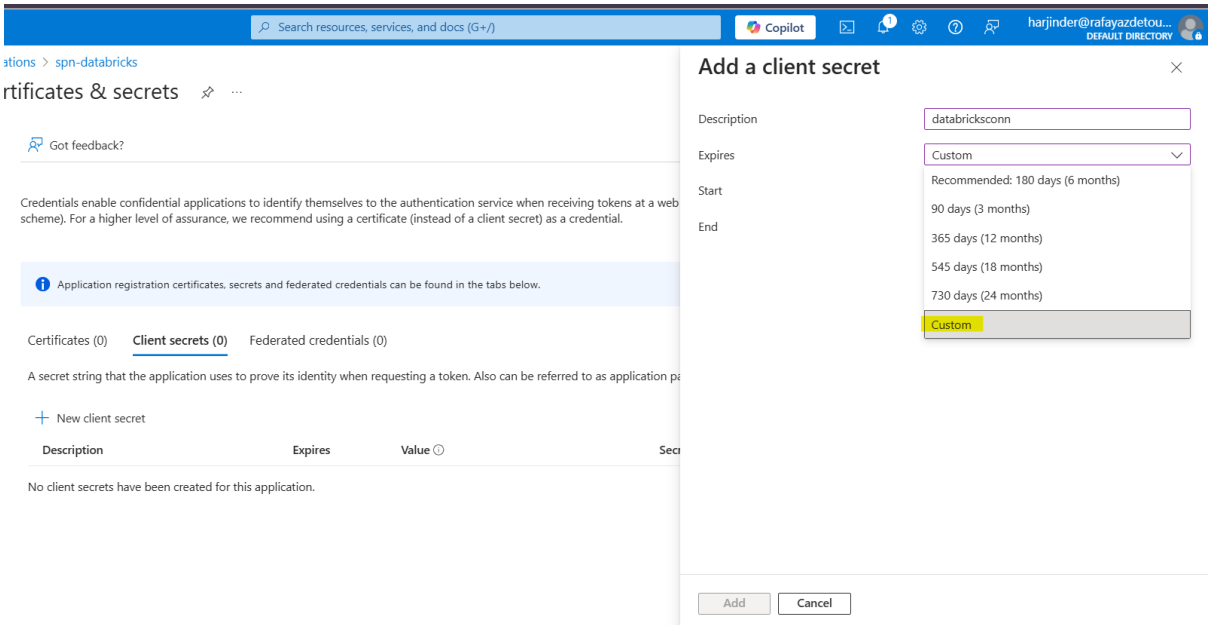
  Next, click on **Add a certificate or secret** option of Client credentials in registered app essentials.



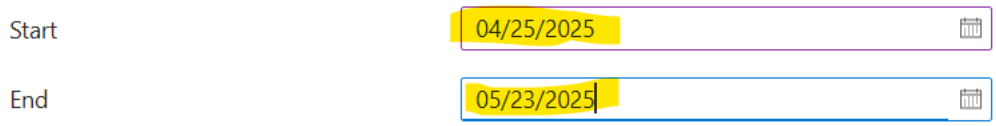
– Next, click on **+ New client secret** option under the client secrets tab.



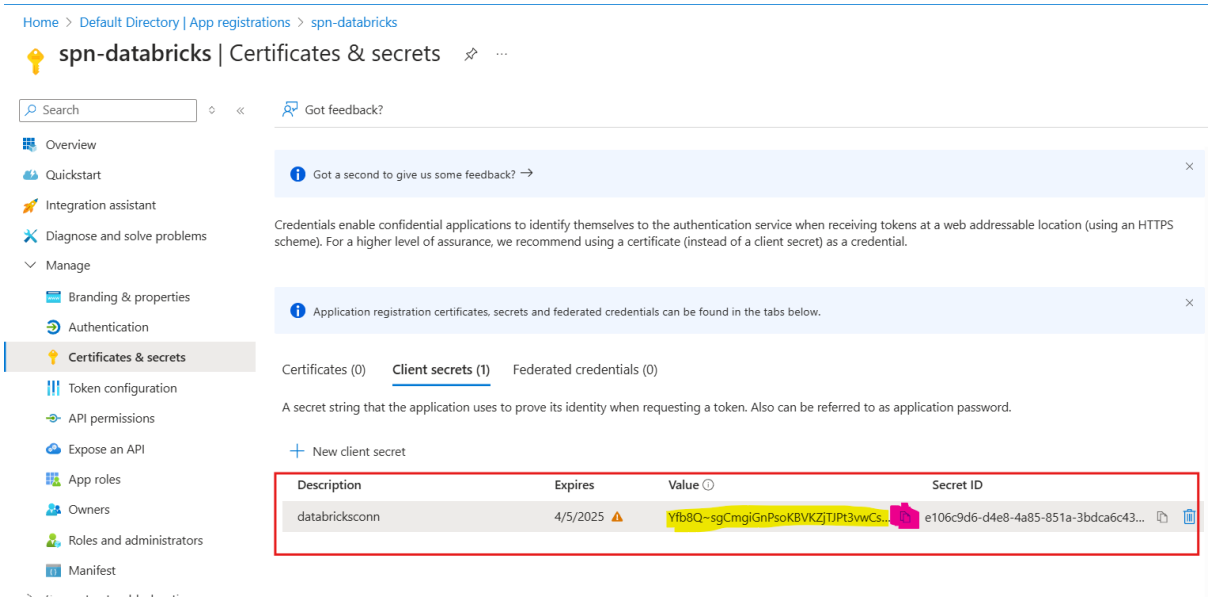
– Now, enter the **description** and **expires** option for client secret. (In my case, I opted for **Custom**).



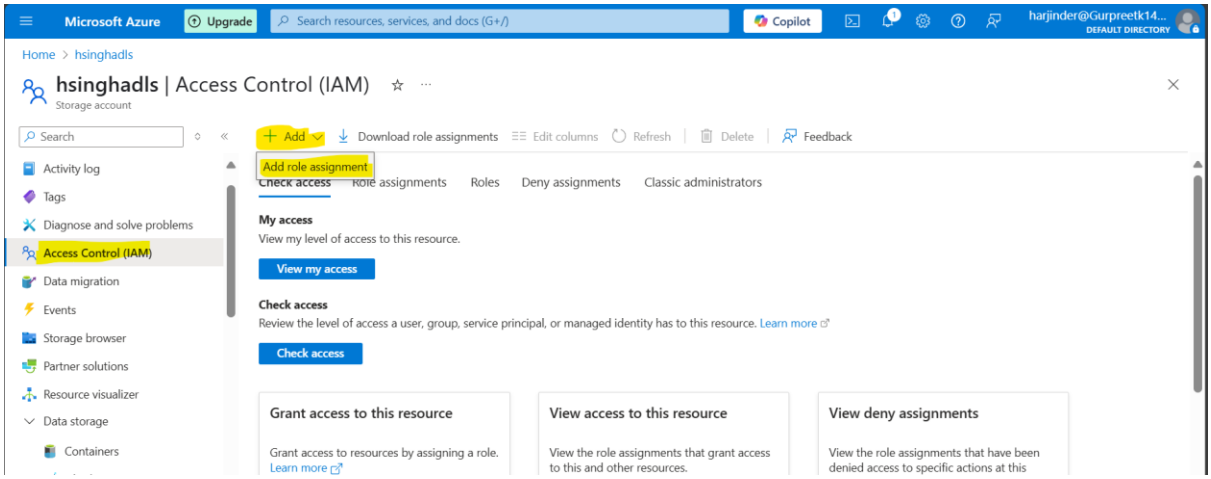
– And, select the **Start** and **End** Date for the secret -> click on **Add** button.
As I opted for custom option for Expires field so I am giving the randomly about 1 week.
Usually, in the industry, they select for **90 days and 180 days** as Expires values.



– 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.)

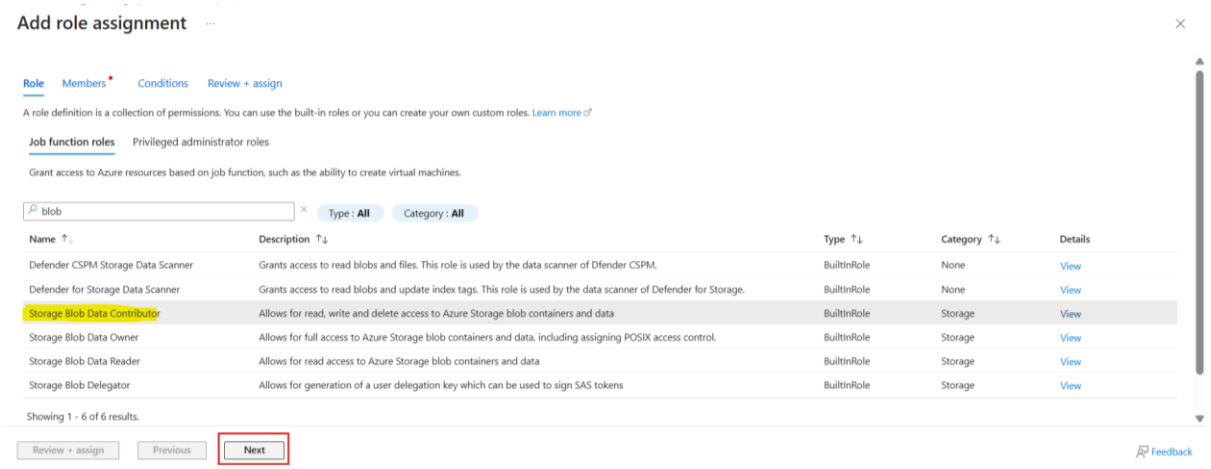


– Next, Move to **ALDS Gen2 storage** account -> go to **Access Control (IAM)** -> click on **+ Add** -> select **Add role assignment.**

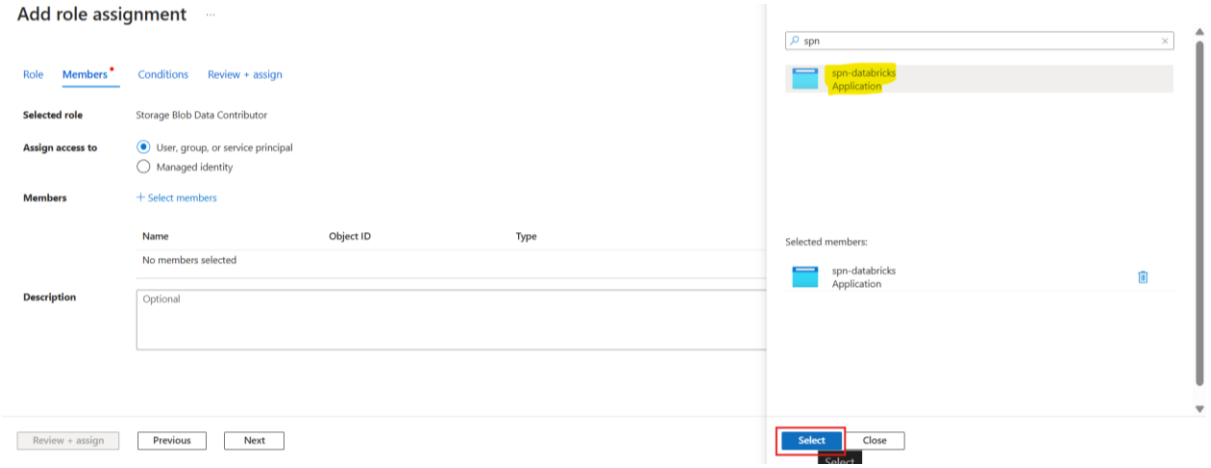


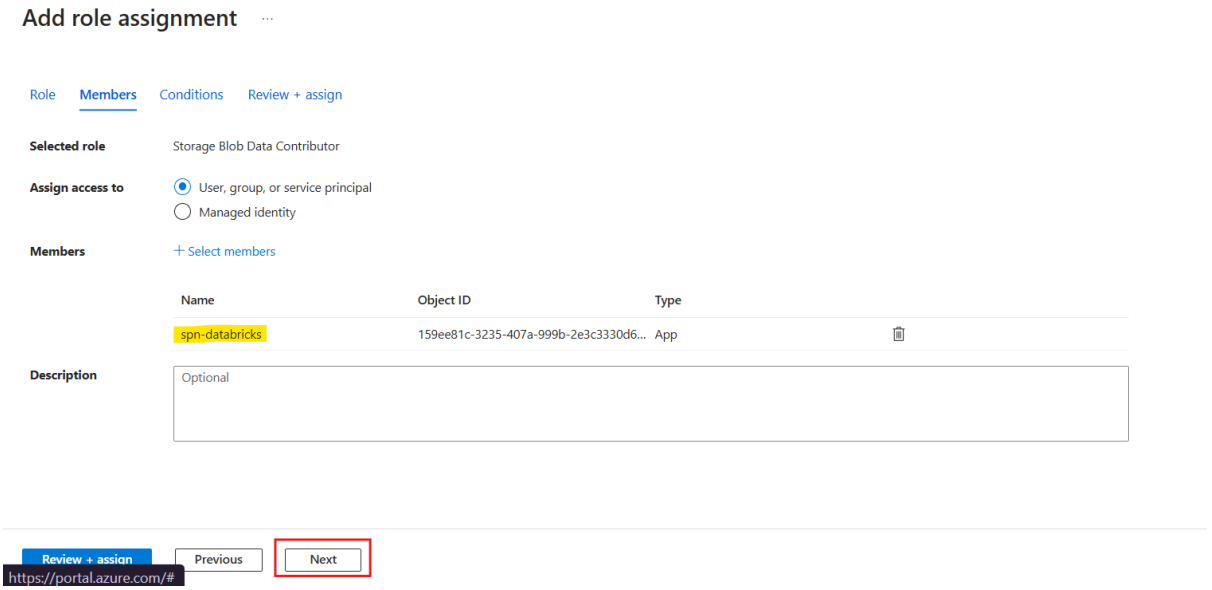
we need to give the Storage blob data contributor access to the Service principle so that we can perform the action using service principle to the ADLS gen2 storage account.

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

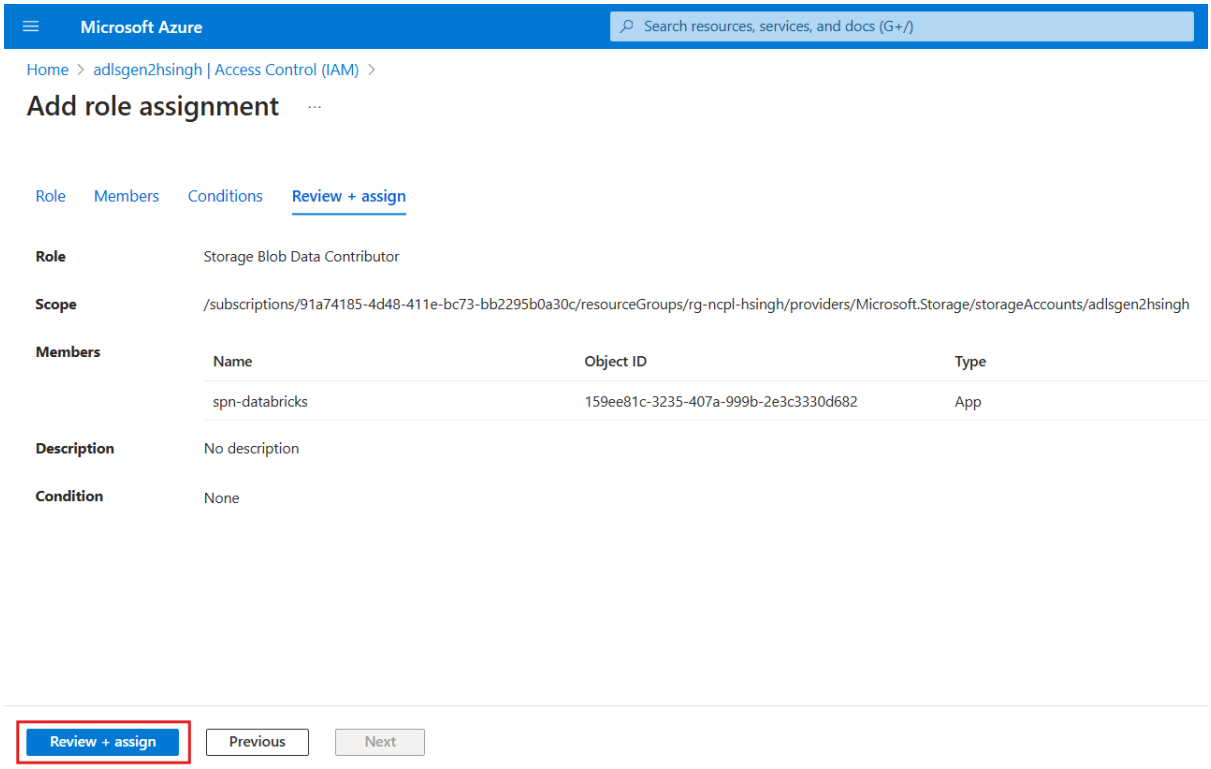


– Next, select the **spn-databricks** as member -> click on Next. (here, spn-databricks is our created service principle)

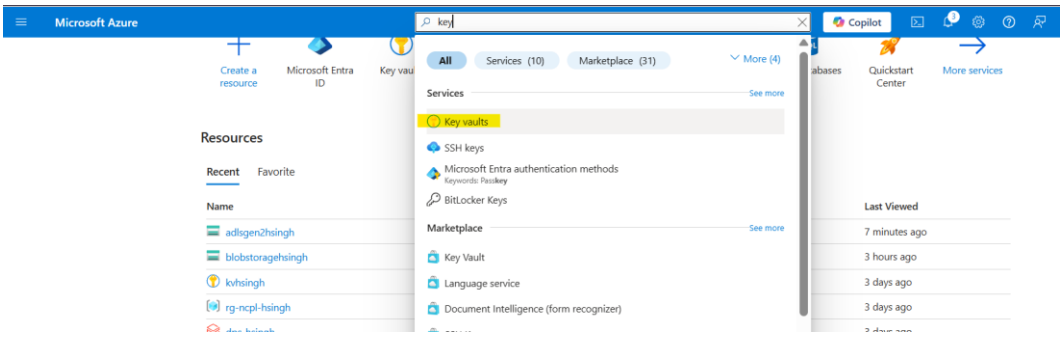




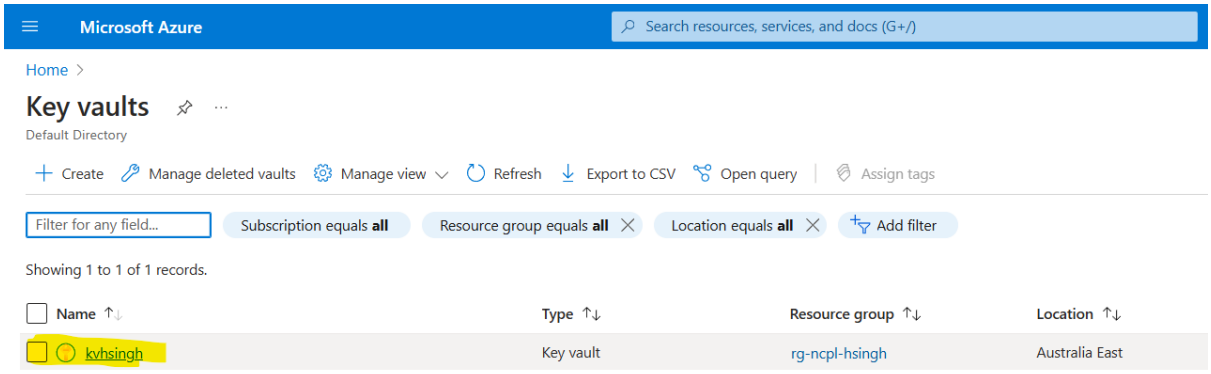
– Click on **Review + assign** button and complete the role assignment



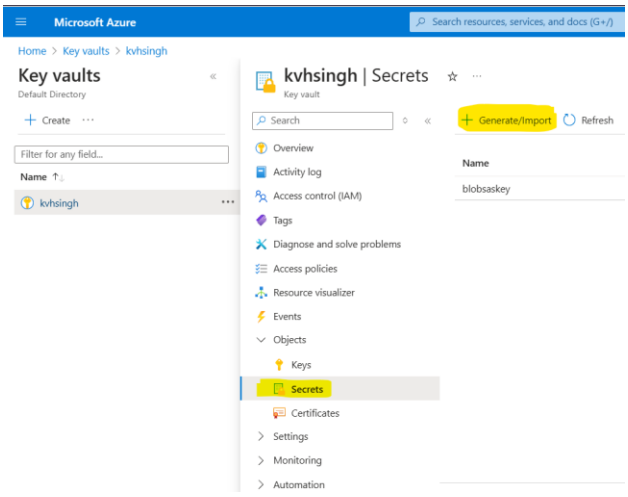
– Next, Move to Home Page of Azure account -> Search for the **Key Vault** in the search bar.



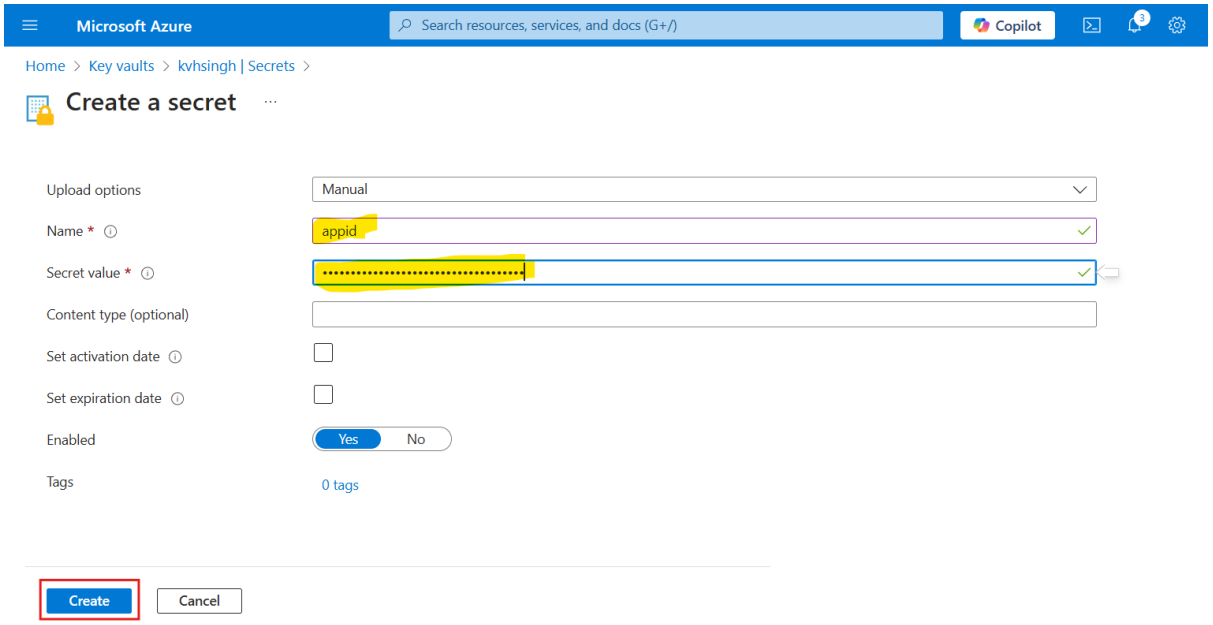
– Here, the previously created Secret is available (with name as **kvhsingh**), the click on the available secret. (we can create new secret as well but here I am using the previous one).



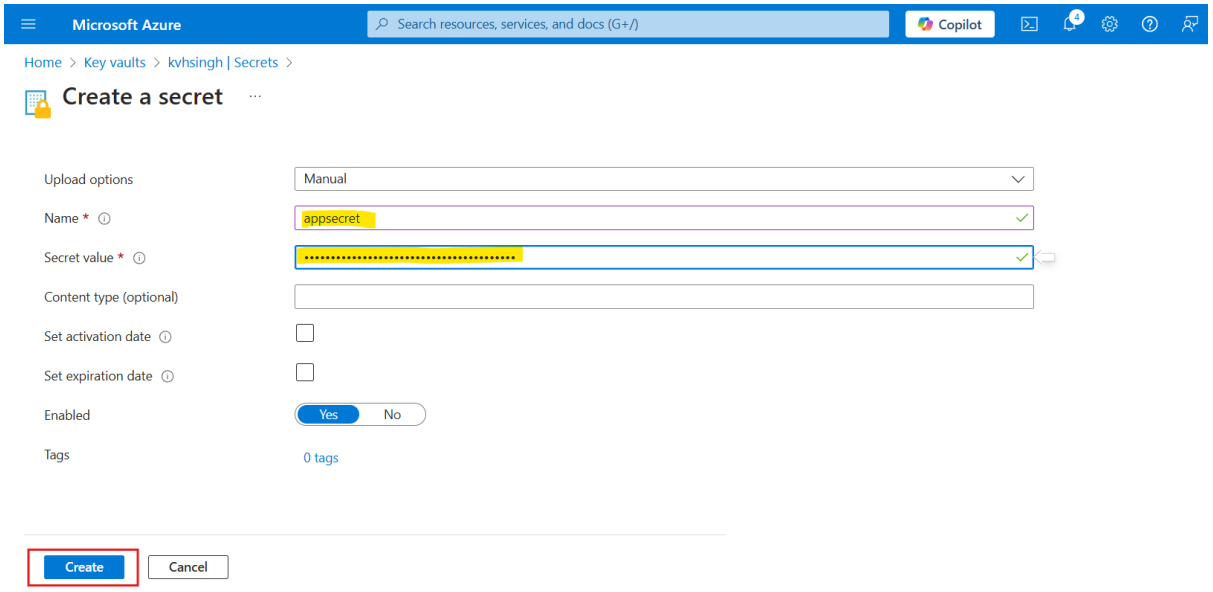
– Next, select the **Secrets** option under the Objects -> click on **+ Generate/Import** option.



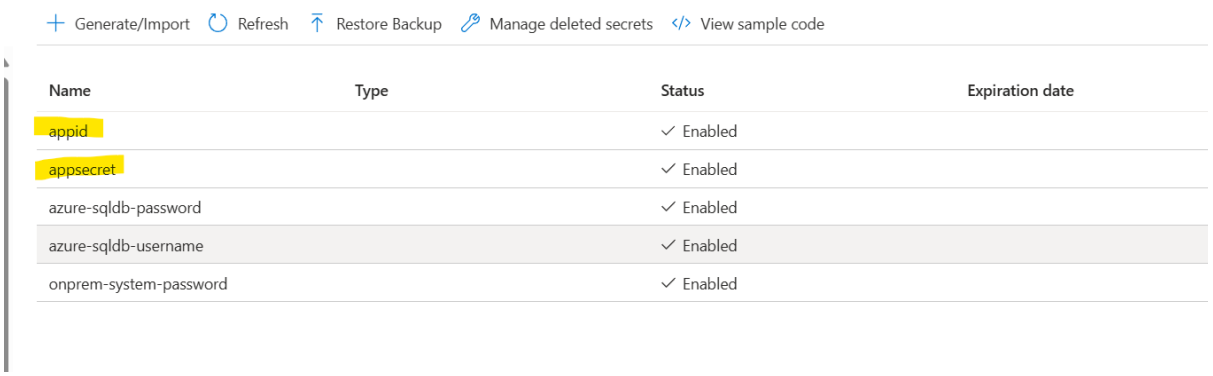
– Name the secret (in my case, I named it as ***appid***) and enter the **Application ID & Directory ID** in the **secret value** field -> click on **Create** button.



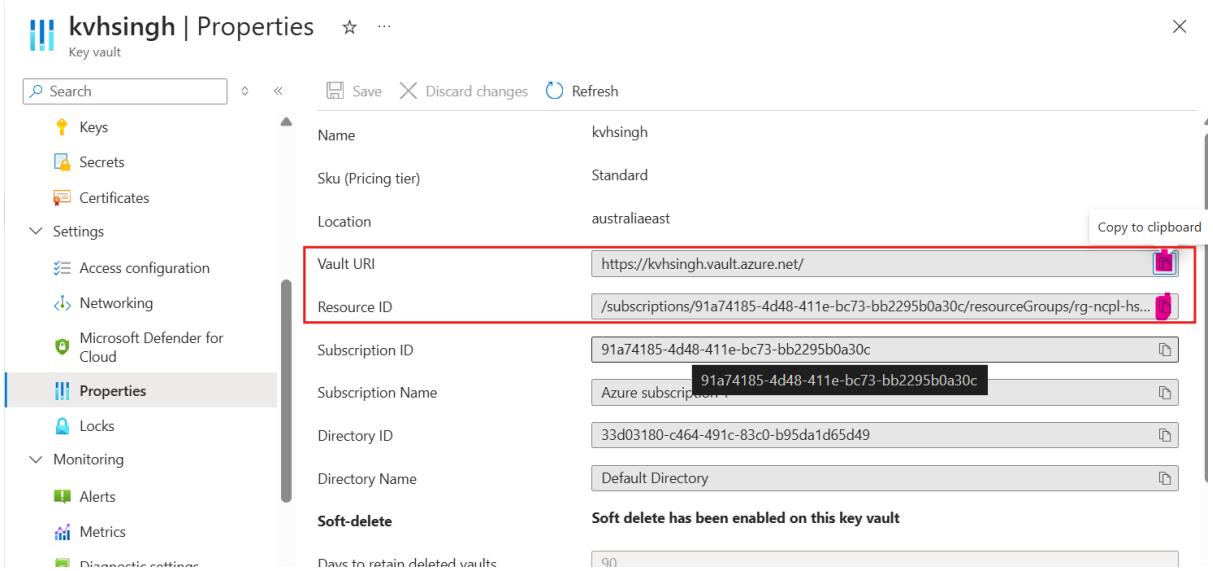
– Next again generate a new secret and name the secret (in my case, I named it as ***appsecret***) and enter the **secret value** from Service Principle created secret in the **secret value** field -> click on **Create** button.



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



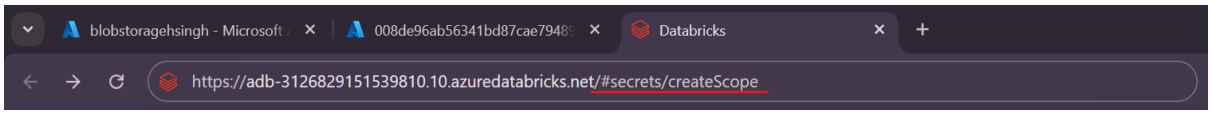
– Next, Go to Key Vault and open the **Properties** for new created **Key Vault** (***kvhsingh***) and **copy** the **Vault URI** and **Resource ID**.
We can use these values to create new scope in databricks to access the storage account.



Databricks Scope Creation

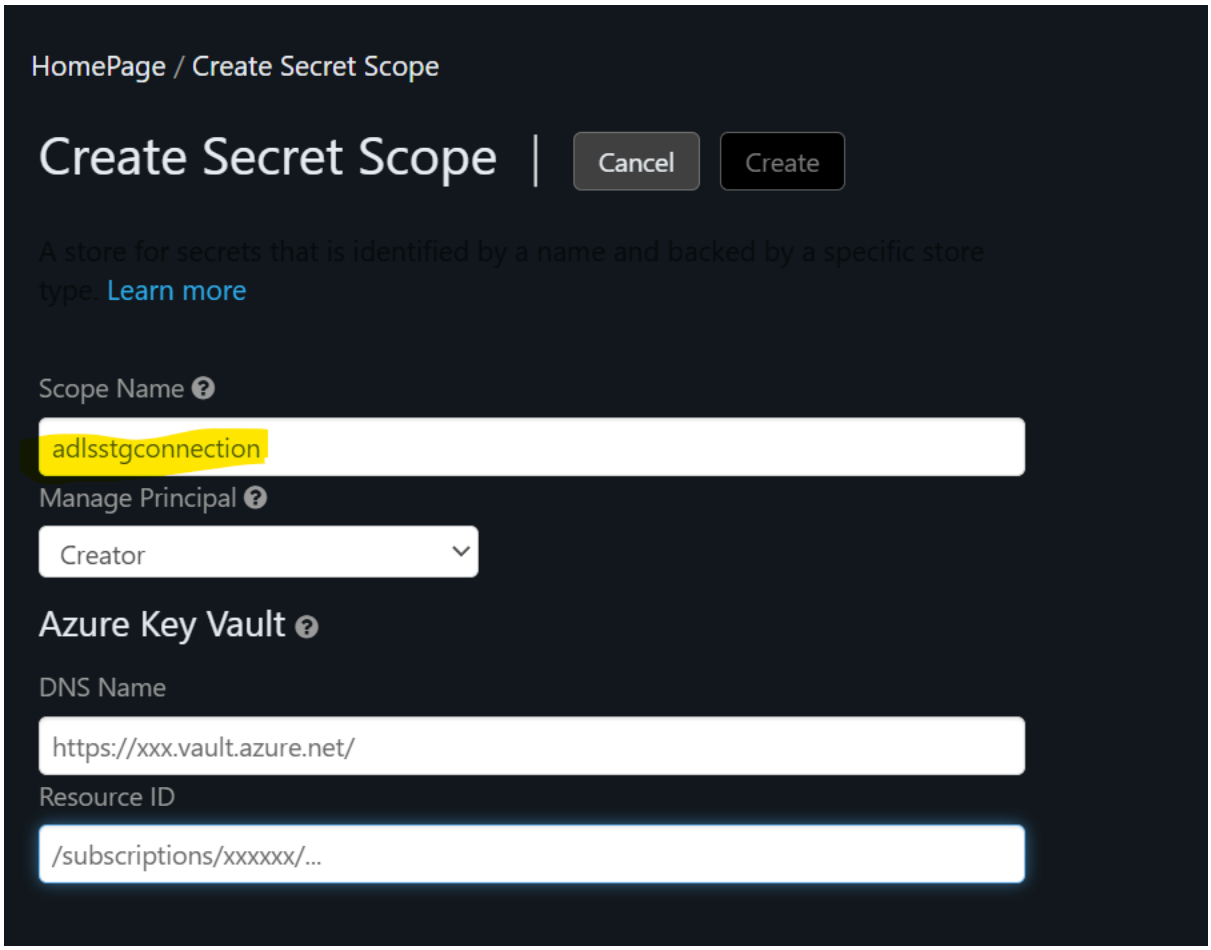
➢ **Steps to create a new scope in databricks using key vault properties:**

- 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

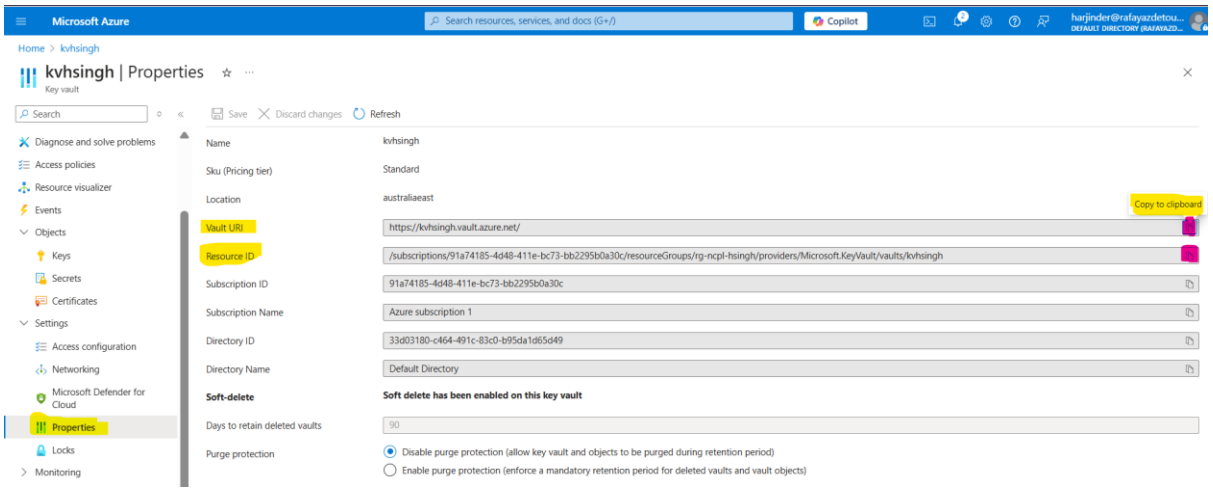


- Now, the new Page is opened in the Databrick workspace to create a **Secret scope**.

  Enter the name of Scope (in my case, I named it as *adlsstgconnection*), and select the manage principle as **Creator,**



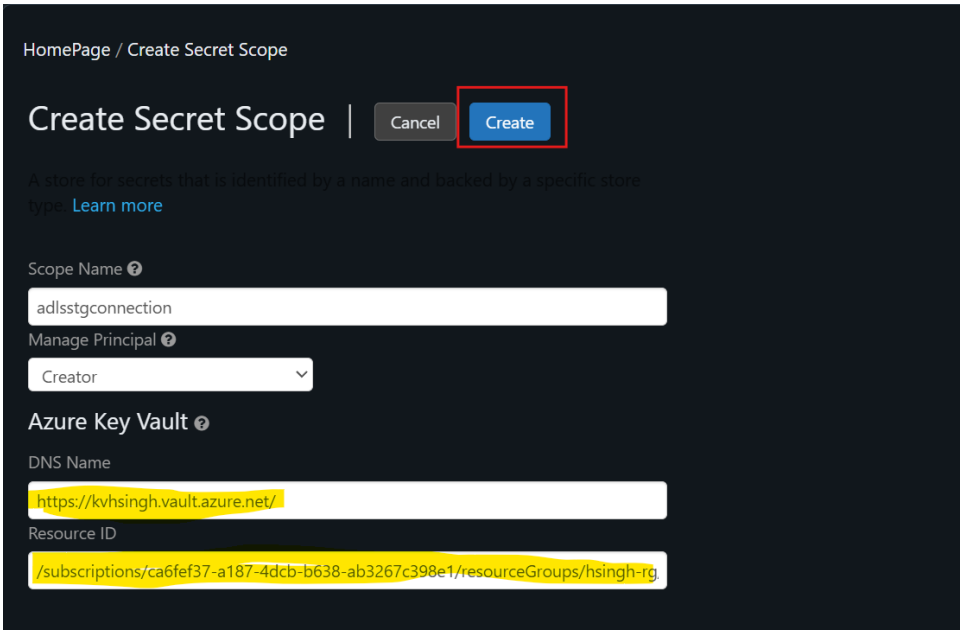
– Next, Go to Key Vault and open the **Properties** for new created **Key Vault** (***kvhsingh***) and **copy** the **Vault URI** and **Resource ID**.



– 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.



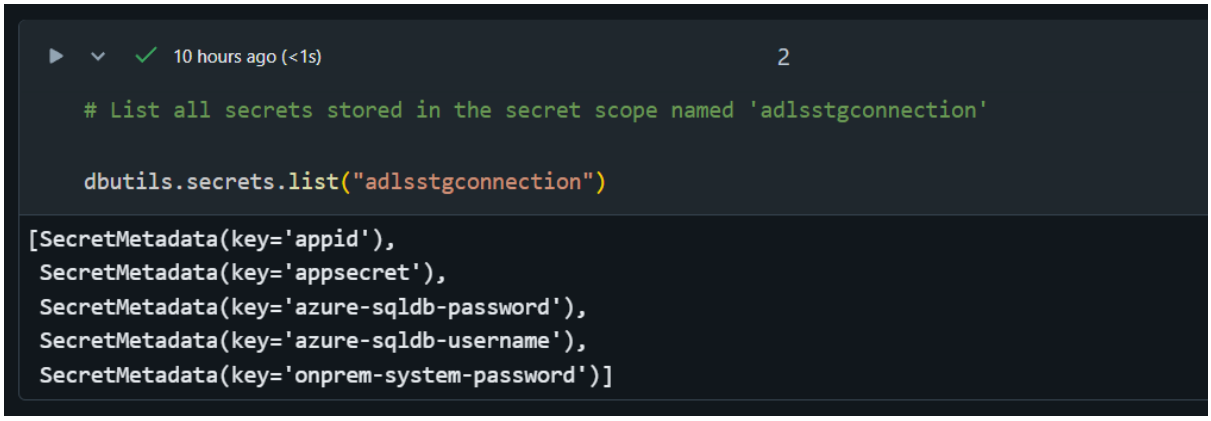
MyUtilityFunctions Notebook

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

    dbutils.secrets.listScopes()

- **List Secrets in a Specific Scope**
  - Lists all the secrets stored under the secret scope named "adlsstgconnection".
  - Code:

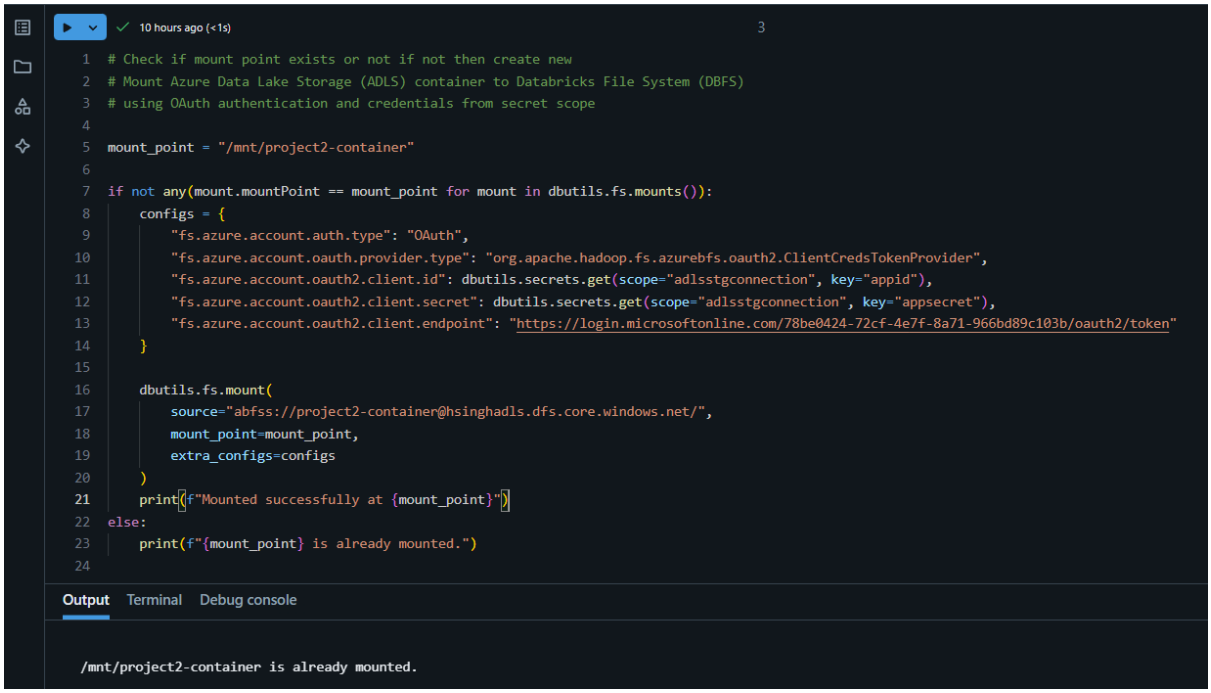
    dbutils.secrets.list("adlsstgconnection")



- **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:

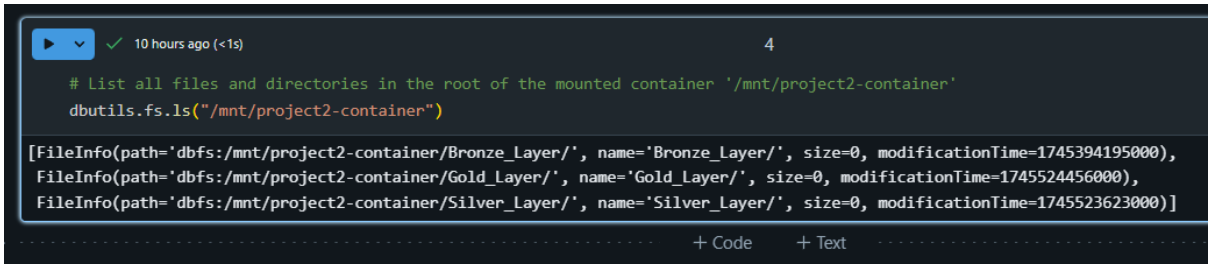
```
mount_point = "/mnt/project2-container"

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.")
```



– **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")



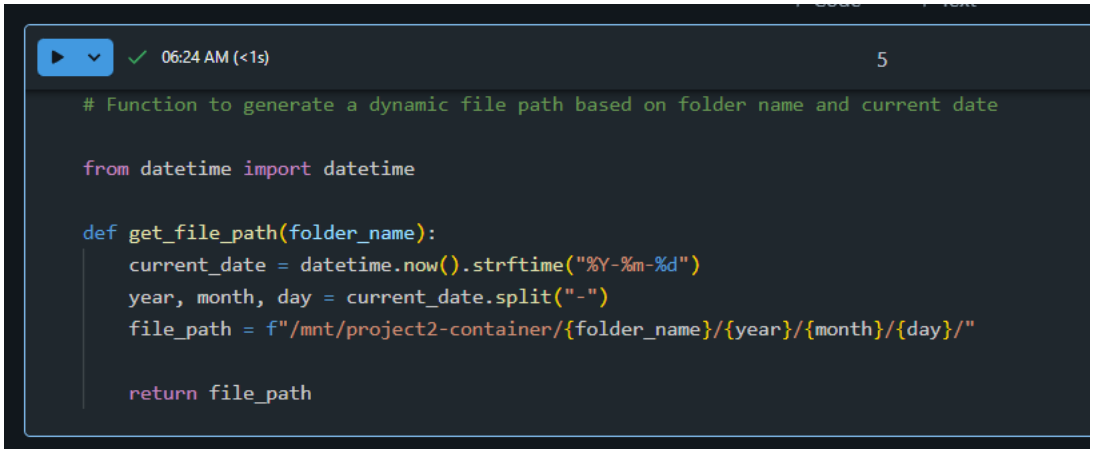
– **Function to Generate File Path Based on Current Date**

- 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")

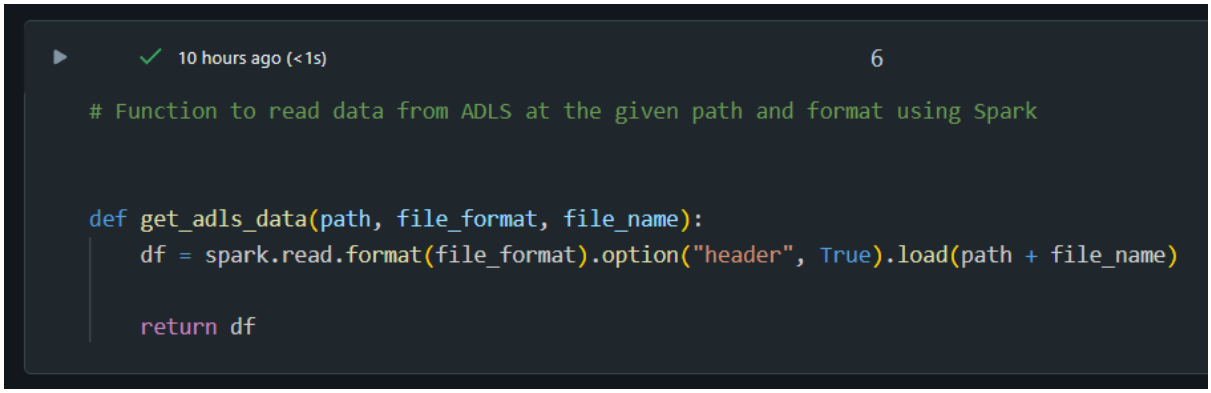
```
    year, month, day = current_date.split("-")
    file_path = f"/mnt/project2-
container/{folder_name}/{year}/{month}/{day}/"

    return file_path
```



– **Function to Read Data from ADLS**

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

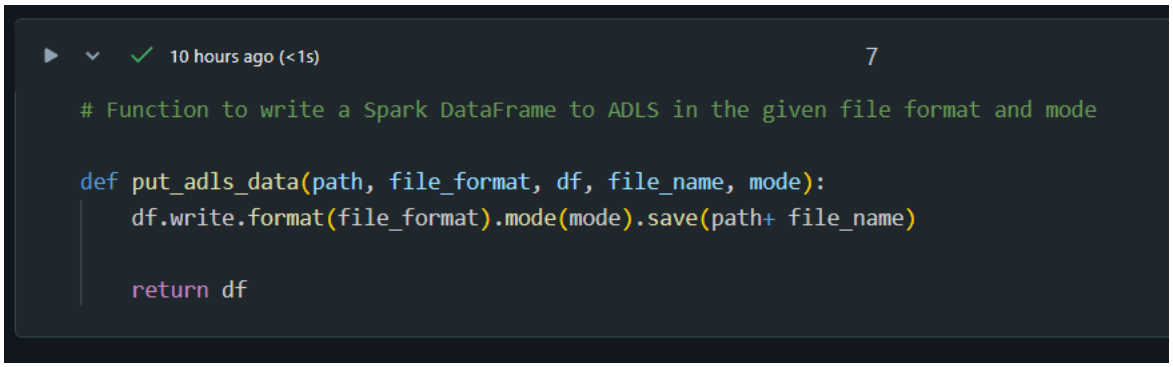
```
def get_adls_data(path, file_format, file_name):
    df = spark.read.format(file_format).option("header", True).load(path +
file_name)

    return df
```



– **Function to 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:

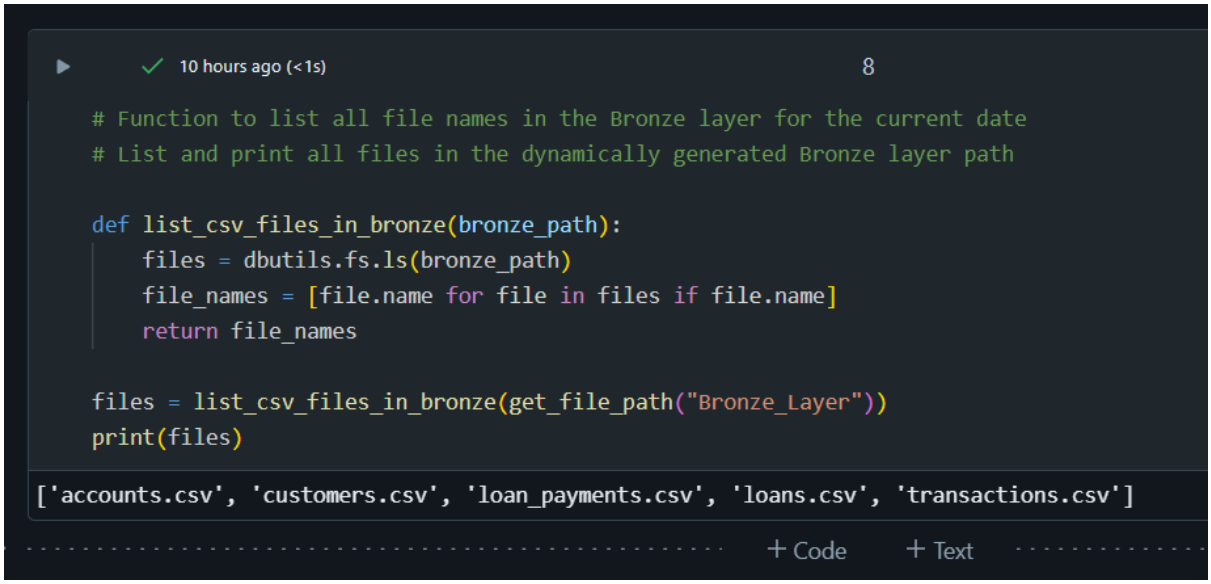
```
def put_adls_data(path, file_format, df, file_name, mode):
  df.write.format(file_format).mode(mode).save(path+ file_name)

  return df
```



– **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**

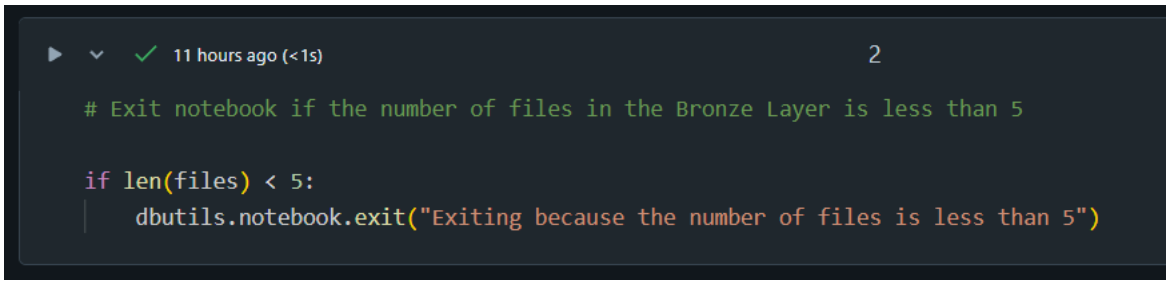
- Utilizes pre-defined utility functions for modular code reuse and simplifies ADLS interactions.
- Code:

  %run "/Workspace/Project2/MyUtilityFunctions"

– **File Count Check**

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

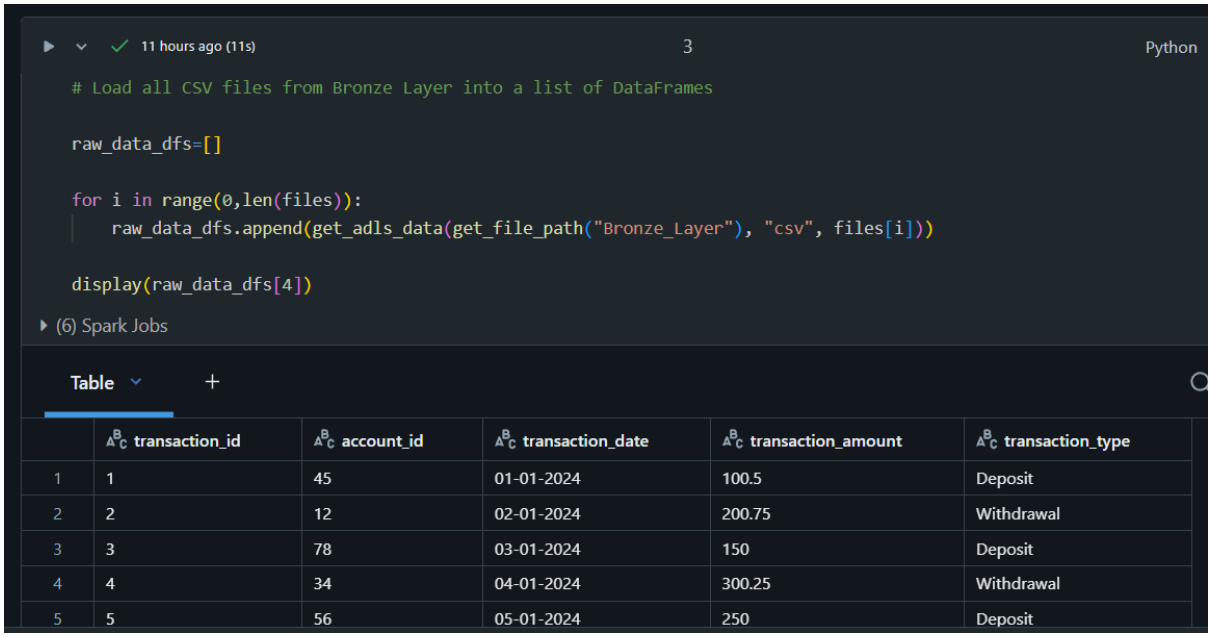
  if len(files) < 5:
  dbutils.notebook.exit("Exiting because the number of files is less than 5")



– **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])
```



11 hours ago (11s) 3 Python

```
# 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 +

	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
4	4	34	04-01-2024	300.25	Withdrawal
5	5	56	05-01-2024	250	Deposit

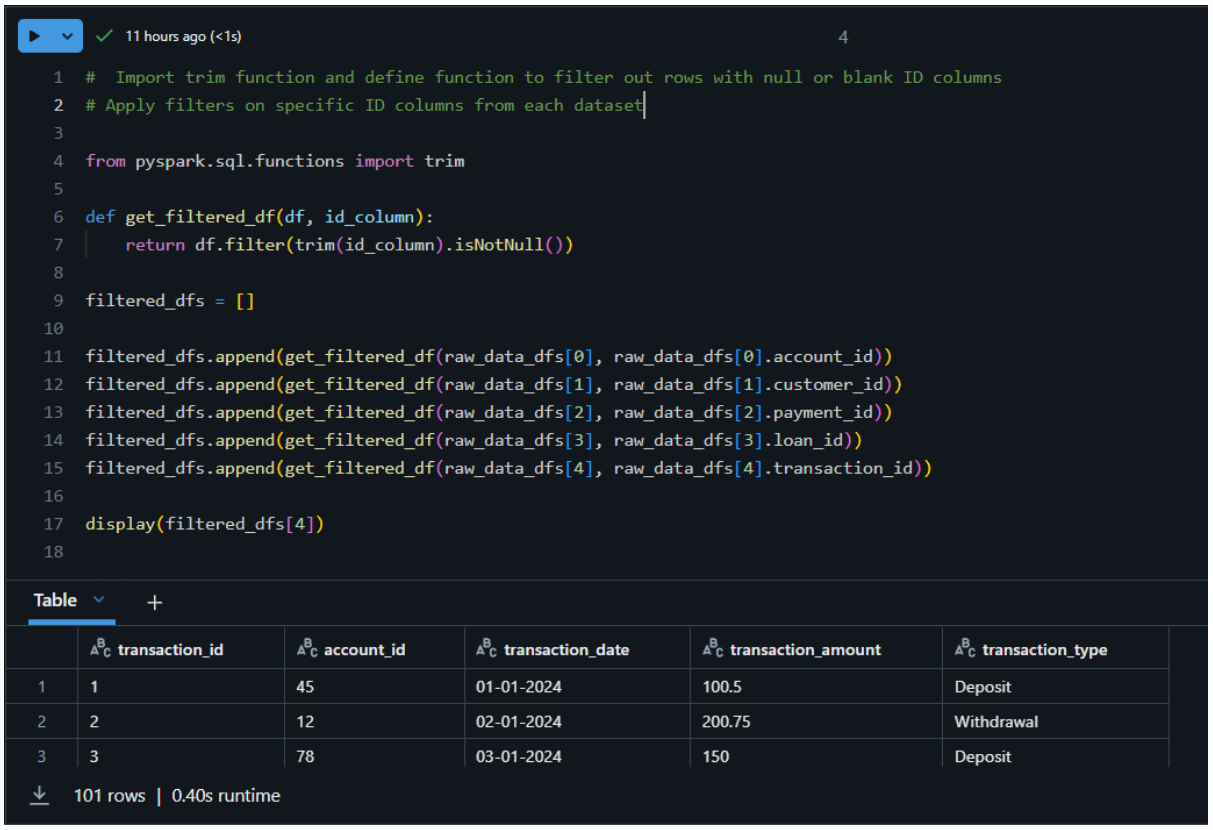
– **Null ID Filtering**

- 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])



– **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

```python
# Define function to return distinct records
# Create a list of distinct (deduplicated) DataFrames

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])
```

Table +

	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

– **Define Fill Values**

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

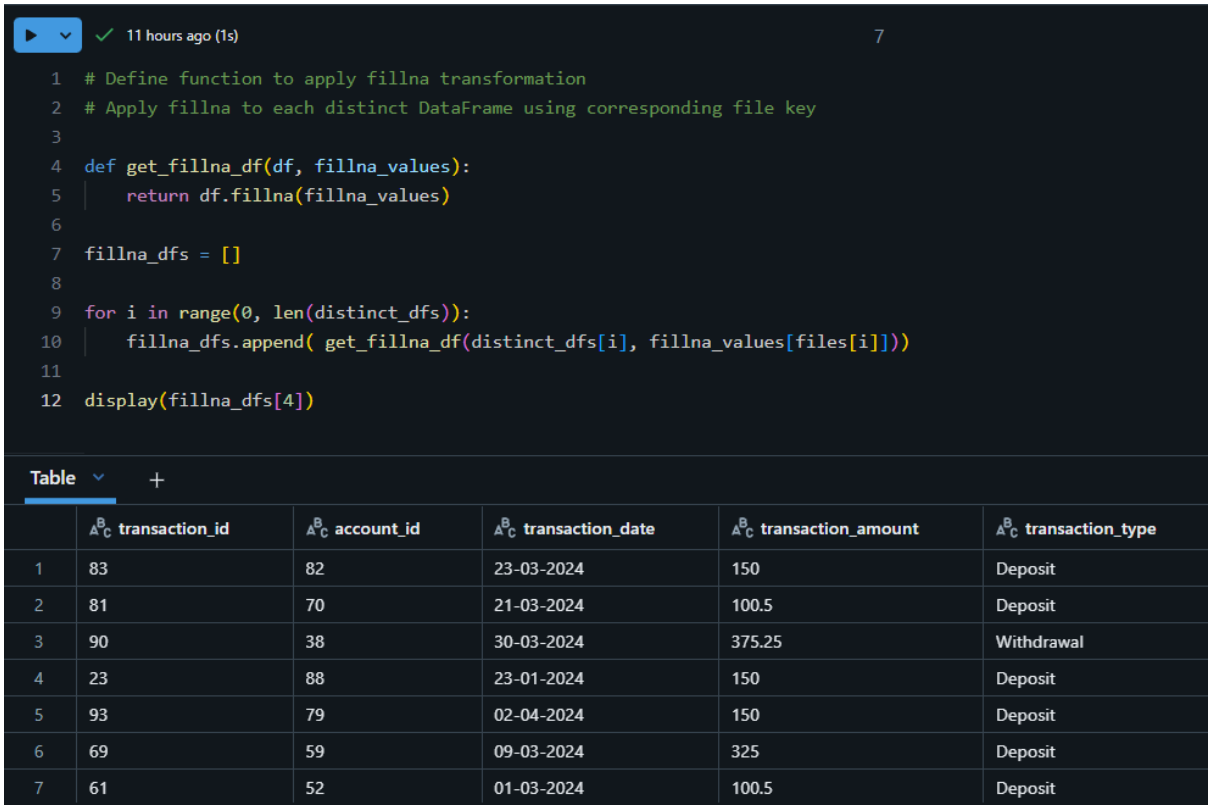
```
fillna_values = {
  "accounts.csv":
  {
    "customer_id" : "-1",
    "account_type": "N/A",
    "balance": "-1.00"
  },

  "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"
    }
}
```

- **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])
```



11 hours ago (1s) 7

```
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])
```

Table +

	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
```

```python
# Import data types and transformation functions
# Cast columns in all dataframes to appropriate types using Spark data types
# Example shown for one dataset (cast_accounts_df)

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)
```

Table +

	transaction_id	account_id	transaction_date	transaction_amount	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
```

```python
# Rename columns for better readability and consistency using CamelCase

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)
```

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

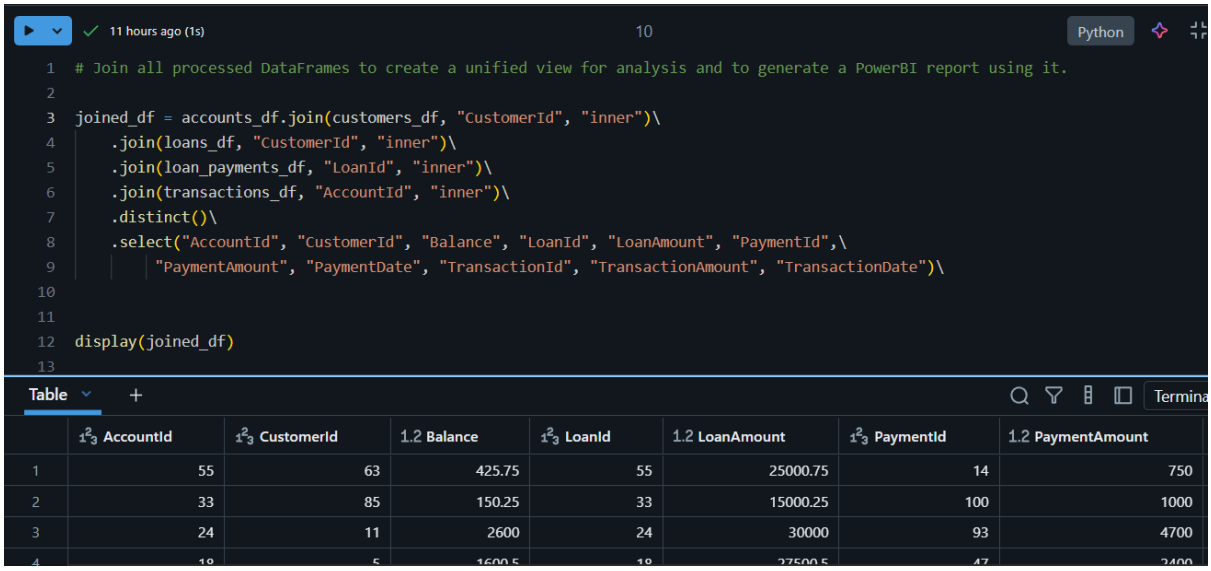
– **Table Joins**

- 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)

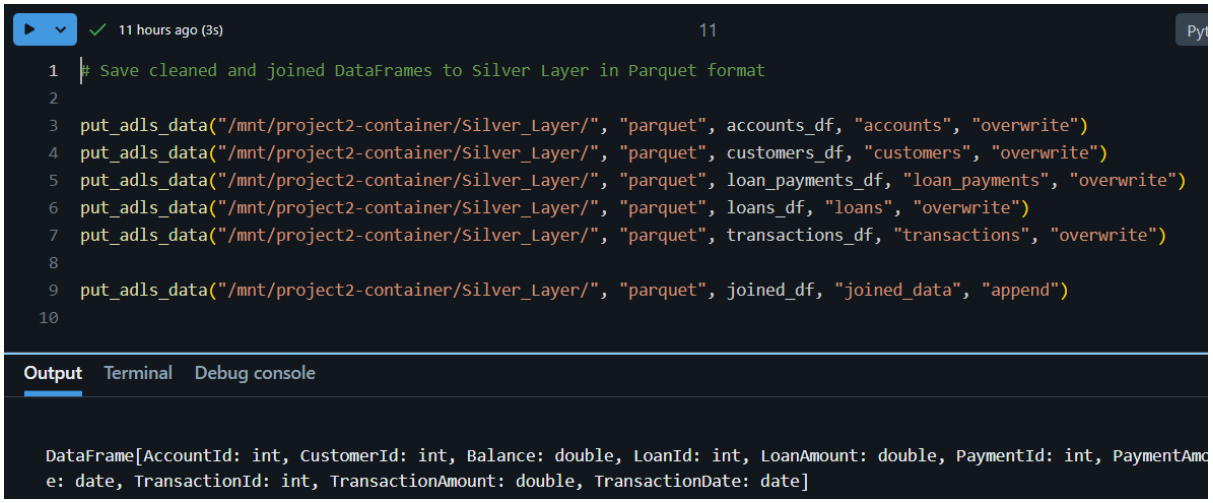


– **Write to Silver Layer**

- 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")


11 hours ago (3s)
11
# Save cleaned and joined DataFrames to Silver Layer in Parquet format
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")
Output
Terminal
Debug console
DataFrame[AccountId: int, CustomerId: int, Balance: double, LoanId: int, LoanAmount: double, PaymentId: int, PaymentAmo
e: date, TransactionId: int, TransactionAmount: double, TransactionDate: date]

Gold Delta Table Creation Notebook

– **Set Catalog**

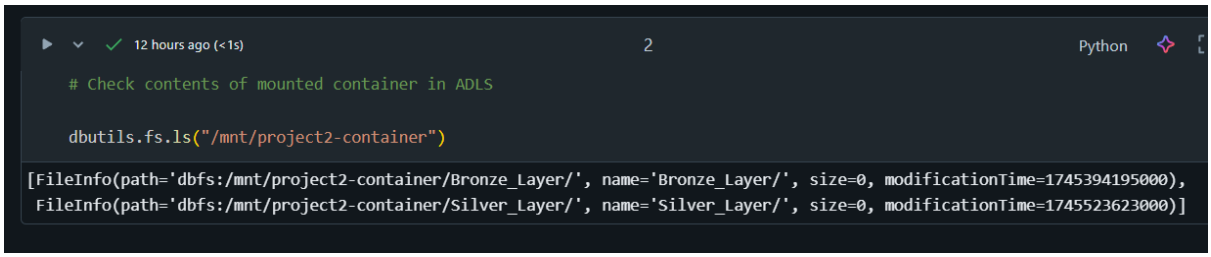
- Sets the SQL context to use the hive_metastore catalog, which is required before creating tables.
- Code:

  %sql
  use catalog hive_metastore

– **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")

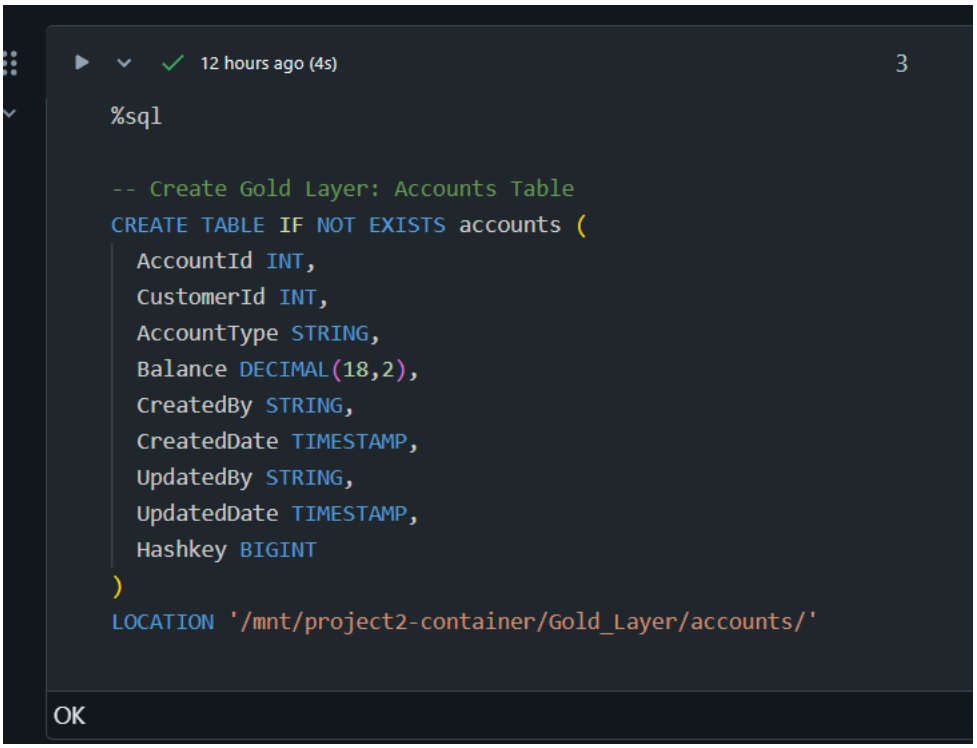


– **Gold Layer Table Creation**

- 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:

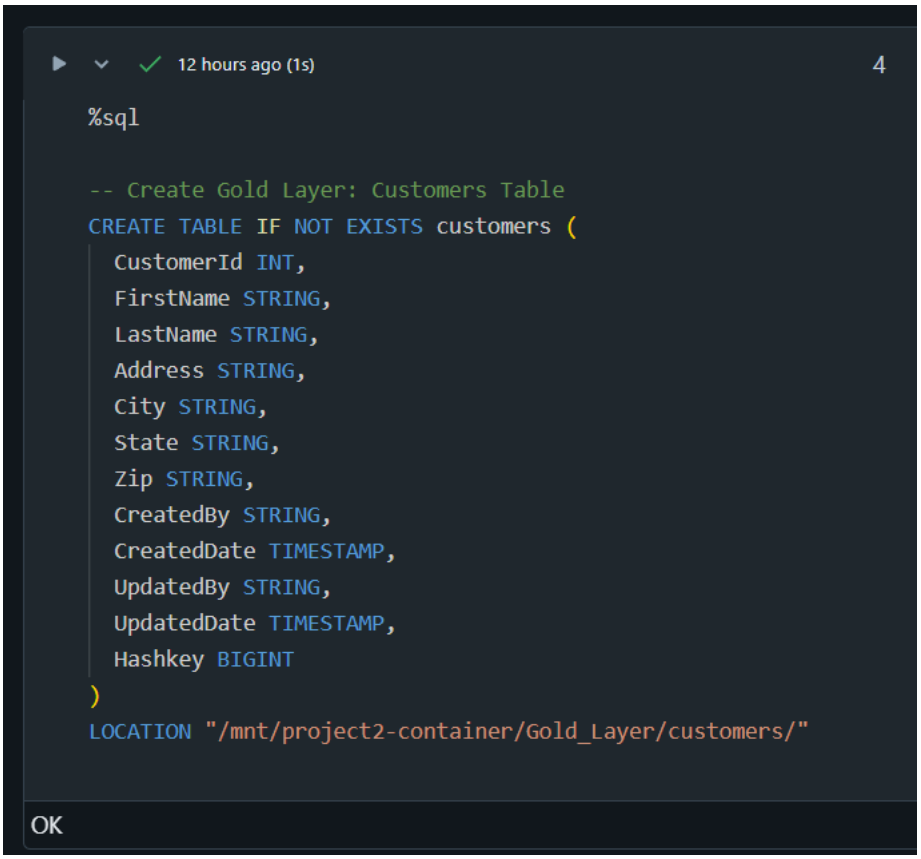
```
%sql
CREATE TABLE IF NOT EXISTS accounts (
 AccountId INT,
 CustomerId INT,
 AccountType STRING,
 Balance DECIMAL(18,2),
 CreatedBy STRING,
 CreatedDate TIMESTAMP,
 UpdatedBy STRING,
 UpdatedDate TIMESTAMP,
 Hashkey BIGINT
)
LOCATION '/mnt/project2-container/Gold_Layer/accounts/'
```



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/"
```



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,
```

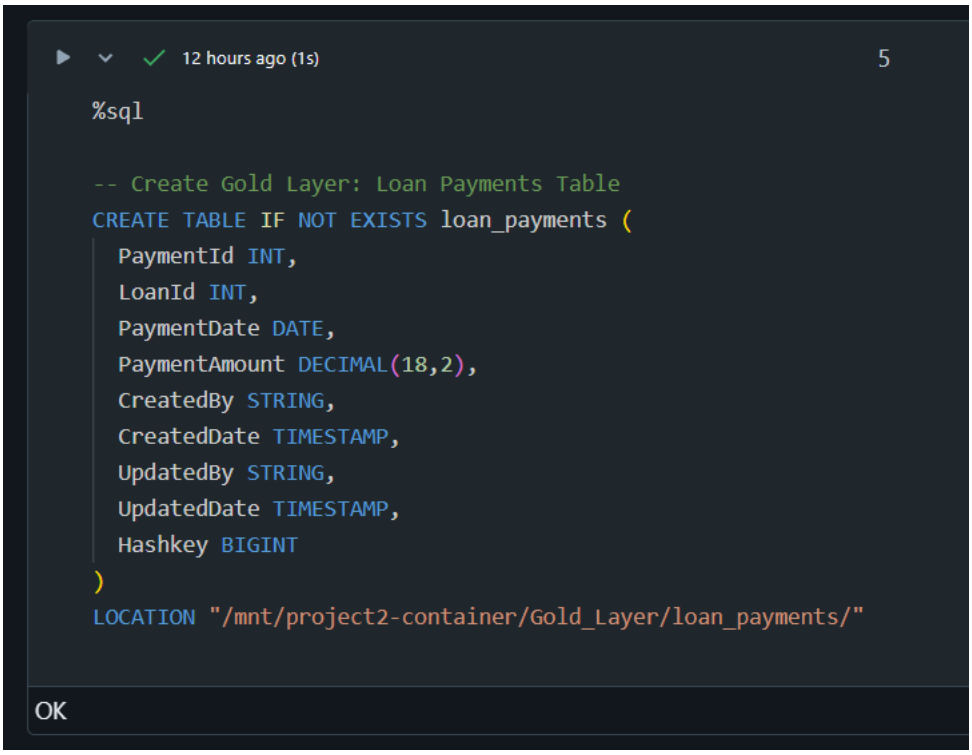
UpdatedBy STRING,

UpdatedDate TIMESTAMP,

Hashkey BIGINT

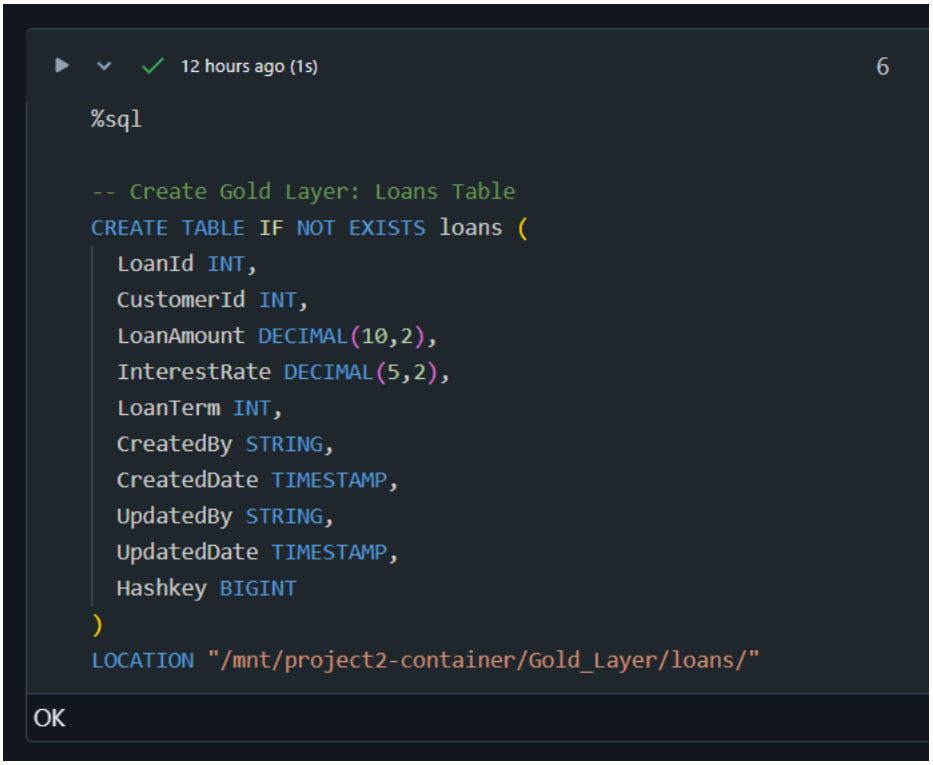
)

LOCATION "/mnt/project2-container/Gold_Layer/loan_payments/"



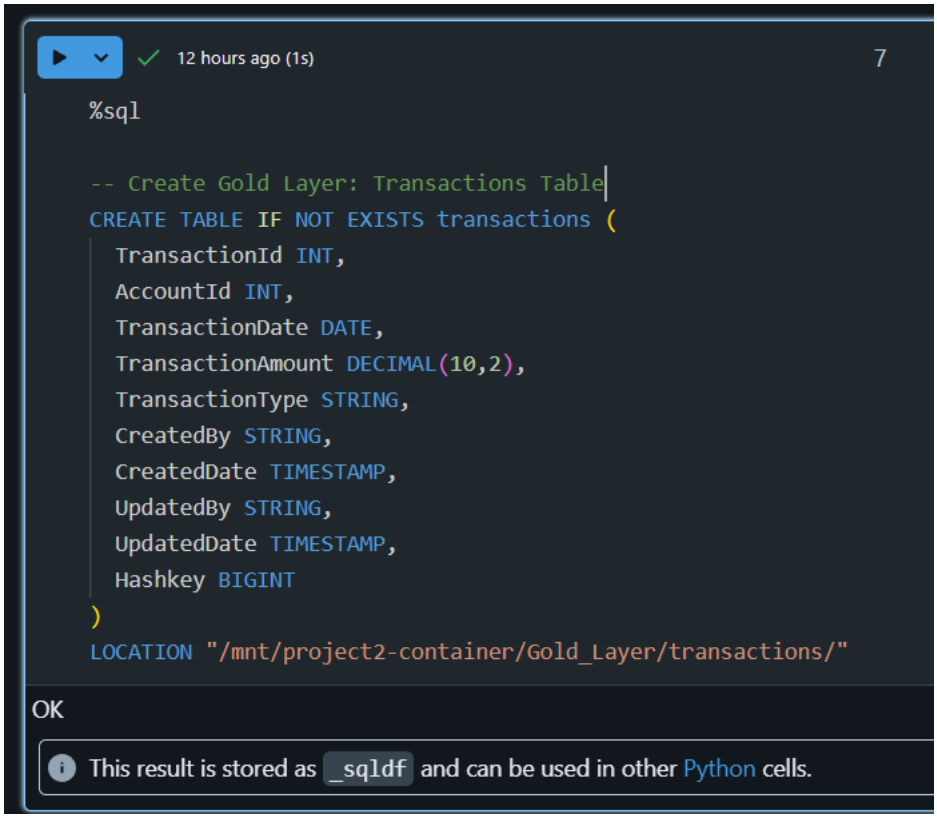
For **loans** table:

%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/"



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"

– **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))
```

– **Gold Delta table creation sql command Inclusion**

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

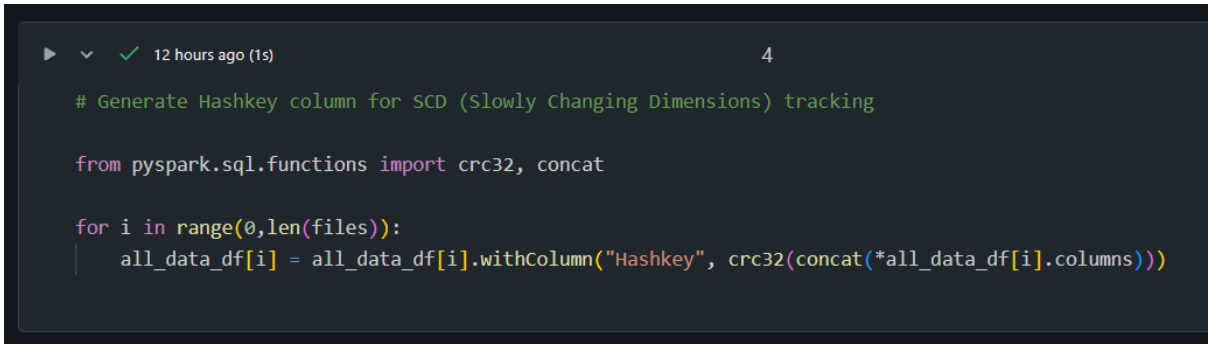
  %run "/Workspace/Project2/GoldDeltaTableCreation"

– **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)))



– **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_tabels = []
delta_table_dfs = []

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

– **Define Update & Insert Logic**

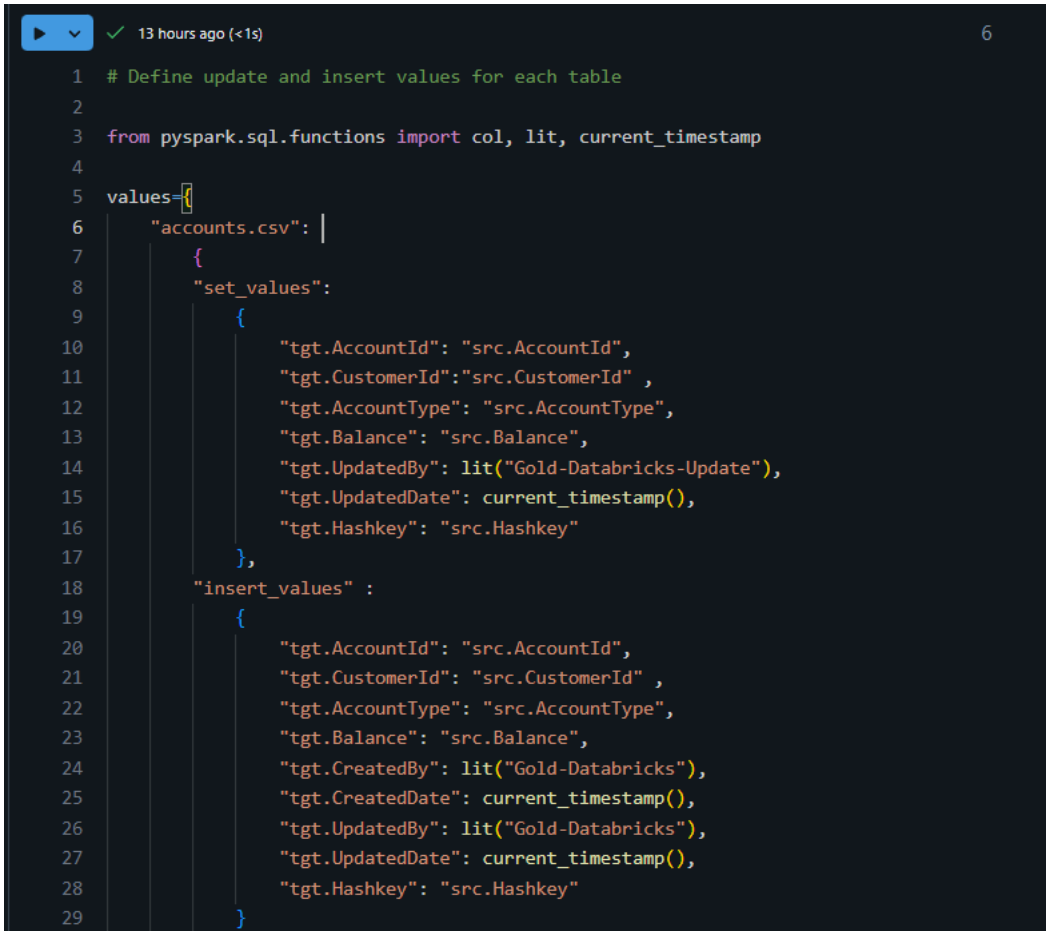
- Stores update and insert column mappings for each table in a dictionary named values.
- Code:

```
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"
    }
  }

}
```



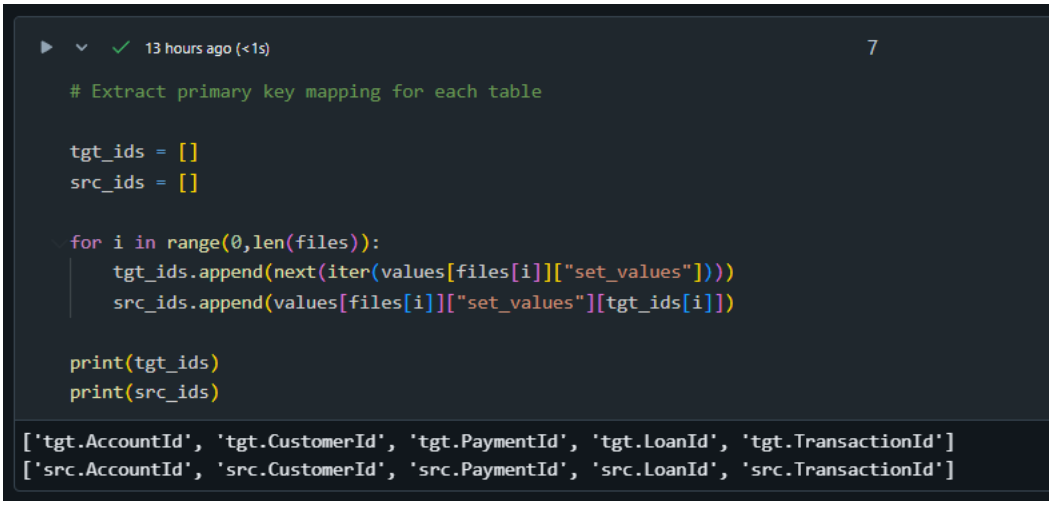
```
    },

    "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.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"
        }
    }

}
```

– **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)
```



– **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.*"))
```

– **Merge into Delta Tables**

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

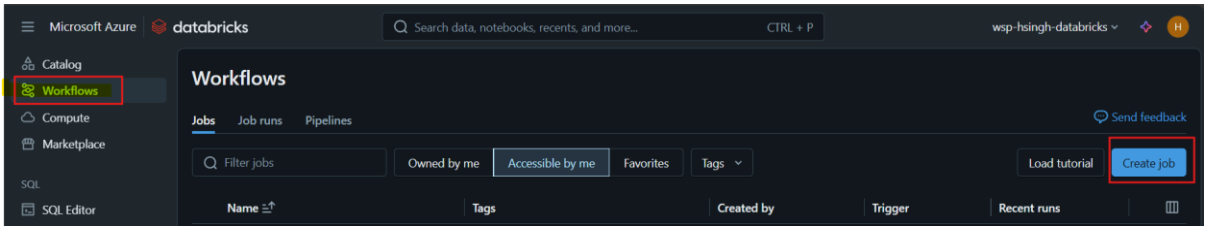
```
for i in range(0,len(files)):
  delta_tabels[i].alias("tgt").merge(src1_dfs[i].alias("src"),((col(tgt_ids[i]) ==
col(src_ids[i])))\
    .whenMatchedUpdate(
      set = values[files[i]]["set_values"]
      )\
      .whenNotMatchedInsert(
        values =  values[files[i]]["insert_values"]
      ).execute()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


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_tabels[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

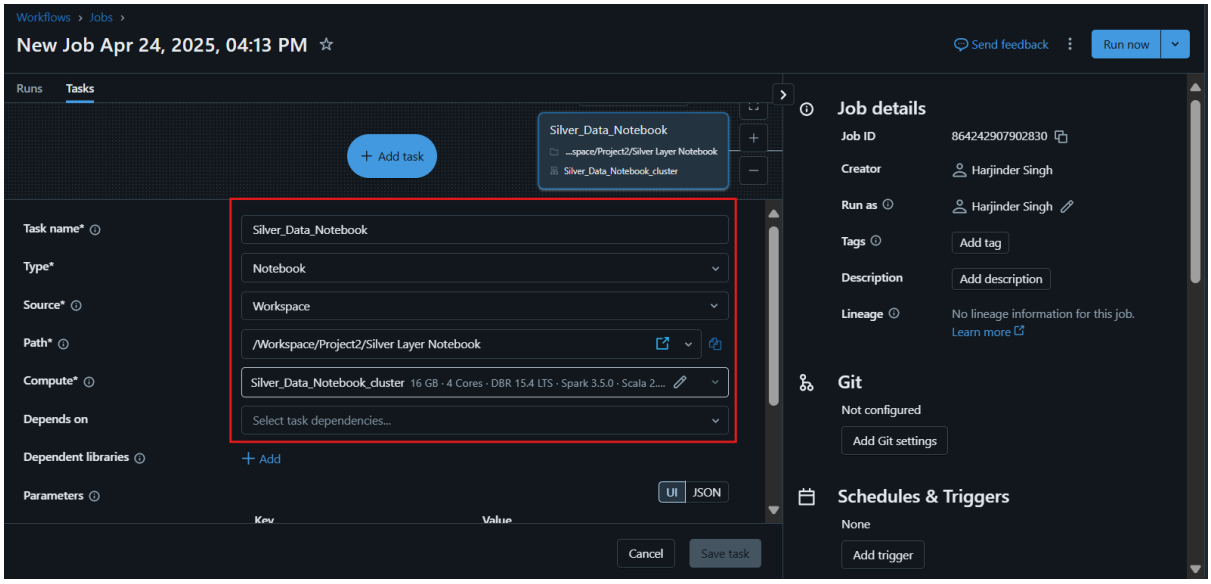
Databricks Workflow Automation

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 sqeuance **every morning at 4AM**.

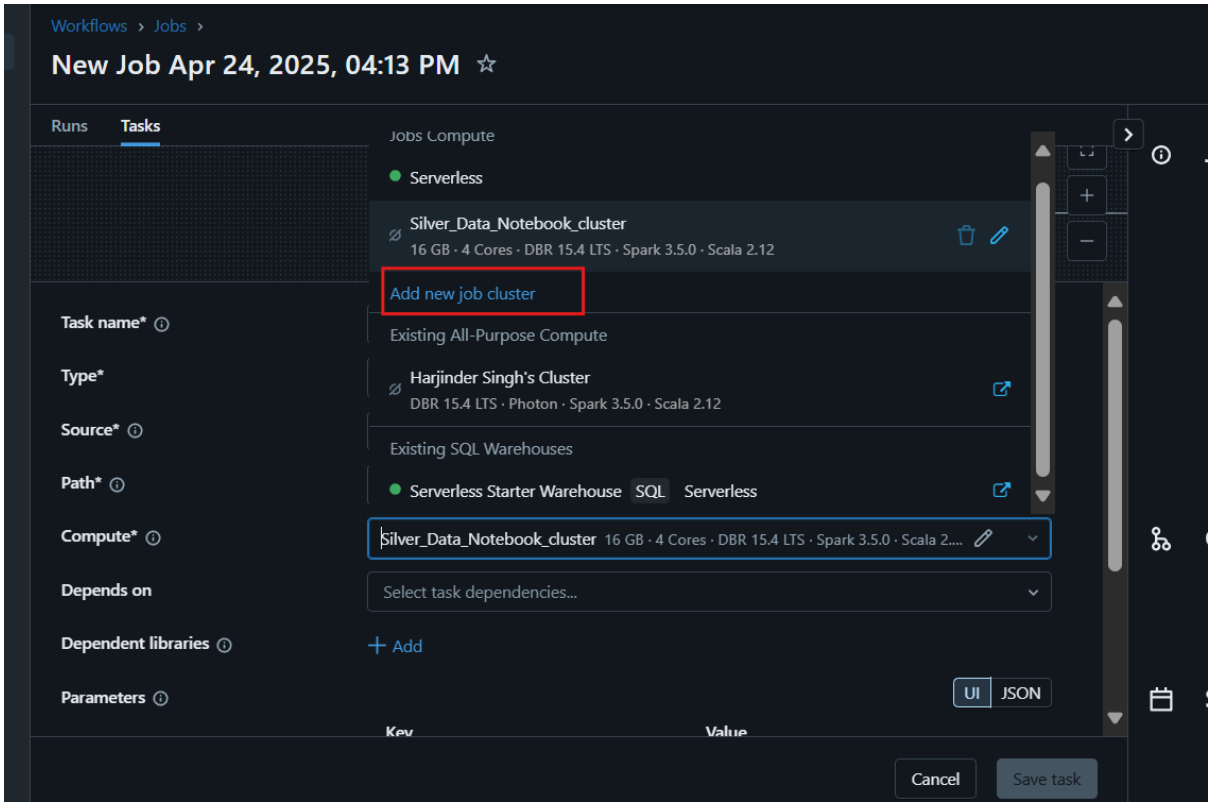
Step 1: Go to Azure Databricks -> click on **Workflows ->** Click on **Create Job.**



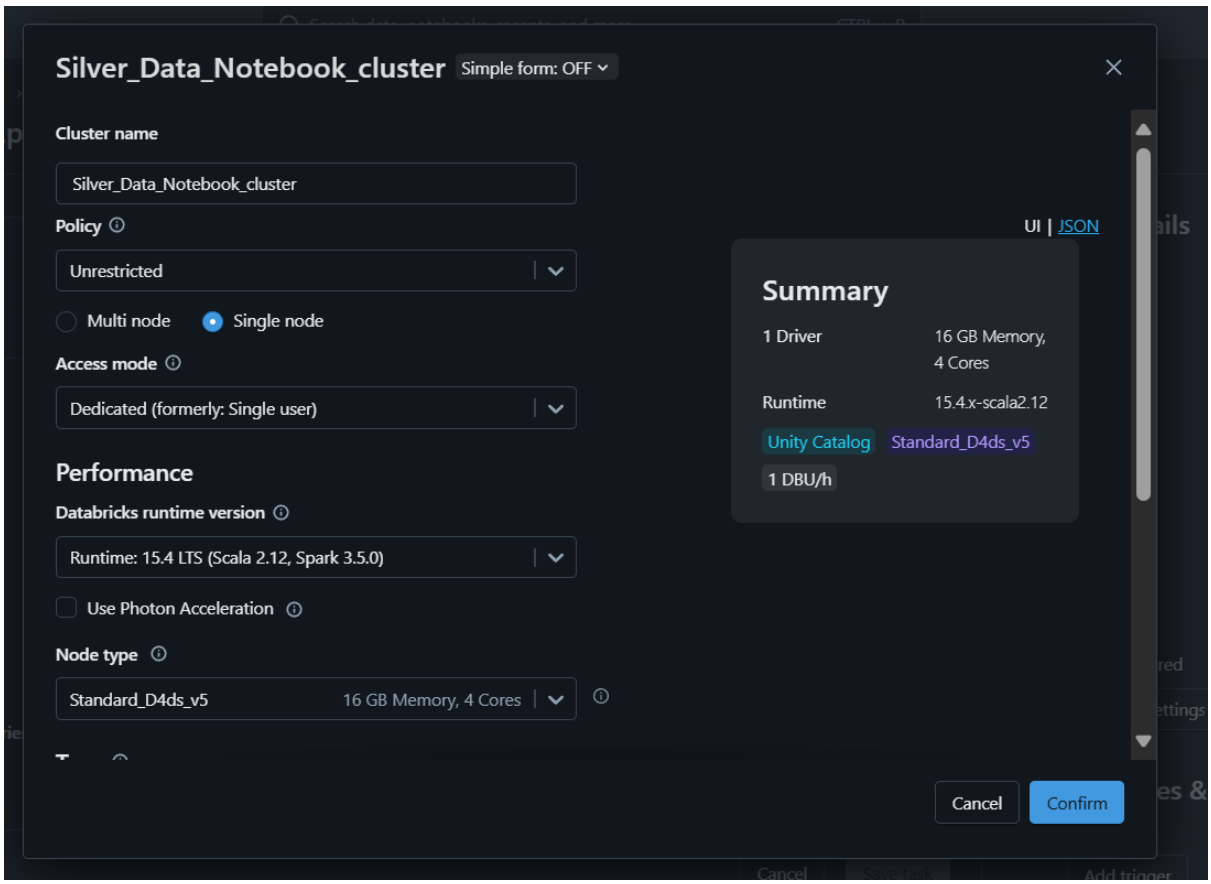
Step 2: Enter the basic details such as task name, source, path etc. -> click on computed to create new job cluster.



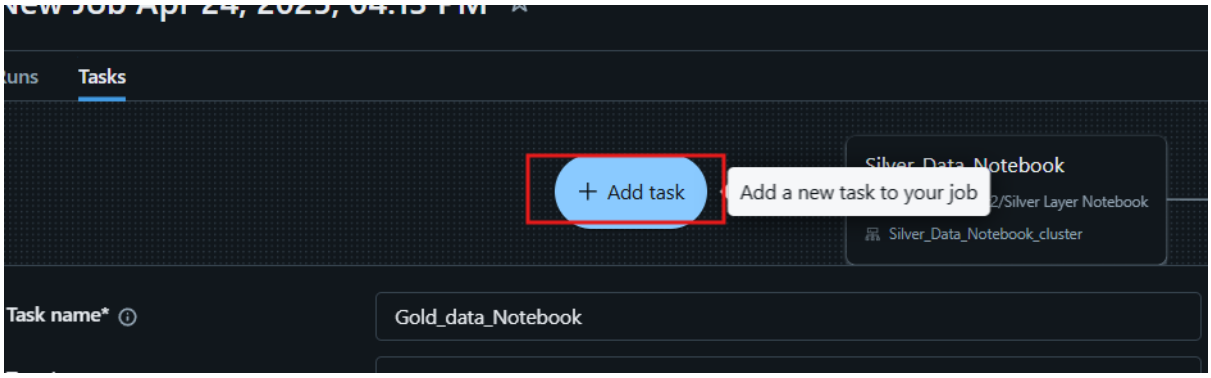
Step 3: In the compute, Click on Add New Job Cluster option.



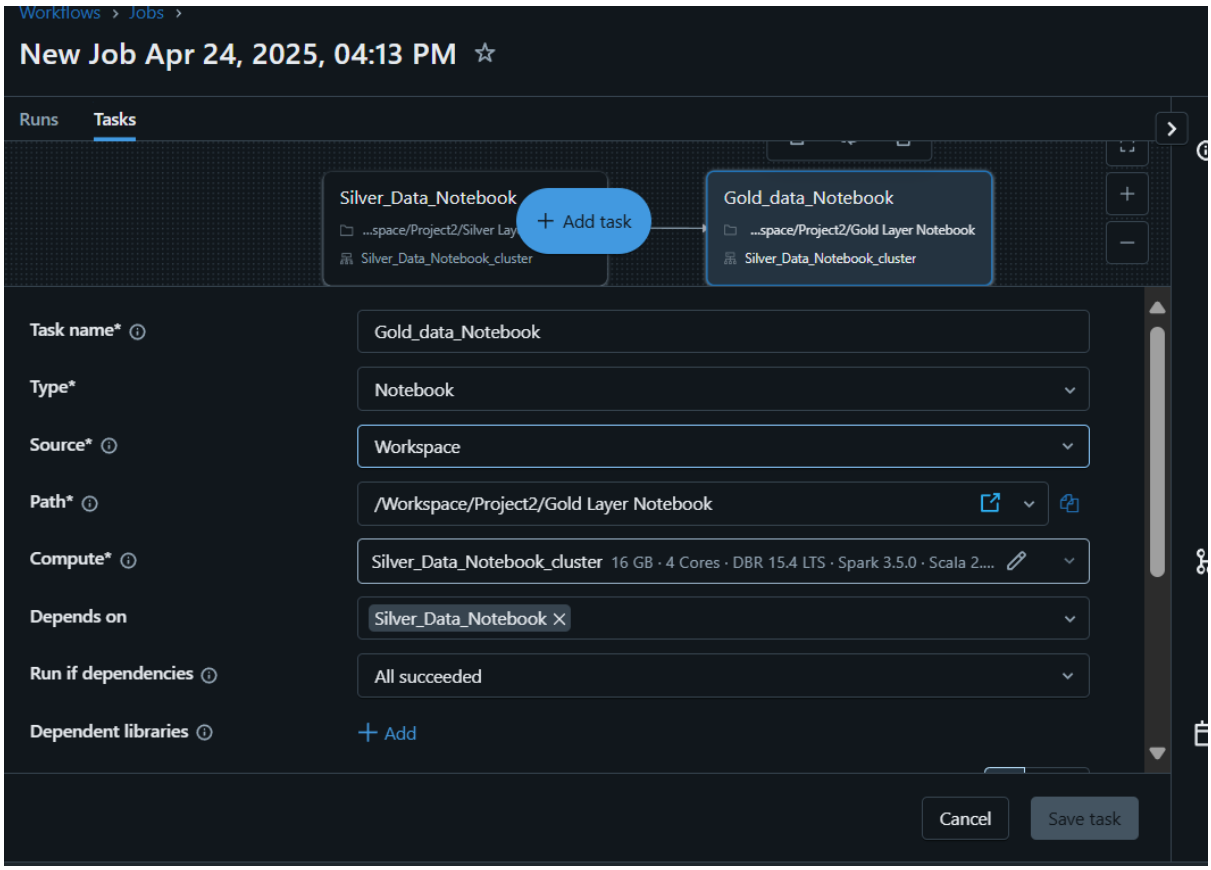
Step 4: Select the single Node and basic node type as shown below.



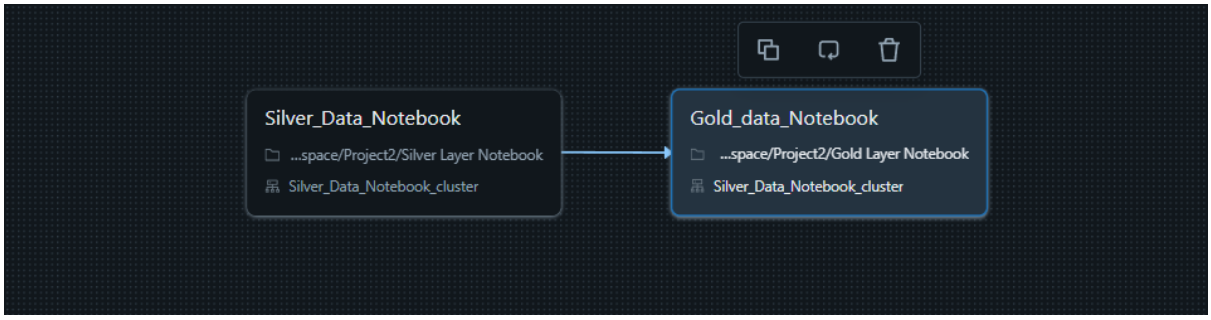
Step 5: Next, Click on Add task to add Glod Layer Notebook as dependent task to Silver Layer Notebook.



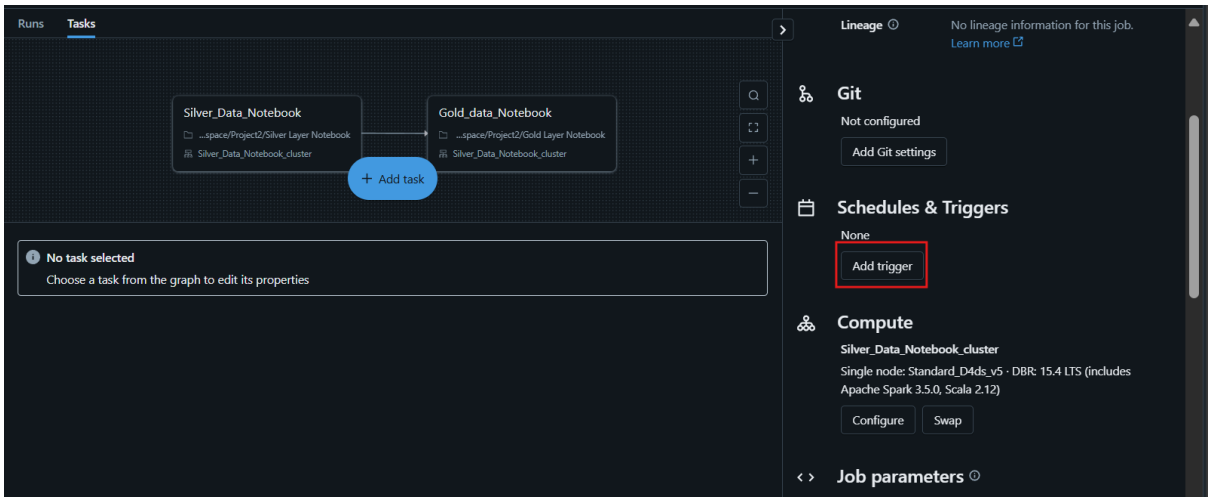
Step 6: Enter basic details such as task name, type etc., and select the previously created compute job cluster. -> click on save task.



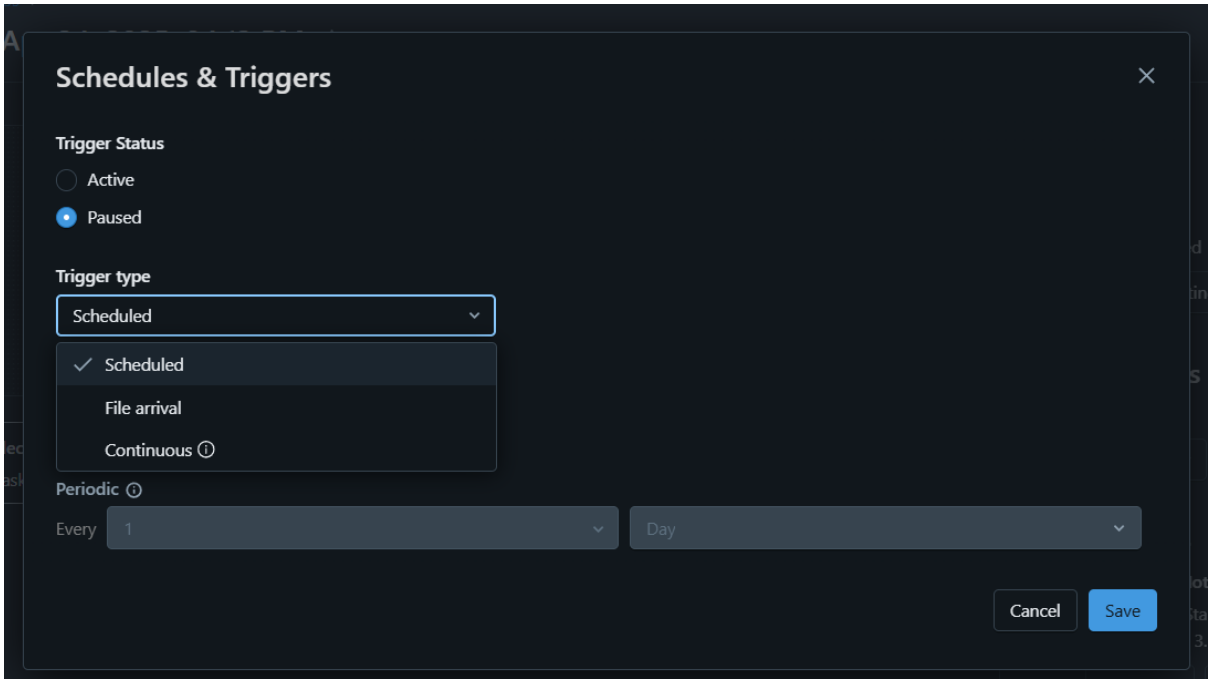
Step 7: Review the workflow tasks.



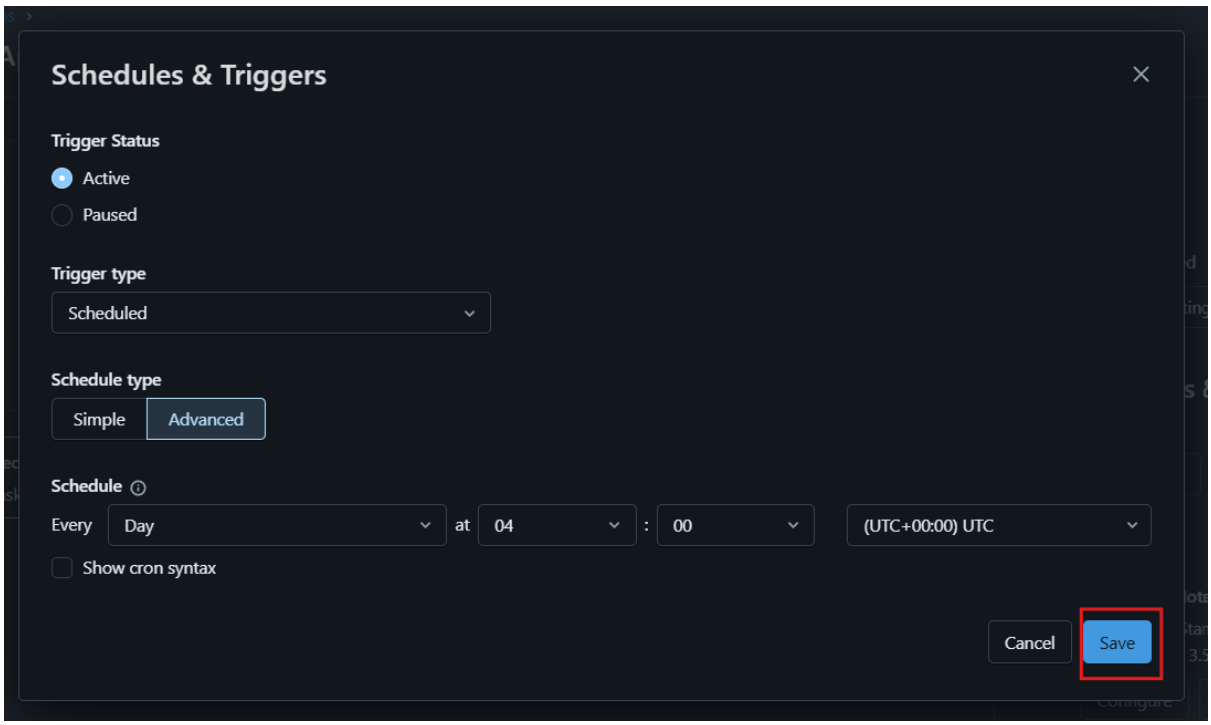
Step 8: To add trigger to automate the workflow execution, click on trigger.



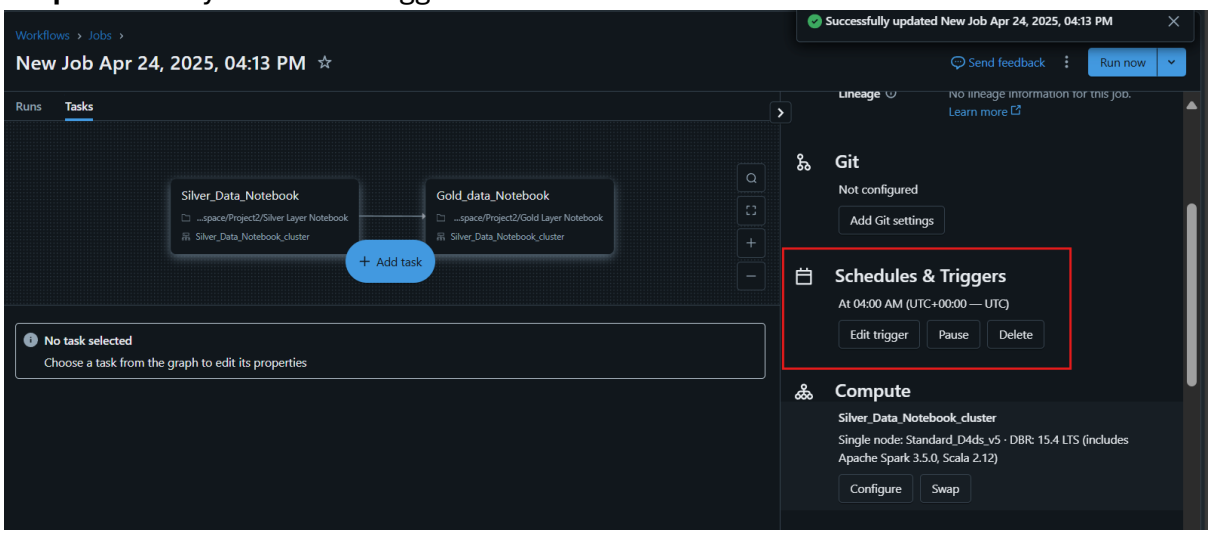
Step 9: select the trigger type as Scheduled.



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.

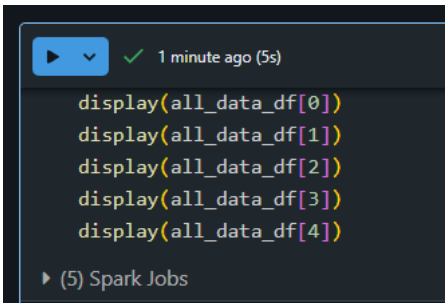


Output Review

✓ **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.



For accounts:

Table +

	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 +

	CustomerId	FirstName	LastName	Address	City	State	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 +

	PaymentId	LoanId	PaymentDate	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 +

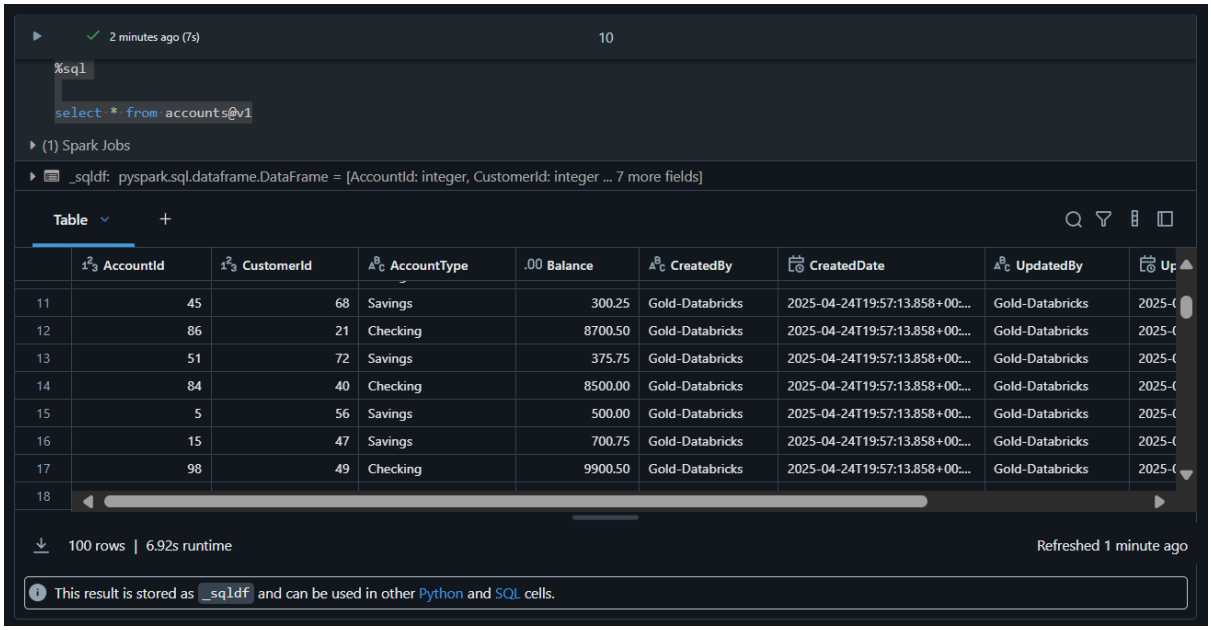
	LoanId	CustomerId	LoanAmount	InterestRate	LoanTerm
1	54	42	30000.5	4	48
2	29	58	32500.25	5	36
3	72	17	20000	3	24
4	88	1	27500	3	24
5	78	4	27500.5	4	48
6	18	5	27500.5	4.5	48
7	70	33	37500.5	4	48
8	17	99	22500.25	5.5	36
9	46	24	17500.5	4	48
10	8	67	27500	3	24
11	79	55	32500.75	6.5	60
12	23	88	15000.75	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

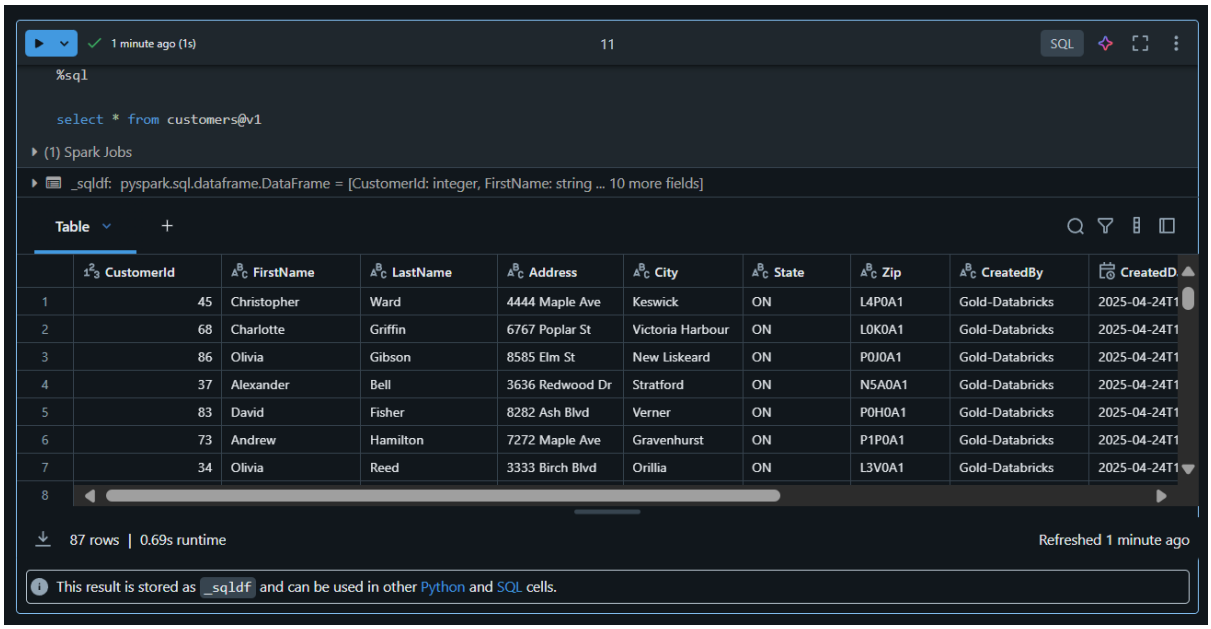
For transactions:

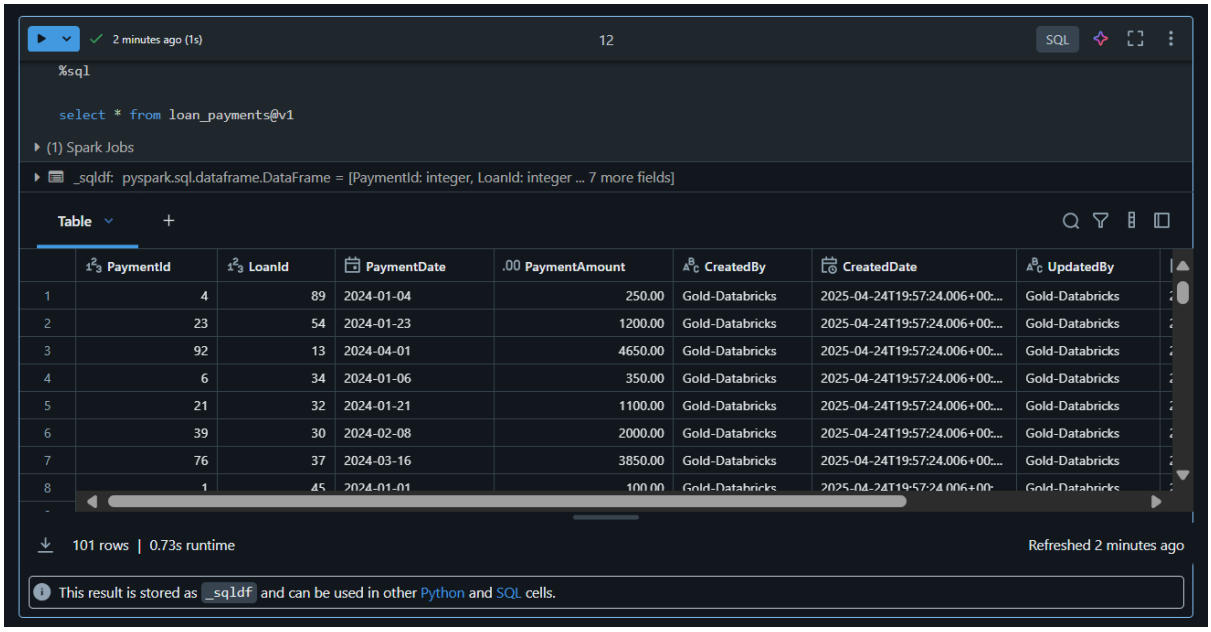
Table +

	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
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
11 | 45 | 68 | Savings | 300.25 | Gold-Databricks | 2025-04-24T19:57:13.858+00:... | Gold-Databricks
12 | 86 | 21 | Checking | 8700.50 | Gold-Databricks | 2025-04-24T19:57:13.858+00:... | Gold-Databricks
13 | 51 | 72 | Savings | 375.75 | Gold-Databricks | 2025-04-24T19:57:13.858+00:... | Gold-Databricks
14 | 84 | 40 | Checking | 8500.00 | Gold-Databricks | 2025-04-24T19:57:13.858+00:... | Gold-Databricks
15 | 5 | 56 | Savings | 500.00 | Gold-Databricks | 2025-04-24T19:57:13.858+00:... | Gold-Databricks
16 | 15 | 47 | Savings | 700.75 | Gold-Databricks | 2025-04-24T19:57:13.858+00:... | Gold-Databricks
17 | 98 | 49 | Checking | 9900.50 | Gold-Databricks | 2025-04-24T19:57:13.858+00:... | Gold-Databricks
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
%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 | 2025-04-24T1
2 | 68 | Charlotte | Griffin | 6767 Poplar St | Victoria Harbour | ON | L0K0A1 | Gold-Databricks | 2025-04-24T1
3 | 86 | Olivia | Gibson | 8585 Elm St | New Liskeard | ON | P0J0A1 | Gold-Databricks | 2025-04-24T1
4 | 37 | Alexander | Bell | 3636 Redwood Dr | Stratford | ON | N5A0A1 | Gold-Databricks | 2025-04-24T1
5 | 83 | David | Fisher | 8282 Ash Blvd | Verner | ON | P0H0A1 | Gold-Databricks | 2025-04-24T1
6 | 73 | Andrew | Hamilton | 7272 Maple Ave | Gravenhurst | ON | P1P0A1 | Gold-Databricks | 2025-04-24T1
7 | 34 | Olivia | Reed | 3333 Birch Blvd | Orillia | ON | L3V0A1 | Gold-Databricks | 2025-04-24T1
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

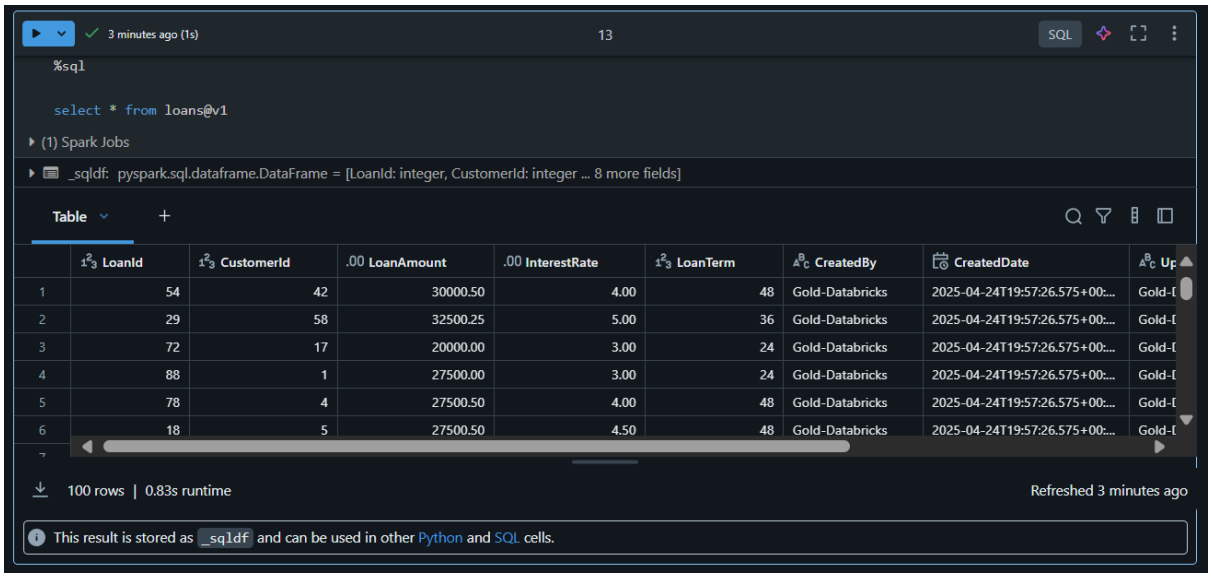
_sqldf: pyspark.sql.dataframe.DataFrame = [PaymentId: integer, LoanId: integer ... 7 more fields]

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

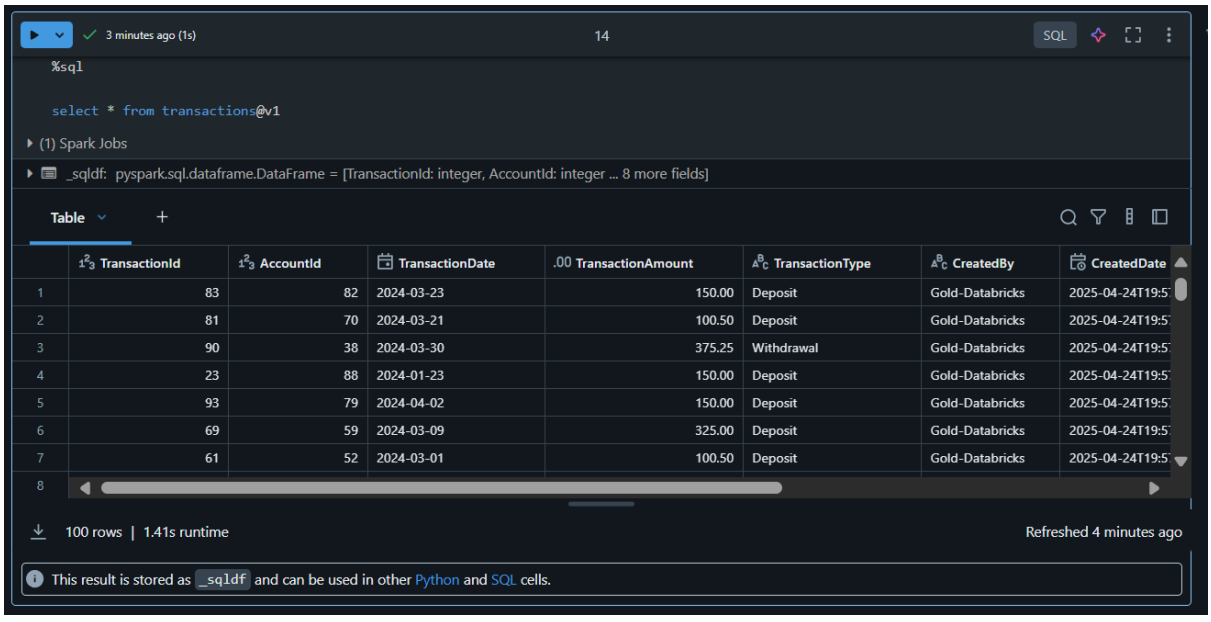
_sqldf: pyspark.sql.dataframe.DataFrame = [LoanId: integer, CustomerId: integer ... 8 more fields]

Table +

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

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	82	2024-03-23	150.00	Deposit	Gold-Databricks	2025-04-24T19:5
2	81	70	2024-03-21	100.50	Deposit	Gold-Databricks	2025-04-24T19:5
3	90	38	2024-03-30	375.25	Withdrawal	Gold-Databricks	2025-04-24T19:5
4	23	88	2024-01-23	150.00	Deposit	Gold-Databricks	2025-04-24T19:5
5	93	79	2024-04-02	150.00	Deposit	Gold-Databricks	2025-04-24T19:5
6	69	59	2024-03-09	325.00	Deposit	Gold-Databricks	2025-04-24T19:5
7	61	52	2024-03-01	100.50	Deposit	Gold-Databricks	2025-04-24T19:5
8							

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		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:

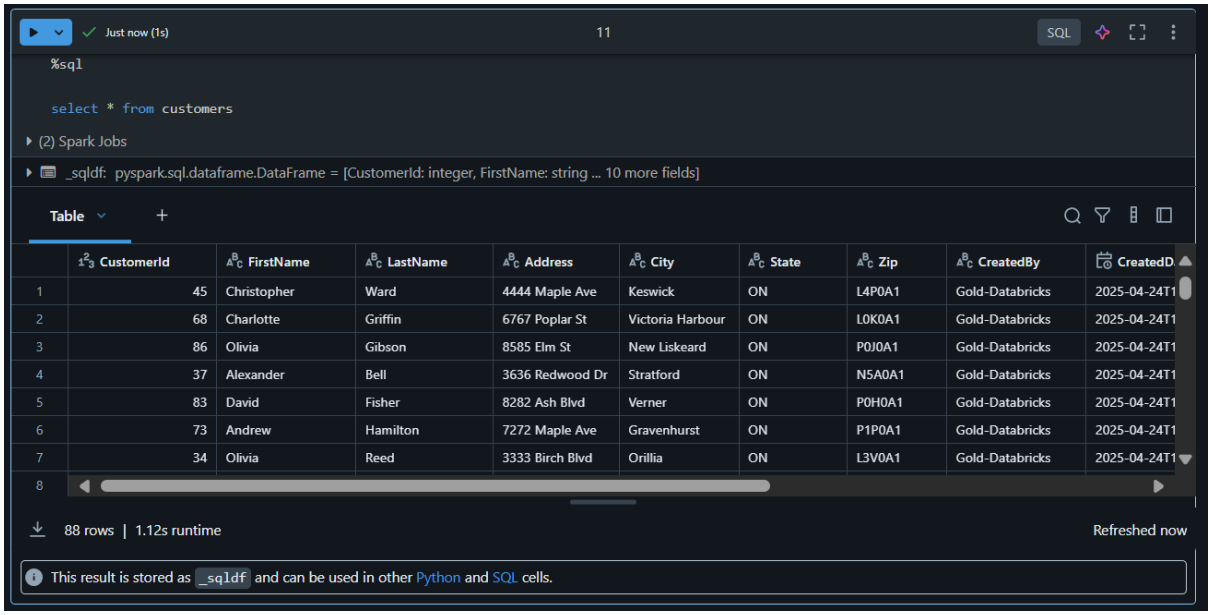
95	94	39	03-04-2024	300.25	Withdrawal
96	95	60	04-04-2024	250	Deposit
97	96	48	05-04-2024	175	Withdrawal
98	97	90	06-04-2024	225.5	Deposit
99	98	49	07-04-2024	275.75	Withdrawal
100	99	80	08-04-2024	325	Deposit
101	100	50	09-04-2024	510.25	Deposit
102					

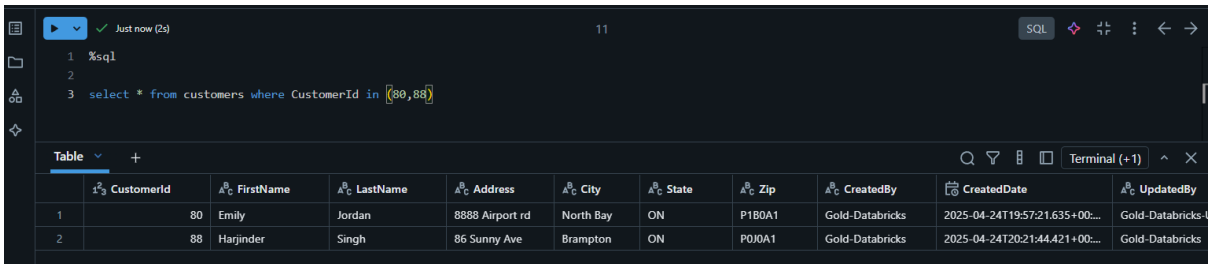
transactions data after changes at Bronze:

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					

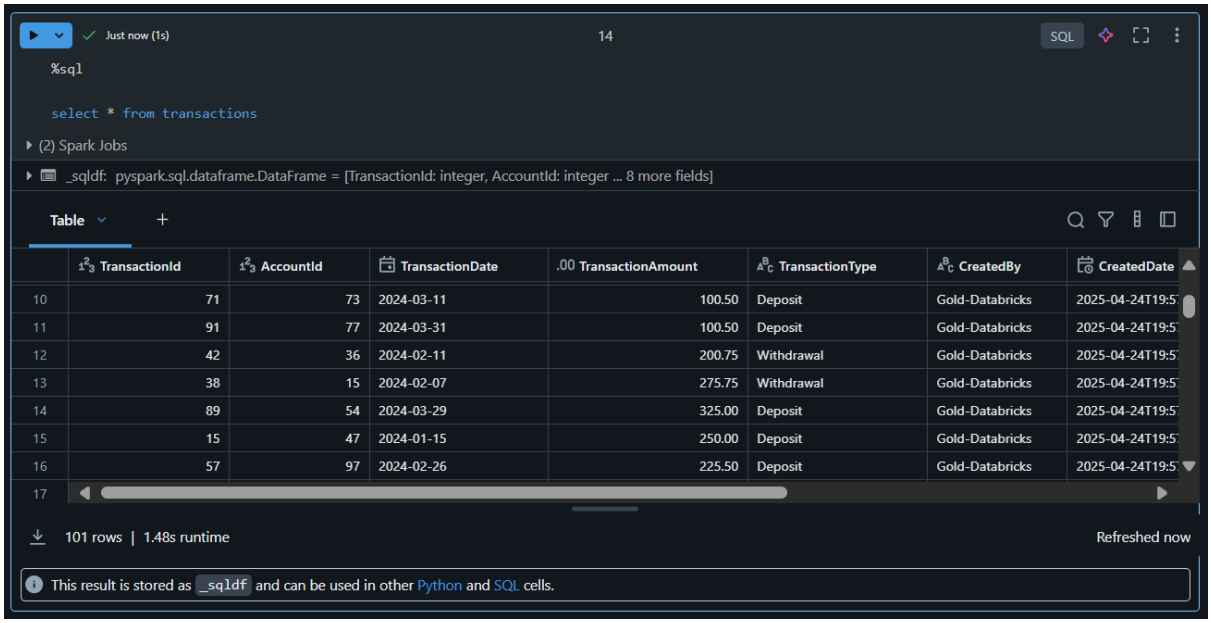
– Result of Gold Layer Notebook After Execution.

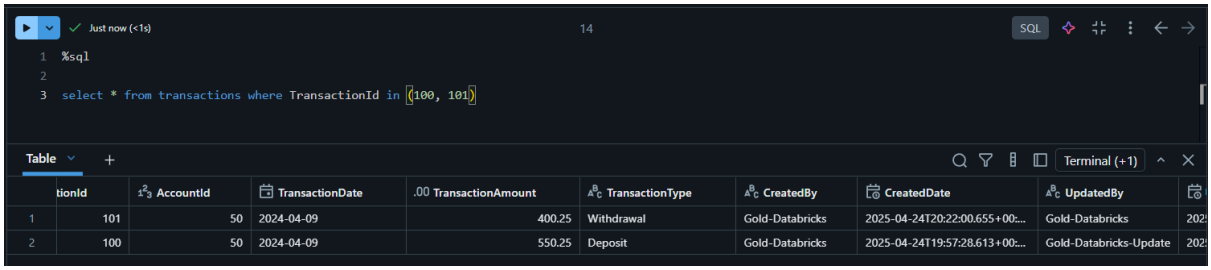
For customers:





For customers:

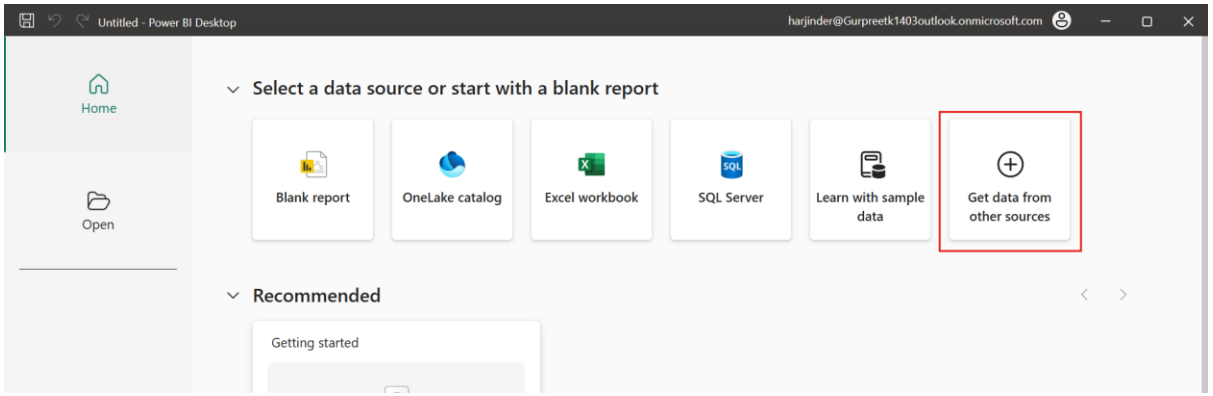

Just now (1s)
14
SQL
%sql
select * from transactions
(2) Spark Jobs
_sqldf: pyspark.sql.dataframe.DataFrame = [TransactionId: integer, AccountId: integer ... 8 more fields]
Table
TransactionId AccountId TransactionDate TransactionAmount TransactionType CreatedBy CreatedDate
10 71 73 2024-03-11 100.50 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
17
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
Terminal (+1)
tionId AccountId TransactionDate TransactionAmount TransactionType CreatedBy CreatedDate UpdatedBy
1 101 50 2024-04-09 400.25 Withdrawal Gold-Databricks 2025-04-24T20:22:00.655+00:... Gold-Databricks
2 100 50 2024-04-09 550.25 Deposit Gold-Databricks 2025-04-24T19:57:28.613+00:... Gold-Databricks-Update

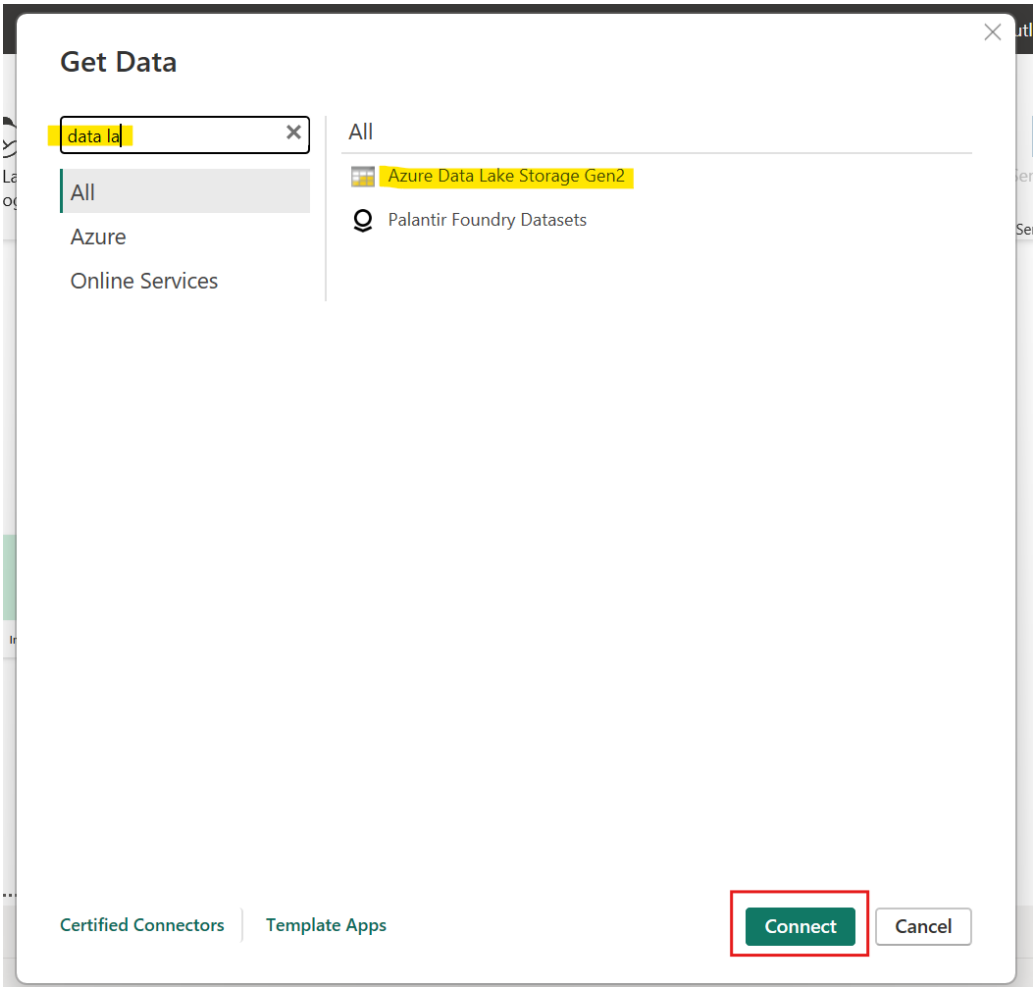
Power BI Report

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).

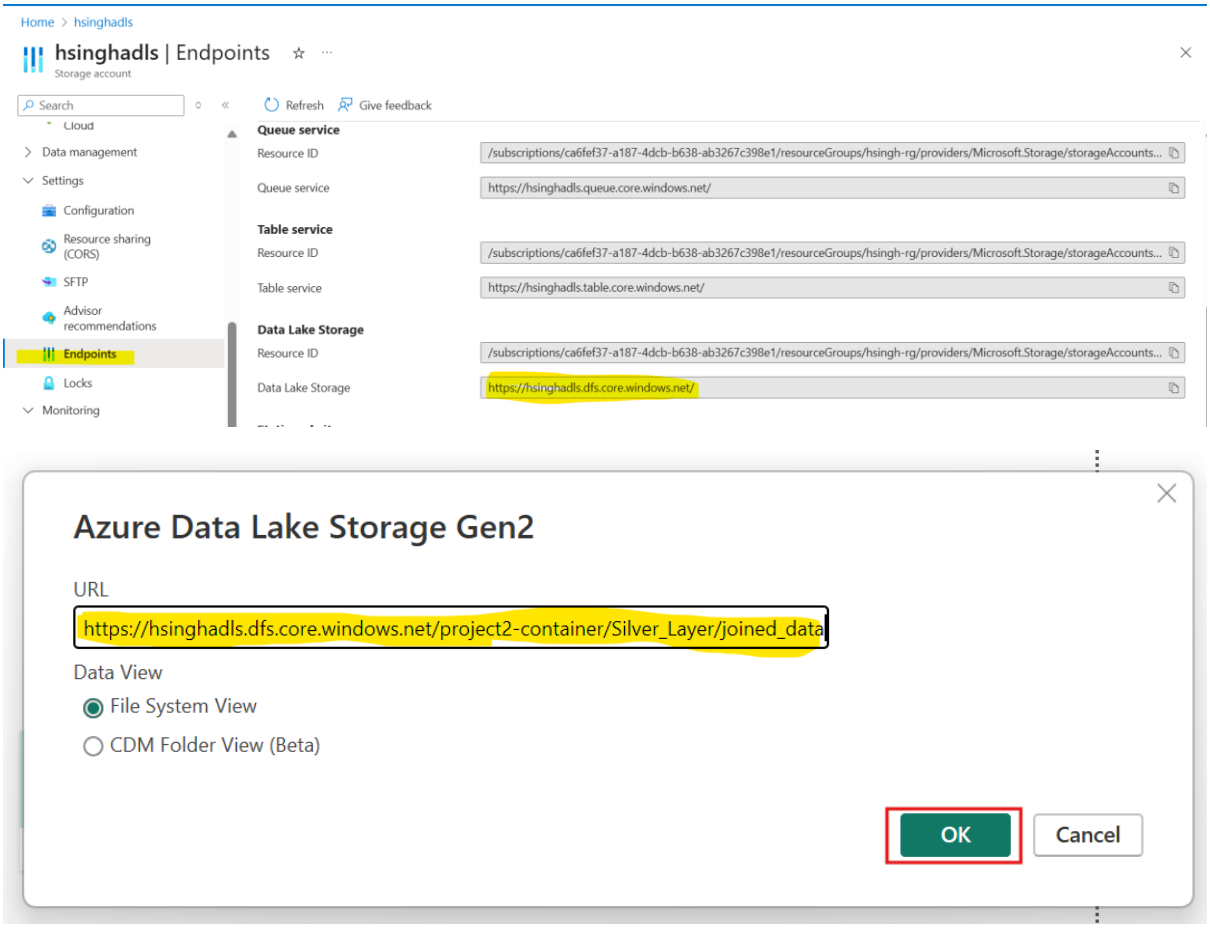
Next, select Get Data From other source as data Source.



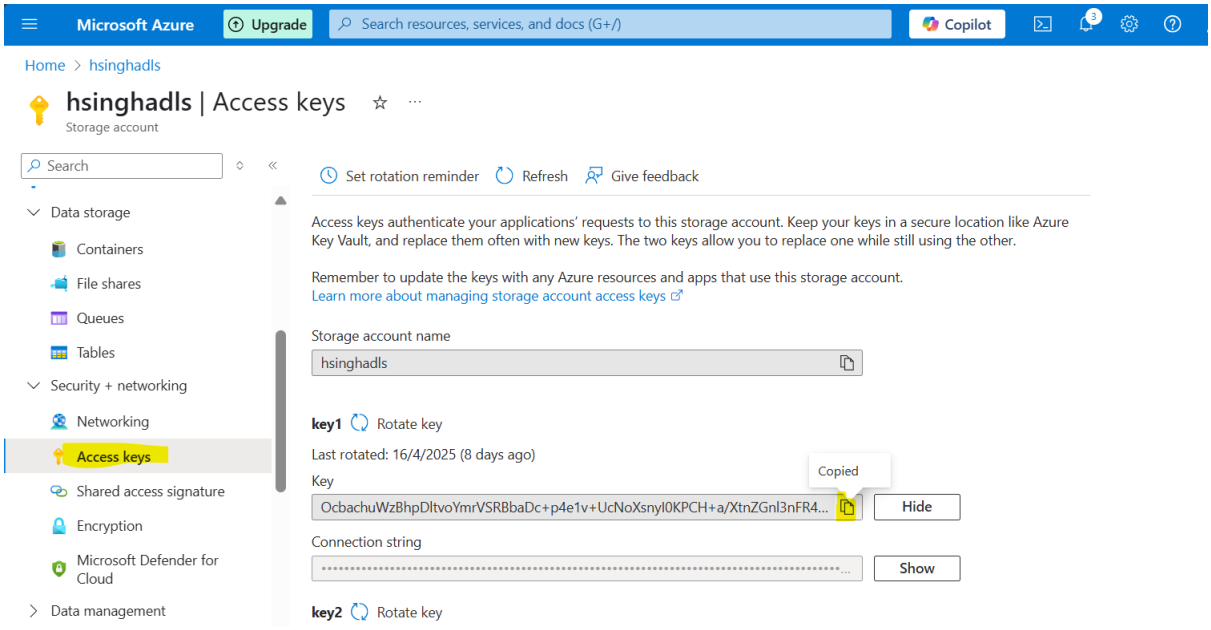
– Search and select Azure Data Lake Storage Gen2 option -> click on Connect.



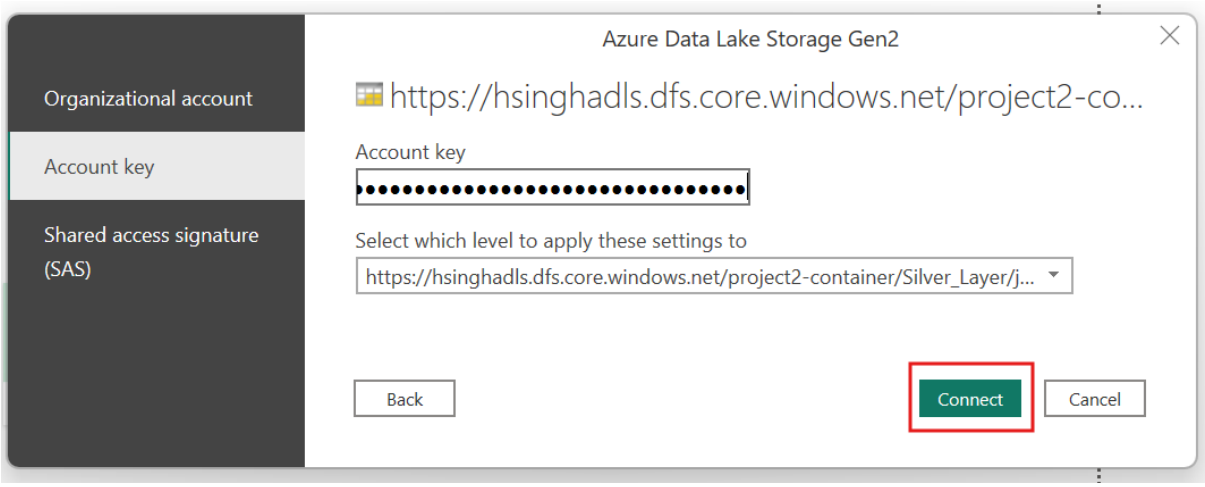
Step 2: Next, enter the URL of Azure Data Lake storage Gen2 account with folder path (find the path in Endpoint tab of Setting in ADLS in DataLake storage section)-> select File System View as Data View. -> click on Ok.



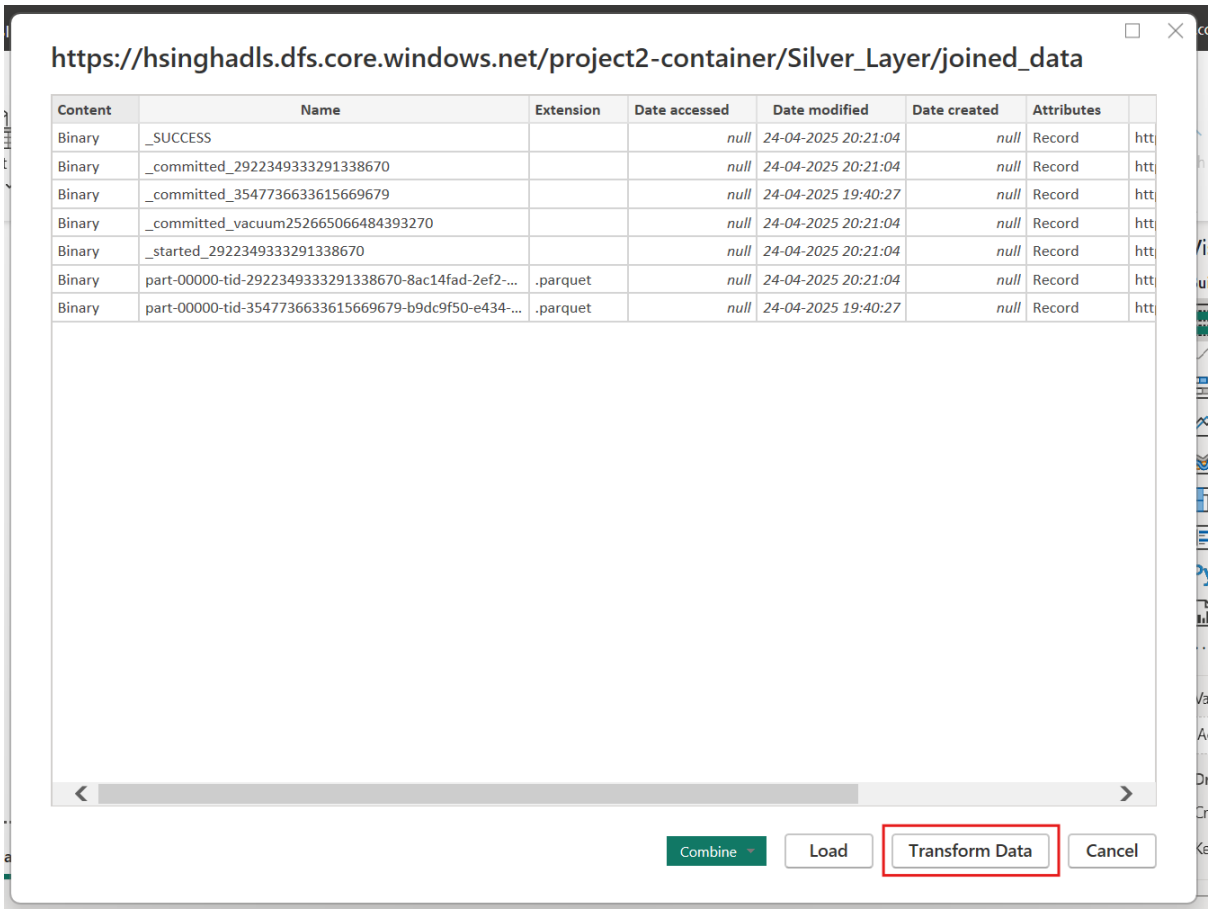
– Go to ADLS -> Access Key tab -> copy key1.



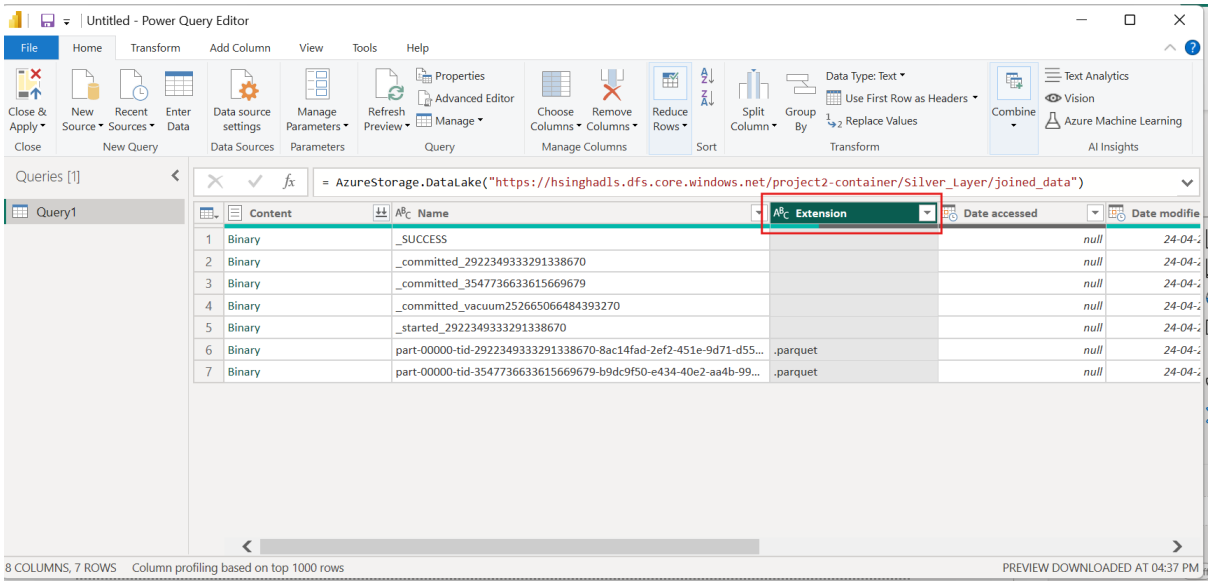
– Paste the copies key1 in the Account Key section -> click on connect.

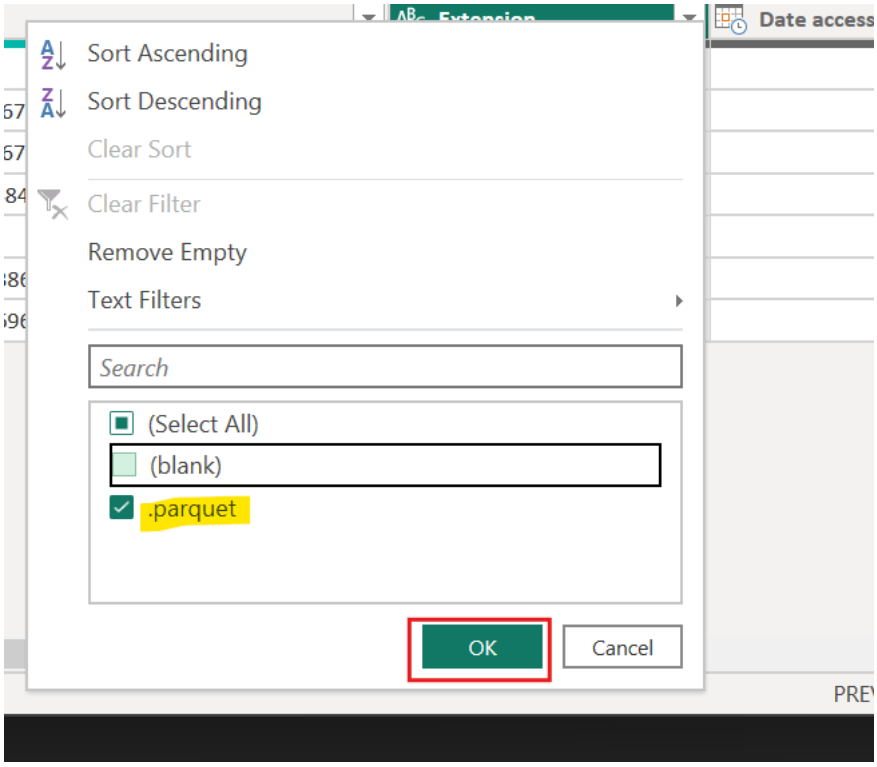


Step 3: Review the files from selected path -> click on Tranform data.

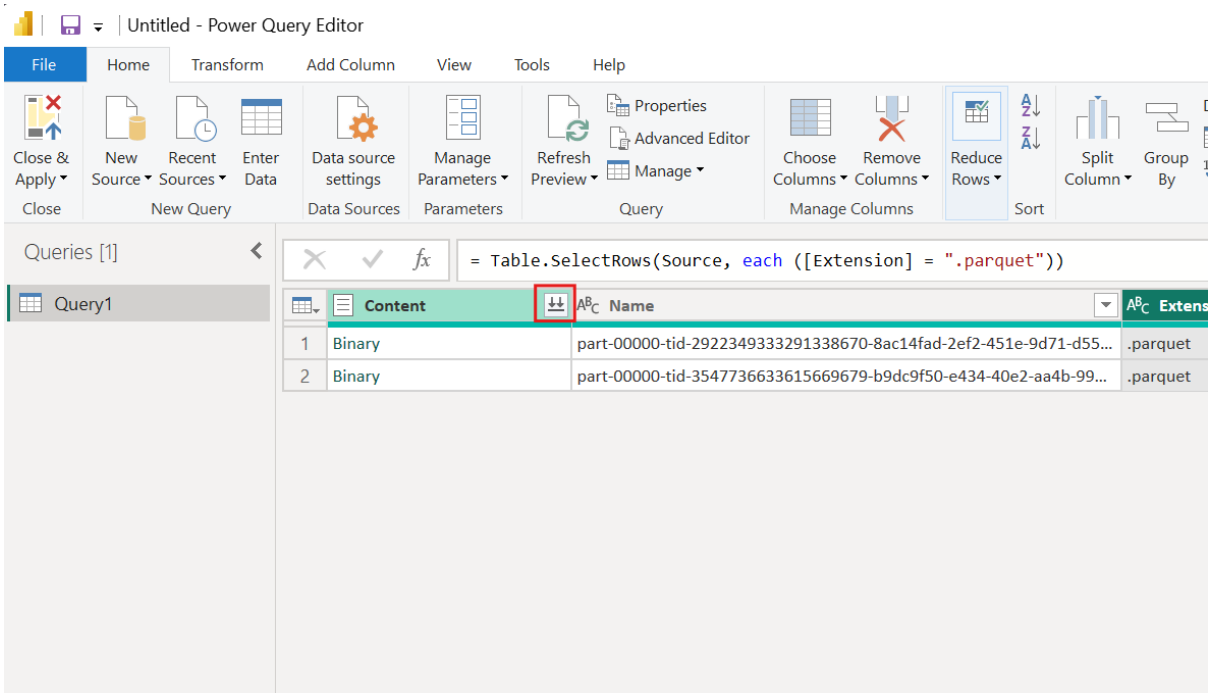


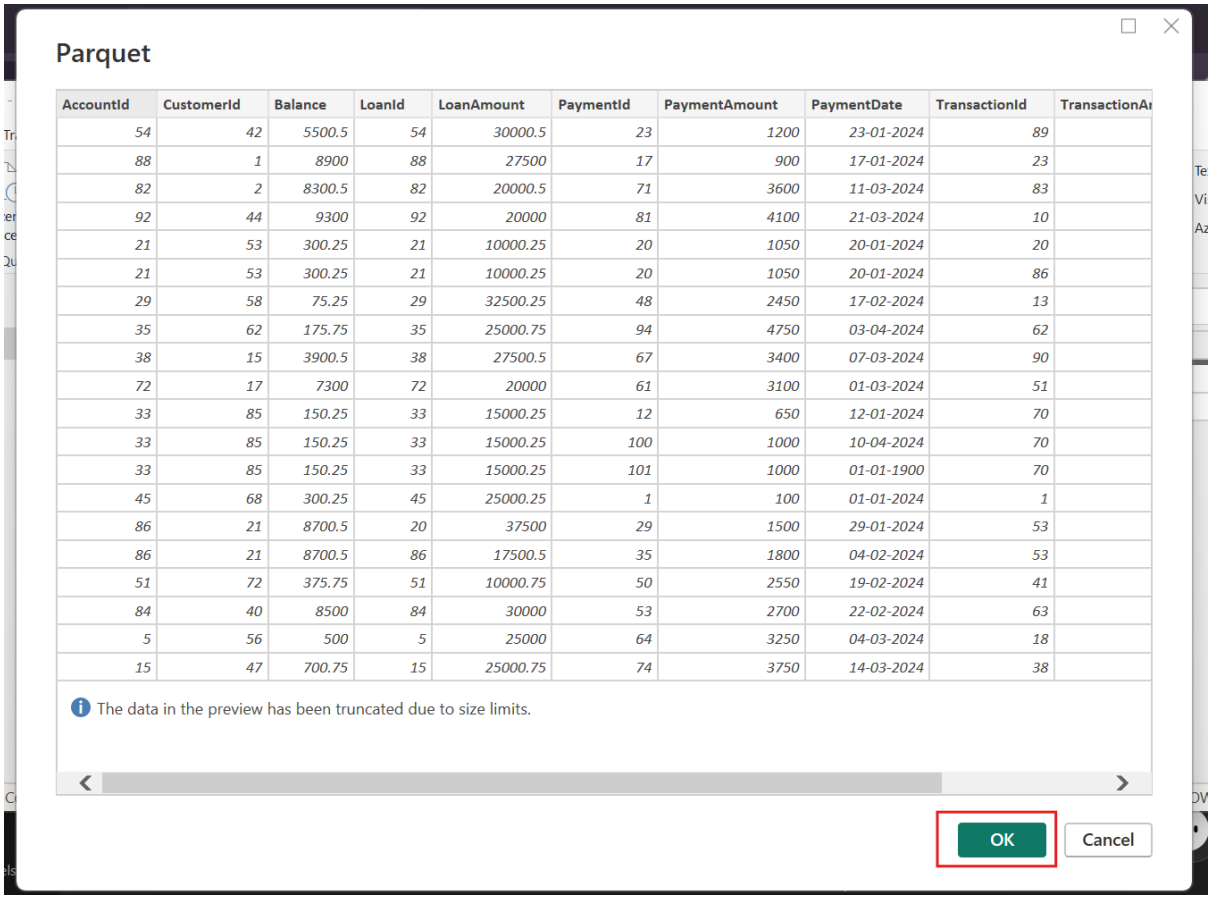
Step 4: Go to Extension column -> select parquet format -> click on ok. (To apply the filter to show all parquet files only).

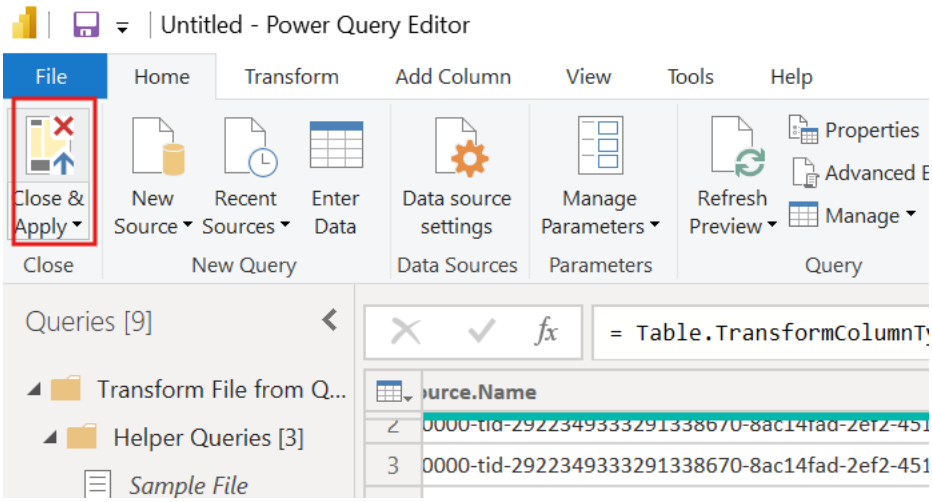




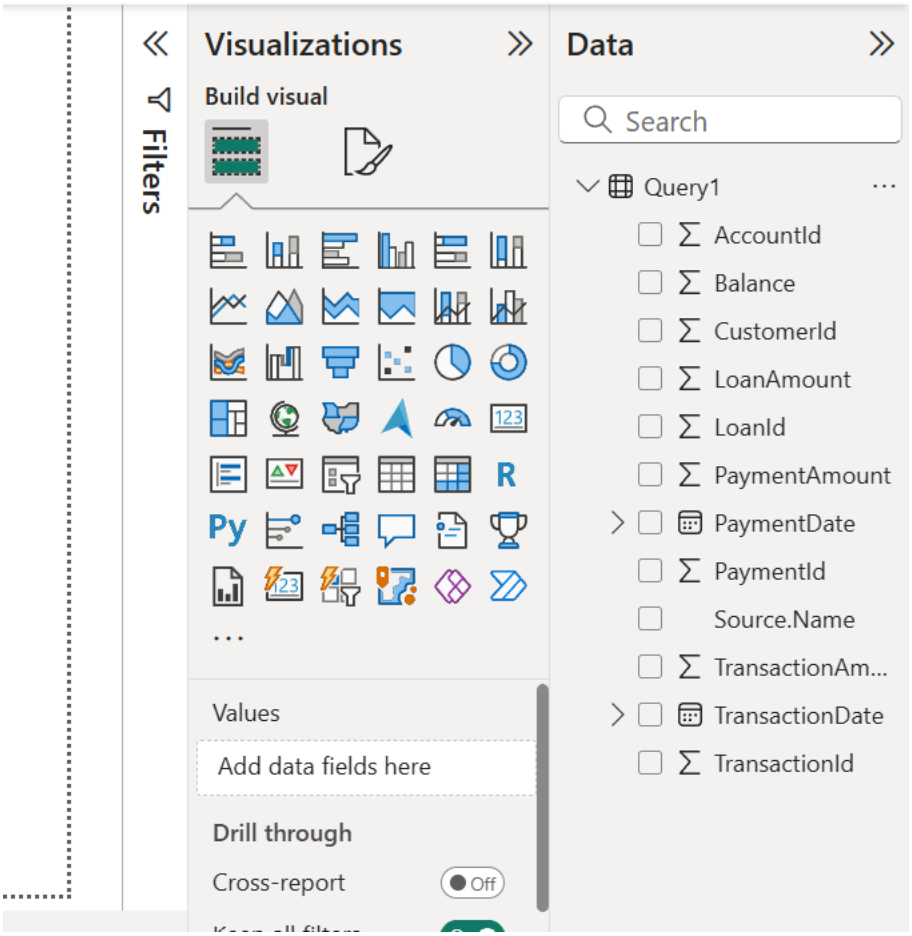
Step 5: Go to Content column click on double arrow to combine the data. -> review the data -> click on OK -> click on Close and apply button.



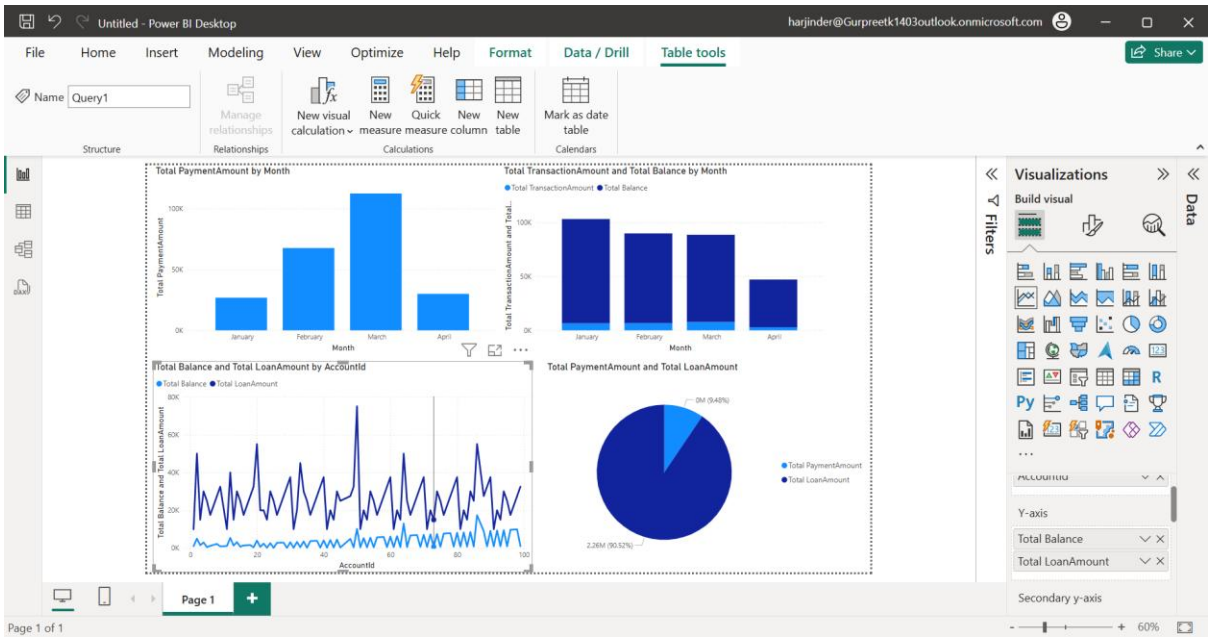




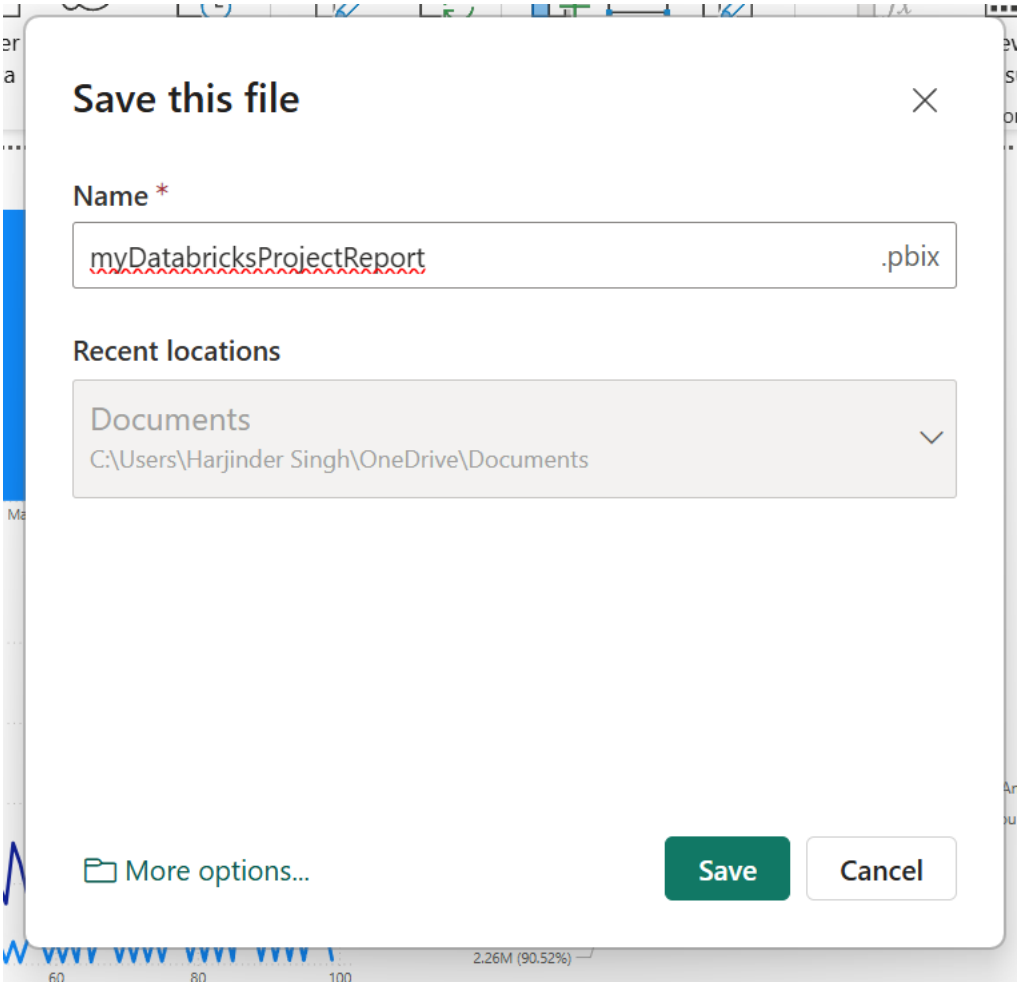
Step 6: Next, select the data columns to visualize the data from data section.

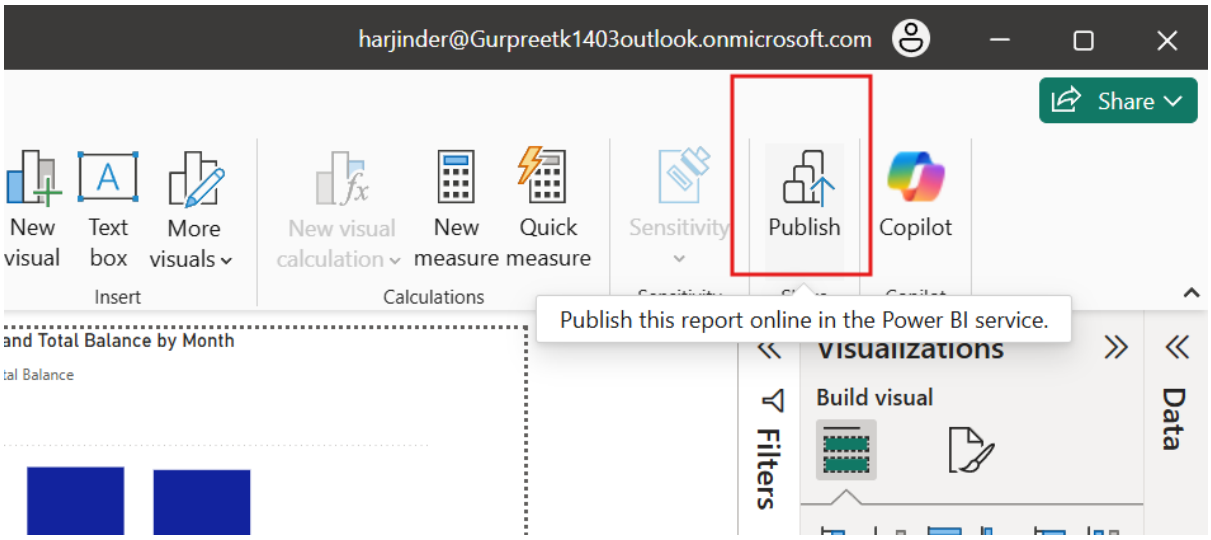


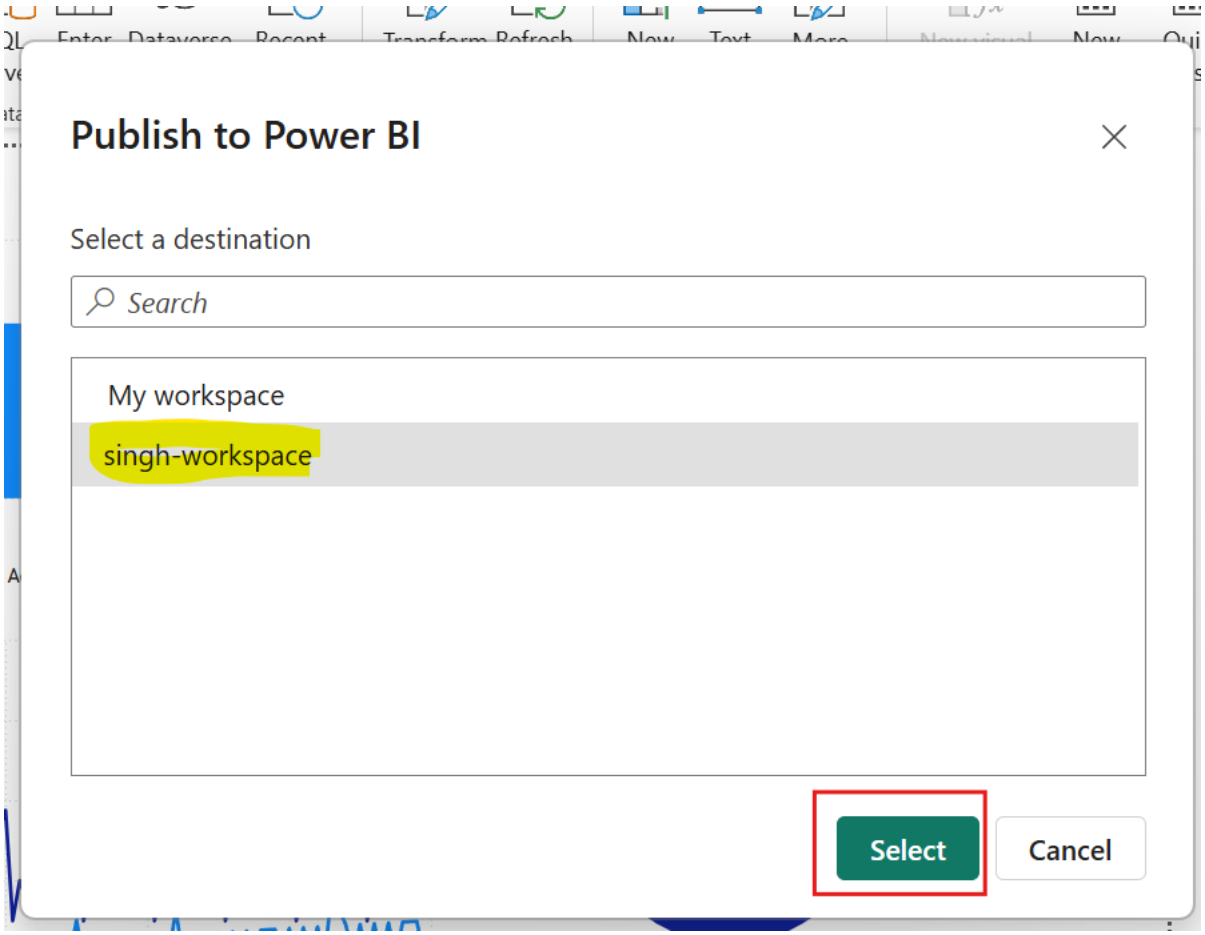
Step 7: Next, create some visuals and report by using data columns.



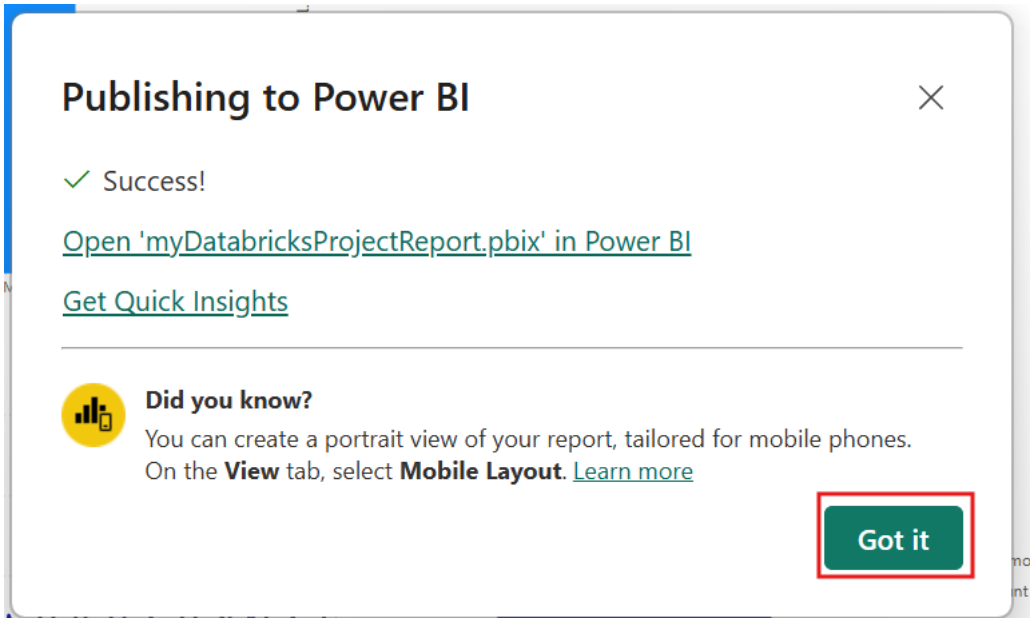
Step 8: Next, save the file -> select the fabric workspace to publish the power BI report -> click on select.

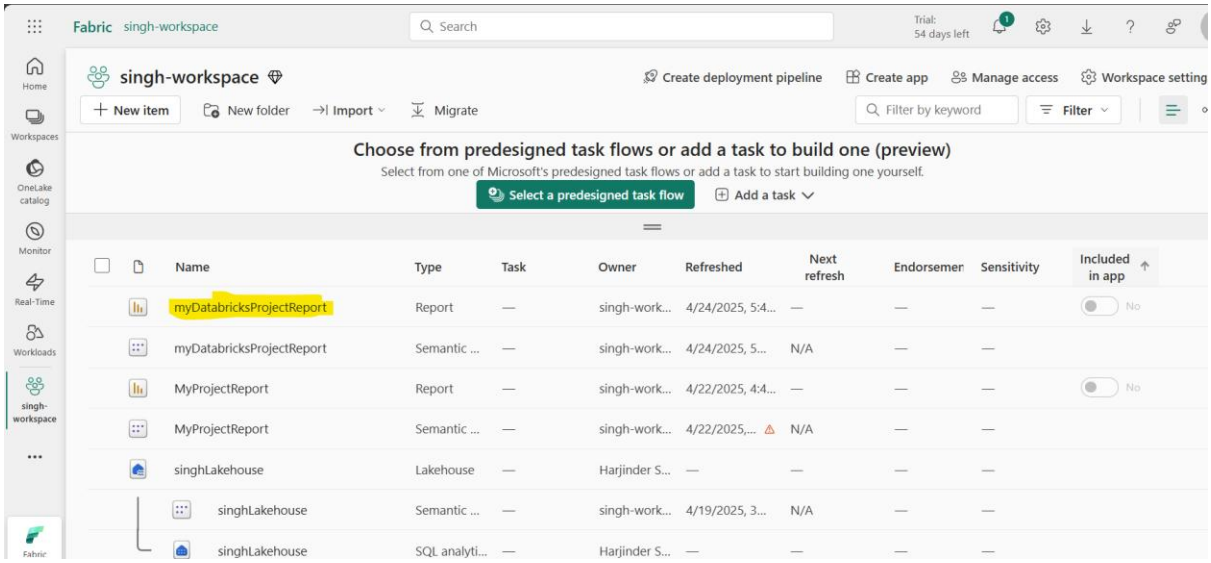


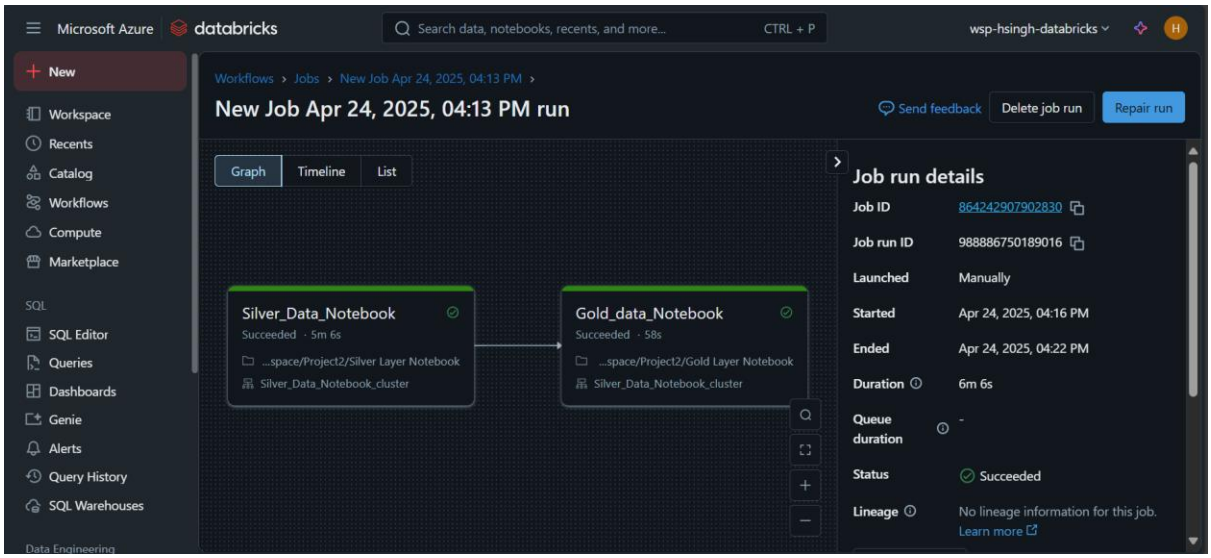




Step 9: Finally, review the published report in the fabric workspace.




Fabric singh-workspace
Search
Trial: 54 days left
singh-workspace
Create deployment pipeline
Create app
Manage access
Workspace setting
New item
New folder
Import
Migrate
Filter by keyword
Filter
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
Endorsemen
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...


Microsoft Azure
databricks
Search data, notebooks, recents, and more...
CTRL + P
wsp-hsingh-databricks
New
Workspace
Recents
Catalog
Workflows
Compute
Marketplace
SQL
SQL Editor
Queries
Dashboards
Genie
Alerts
Query History
SQL Warehouses
Data Engineering
Workflows › Jobs › New Job Apr 24, 2025, 04:13 PM ›
New Job Apr 24, 2025, 04:13 PM run
Send feedback
Delete job run
Repair run
Graph
Timeline
List
Silver_Data_Notebook
Succeeded · 5m 6s
...space/Project2/Silver Layer Notebook
Silver_Data_Notebook_cluster
Gold_data_Notebook
Succeeded · 58s
...space/Project2/Gold Layer Notebook
Silver_Data_Notebook_cluster
Job run details
Job ID
864242907902830
Job run ID
988886750189016
Launched
Manually
Started
Apr 24, 2025, 04:16 PM
Ended
Apr 24, 2025, 04:22 PM
Duration
6m 6s
Queue duration
-
Status
Succeeded
Lineage
No lineage information for this job.
Learn more