Amandeep Singh

Bootcamp Project 2 – Financial Data Analysis

Outline of the Project:

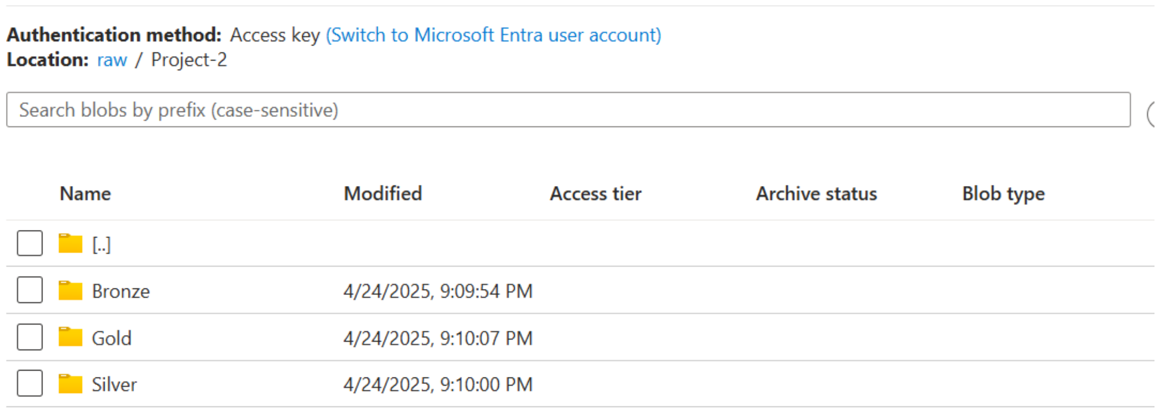
You are given 5 csv files which represent data related to Transactions and Loan acquired by Customers. Your task is to do proper ETL process for the data which includes data cleaning and transformation for storage purpose and store the processed data into SCD Type 1 Dimension Tables. Also, create table which encapsulate combined data created by joining raw tables. And At last, populate the data to Desktop Power BI.

Use only Databricks for all the computation and transformations.

Work Flow of Project:

Storing of the CSV files in ADLS G2 (bronze layer)

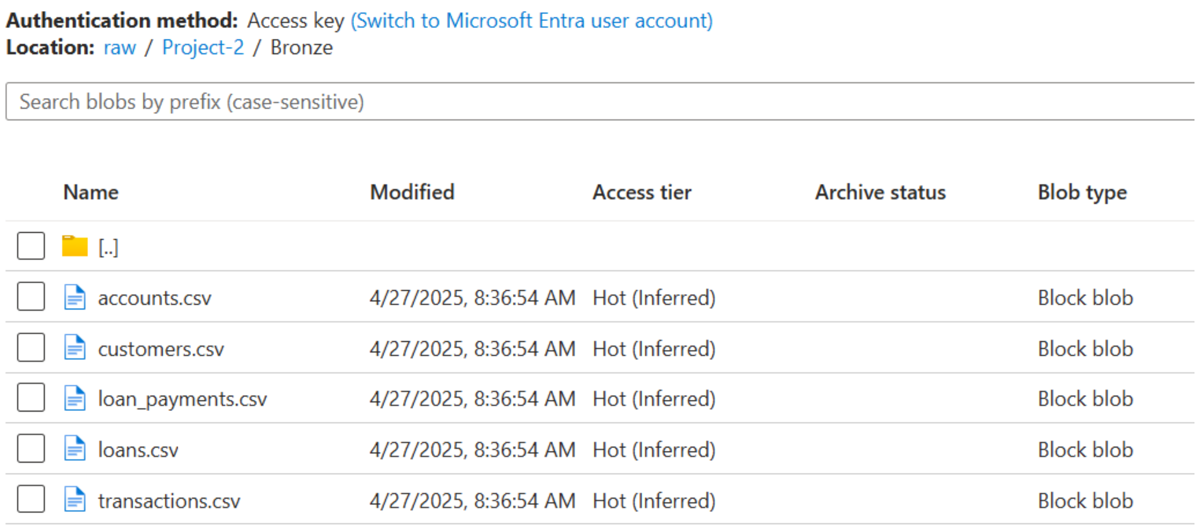
- I've created 3 directories in ADLS G2 (bronze, silver, Gold) which willbe used in the project to follow medilean structure:



Authentication method: Access key (Switch to Microsoft Entra user account)
Location: raw / Project-2

Name	Modified	Access tier	Archive status	Blob type
[..]				
Bronze	4/24/2025, 9:09:54 PM			
Gold	4/24/2025, 9:10:07 PM			
Silver	4/24/2025, 9:10:00 PM			

- I've stored 5 csv files in bronze directory:



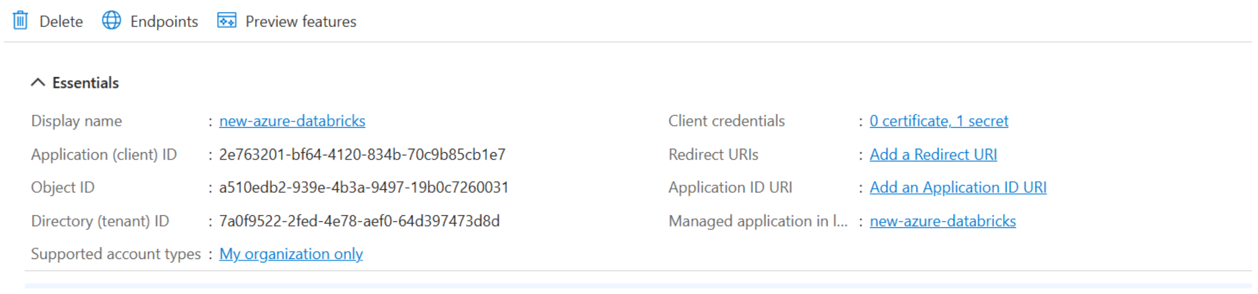
Authentication method: Access key (Switch to Microsoft Entra user account)
Location: raw / Project-2 / Bronze

Search blobs by prefix (case-sensitive)

Name	Modified	Access tier	Archive status	Blob type
[..]				
accounts.csv	4/27/2025, 8:36:54 AM	Hot (Inferred)		Block blob
customers.csv	4/27/2025, 8:36:54 AM	Hot (Inferred)		Block blob
loan_payments.csv	4/27/2025, 8:36:54 AM	Hot (Inferred)		Block blob
loans.csv	4/27/2025, 8:36:54 AM	Hot (Inferred)		Block blob
transactions.csv	4/27/2025, 8:36:54 AM	Hot (Inferred)		Block blob

Creating connection between ADLS G2 and databricks using Service principal method:

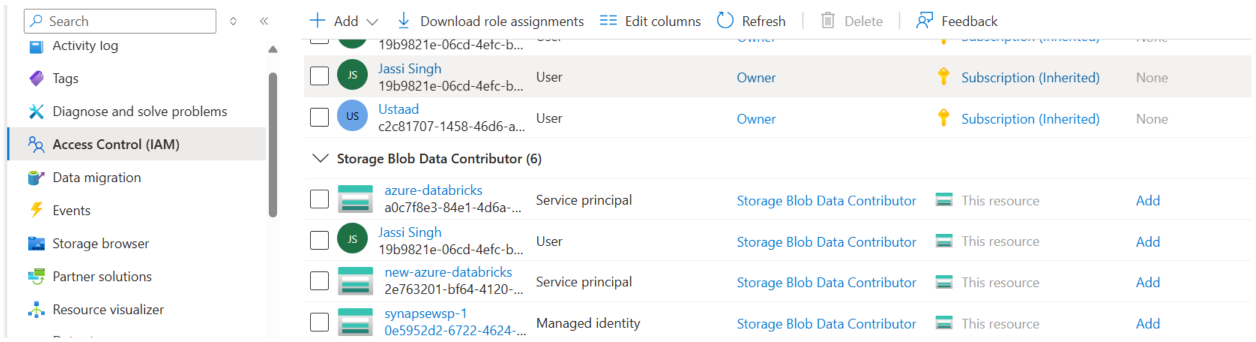
- To create service principal, we need to do new app registration in Microsoft Entra ID:



Display name	new-azure-databricks
Application (client) ID	2e763201-bf64-4120-834b-70c9b85cb1e7
Object ID	a510edb2-939e-4b3a-9497-19b0c7260031
Directory (tenant) ID	7a0f9522-2fed-4e78-aef0-64d397473d8d
Supported account types	My organization only
Client credentials	0 certificate, 1 secret
Redirect URIs	Add a Redirect URI
Application ID URI	Add an Application ID URI
Managed application in l...	new-azure-databricks

Store application ID and Tenant ID of the app in your system for future use.
Next, we need to create Client secret.
Create the secret and store it's value with you for future use (we cann't access this value later)

After saving all 3 info in local, we need to give Storage Blob contributor role to this new service principal.
To do that, go to ADLS G2 contianer> IAM access> add new role>select Blob Storage contributor role> assign to new service principal name.

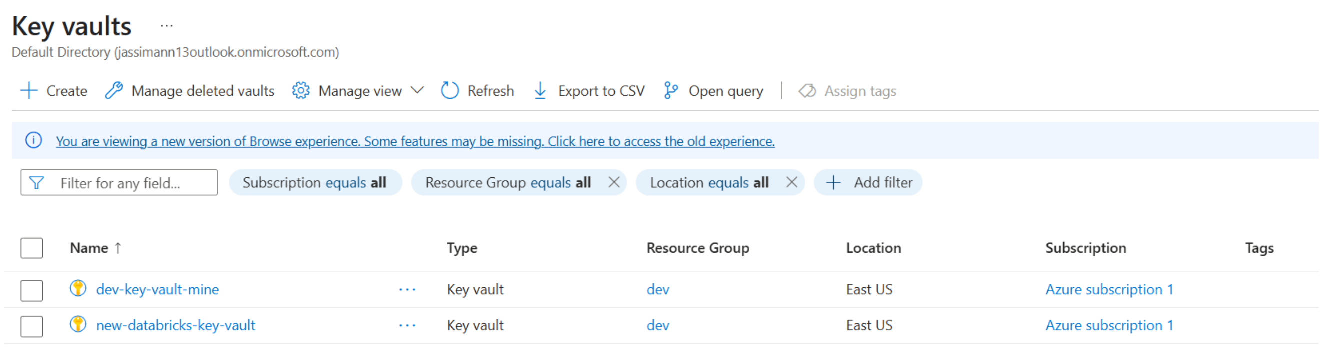


Search | Activity log | Tags | Diagnose and solve problems | Access Control (IAM) | Data migration | Events | Storage browser | Partner solutions | Resource visualizer

Add | Download role assignments | Edit columns | Refresh | Delete | Feedback

19b9821e-06cd-4efc-b...				
Jassi Singh 19b9821e-06cd-4efc-b...	User	Owner	Subscription (Inherited)	None
Ustaad c2c81707-1458-46d6-a...	User	Owner	Subscription (Inherited)	None
Storage Blob Data Contributor (6)				
azure-databricks a0c7f8e3-84e1-4d6a-...	Service principal	Storage Blob Data Contributor	This resource	Add
Jassi Singh 19b9821e-06cd-4efc-b...	User	Storage Blob Data Contributor	This resource	Add
new-azure-databricks 2e763201-bf64-4120-...	Service principal	Storage Blob Data Contributor	This resource	Add
synapsewsp-1 0e5952d2-6722-4624-...	Managed identity	Storage Blob Data Contributor	This resource	Add

- Creation of key vault and add 3 secrets in key vault and give permissions of reading and listing secrets to New service principal:



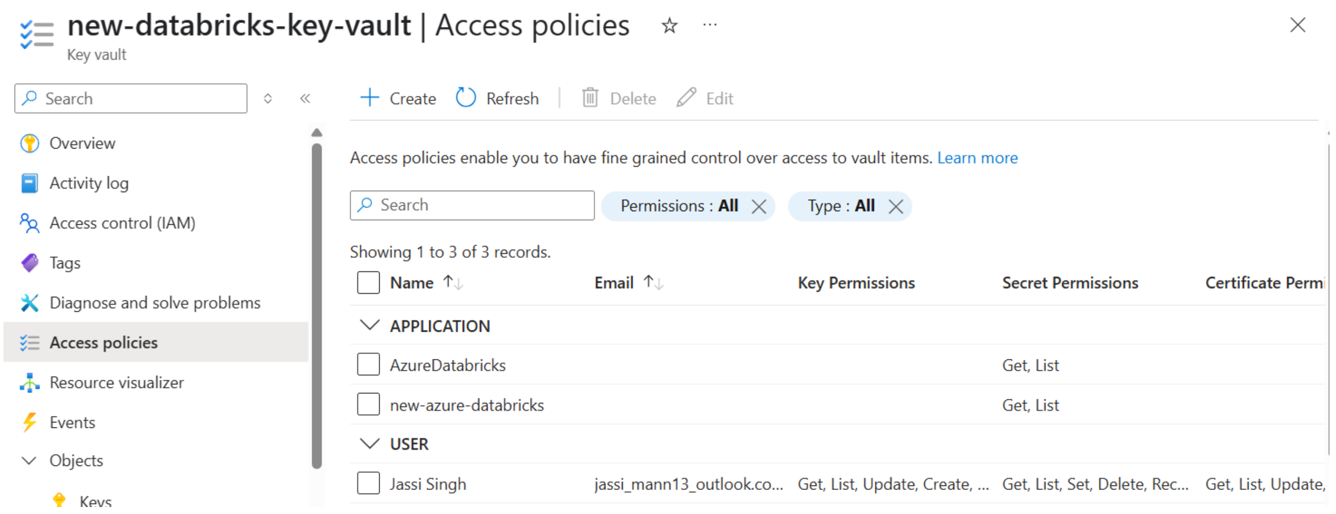
Key vaults
Default Directory (jassimann13outlook.onmicrosoft.com)

Create | Manage deleted vaults | Manage view | Refresh | Export to CSV | Open query | Assign tags

You are viewing a new version of Browse experience. Some features may be missing. Click here to access the old experience.

Filter for any field... | Subscription equals **all** | Resource Group equals **all** | Location equals **all** | Add filter

Name ↑	Type	Resource Group	Location	Subscription	Tags
dev-key-vault-mine	Key vault	dev	East US	Azure subscription 1	
new-databricks-key-vault	Key vault	dev	East US	Azure subscription 1	



new-databricks-key-vault | Access policies
Key vault

Search | Overview | Activity log | Access control (IAM) | Tags | Diagnose and solve problems | Access policies | Resource visualizer | Events | Objects | Keys

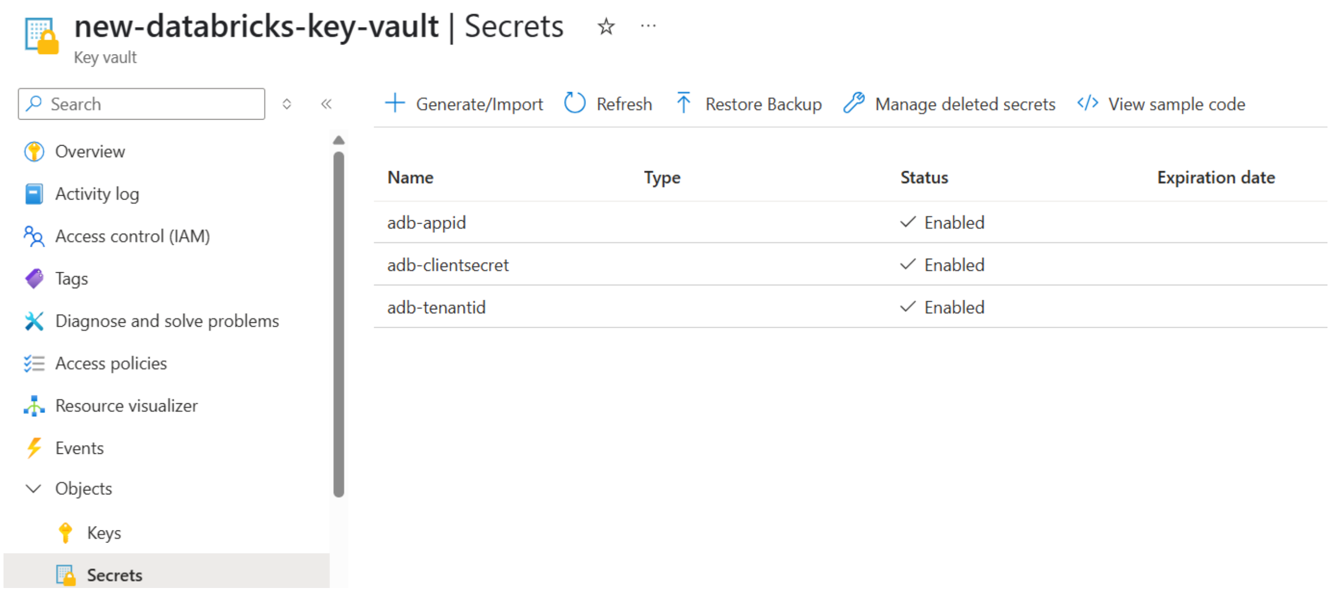
Create | Refresh | Delete | Edit

Access policies enable you to have fine grained control over access to vault items. Learn more

Search | Permissions : **All** | Type : **All**

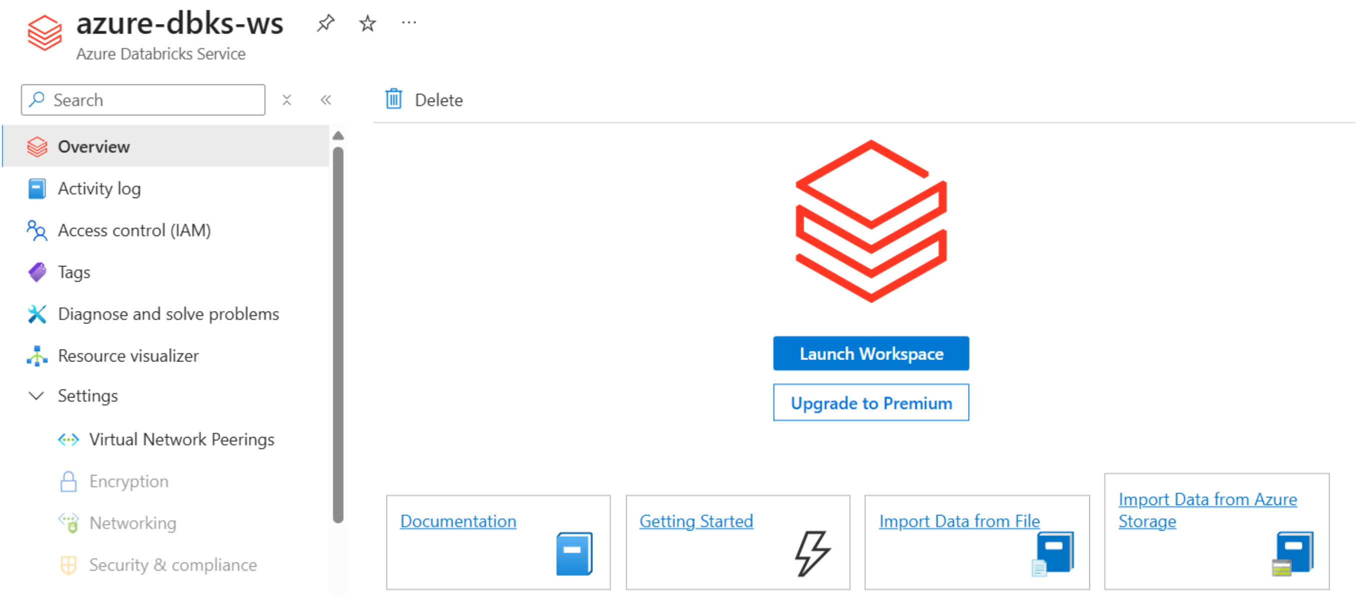
Showing 1 to 3 of 3 records.

Name	Email	Key Permissions	Secret Permissions	Certificate Perm
APPLICATION				
AzureDatabricks			Get, List	
new-azure-databricks			Get, List	
USER				
Jassi Singh	jassi_mann13_outlook.co...	Get, List, Update, Create, ...	Get, List, Set, Delete, Rec...	Get, List, Update,

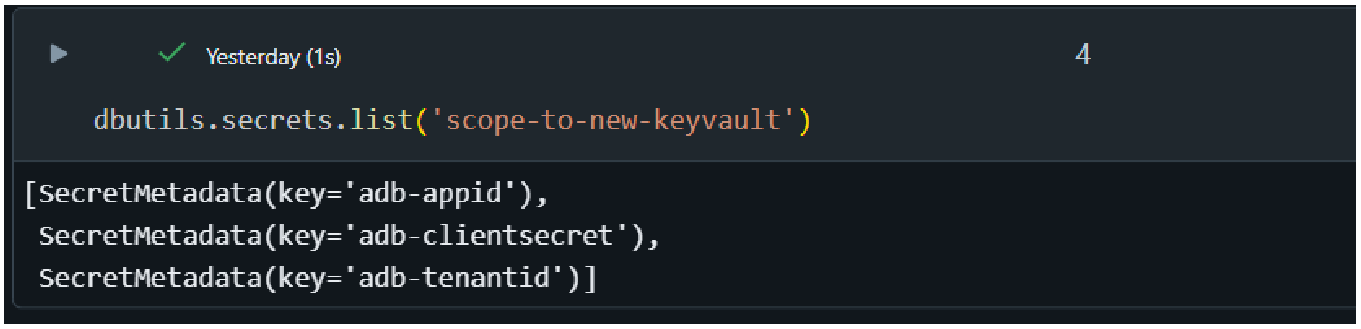


Name	Type	Status	Expiration date
adb-appid		✓ Enabled	
adb-clientsecret		✓ Enabled	
adb-tenantid		✓ Enabled	

- Creation of Azure Databricks Workspace



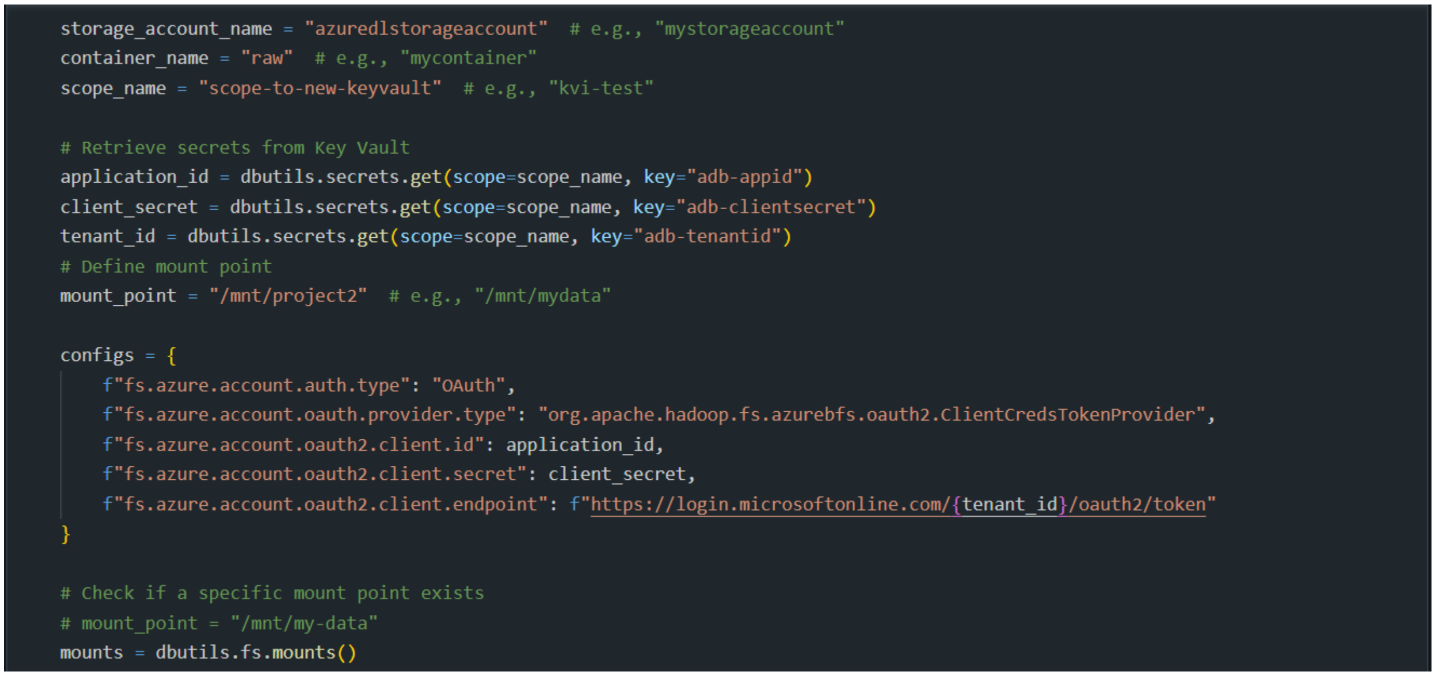
- To access ADLS G2 from Databricks notebook, we need to create scope in notebook so that, we can access key vault secrets we have created. For that, we need connection string/URI of Key vault and resource ID of it too.
- Go to <-------databricks url---->/#secrets/createScope
- Then you can access the key vault secrets by using dbutils.secrets commands


Yesterday (1s) 4

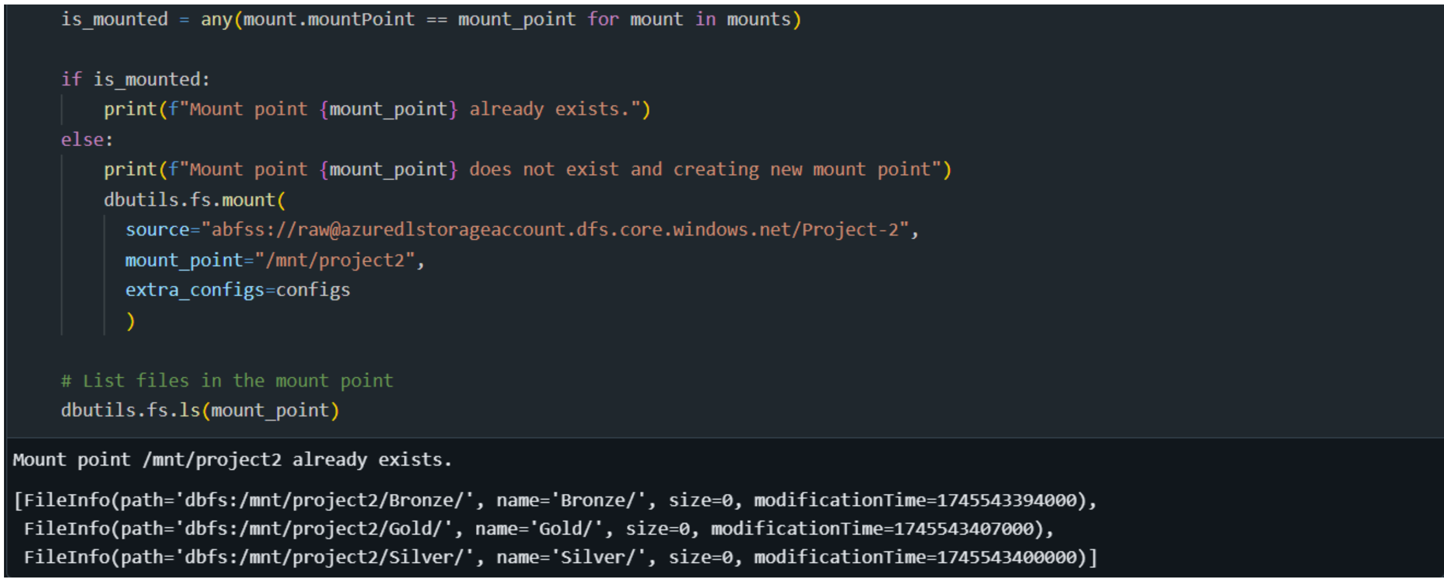
```
dbutils.secrets.list('scope-to-new-keyvault')
```

```
[SecretMetadata(key='adb-appid'),
 SecretMetadata(key='adb-clientsecret'),
 SecretMetadata(key='adb-tenantid')]
```

- Connect with ADLS G2 using service principal method by using following Code:



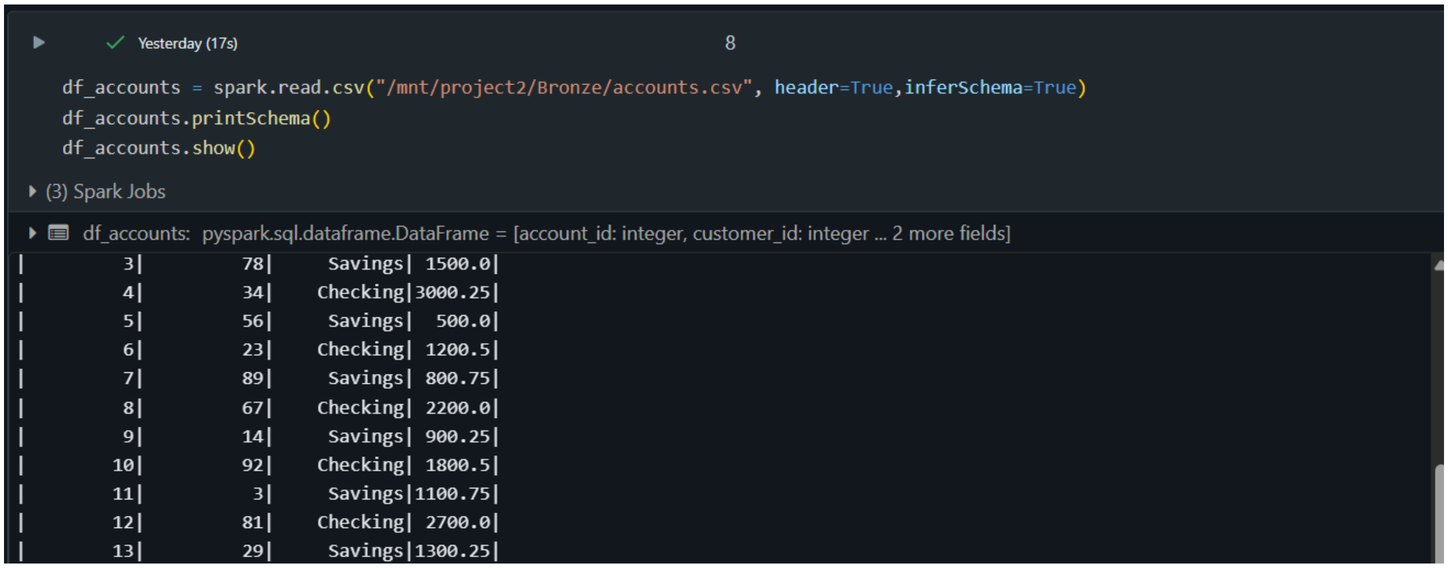
```python
storage_account_name = "azuredlstorageaccount"  # e.g., "mystorageaccount"
container_name = "raw"  # e.g., "mycontainer"
scope_name = "scope-to-new-keyvault"  # e.g., "kvi-test"

# Retrieve secrets from Key Vault
application_id = dbutils.secrets.get(scope=scope_name, key="adb-appid")
client_secret = dbutils.secrets.get(scope=scope_name, key="adb-clientsecret")
tenant_id = dbutils.secrets.get(scope=scope_name, key="adb-tenantid")
# Define mount point
mount_point = "/mnt/project2"  # e.g., "/mnt/mydata"

configs = {
    f"fs.azure.account.auth.type": "OAuth",
    f"fs.azure.account.oauth.provider.type": "org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider",
    f"fs.azure.account.oauth2.client.id": application_id,
    f"fs.azure.account.oauth2.client.secret": client_secret,
    f"fs.azure.account.oauth2.client.endpoint": f"https://login.microsoftonline.com/{tenant_id}/oauth2/token"
}

# Check if a specific mount point exists
# mount_point = "/mnt/my-data"
mounts = dbutils.fs.mounts()
```



```python
is_mounted = any(mount.mountPoint == mount_point for mount in mounts)

if is_mounted:
    print(f"Mount point {mount_point} already exists.")
else:
    print(f"Mount point {mount_point} does not exist and creating new mount point")
    dbutils.fs.mount(
      source="abfss://raw@azuredlstorageaccount.dfs.core.windows.net/Project-2",
      mount_point="/mnt/project2",
      extra_configs=configs
      )

# List files in the mount point
dbutils.fs.ls(mount_point)
```

```
Mount point /mnt/project2 already exists.
[FileInfo(path='dbfs:/mnt/project2/Bronze/', name='Bronze/', size=0, modificationTime=1745543394000),
 FileInfo(path='dbfs:/mnt/project2/Gold/', name='Gold/', size=0, modificationTime=1745543407000),
 FileInfo(path='dbfs:/mnt/project2/Silver/', name='Silver/', size=0, modificationTime=1745543400000)]
```

- To Read CSV file from ADLS G2 directory :



Yesterday (17s) 8

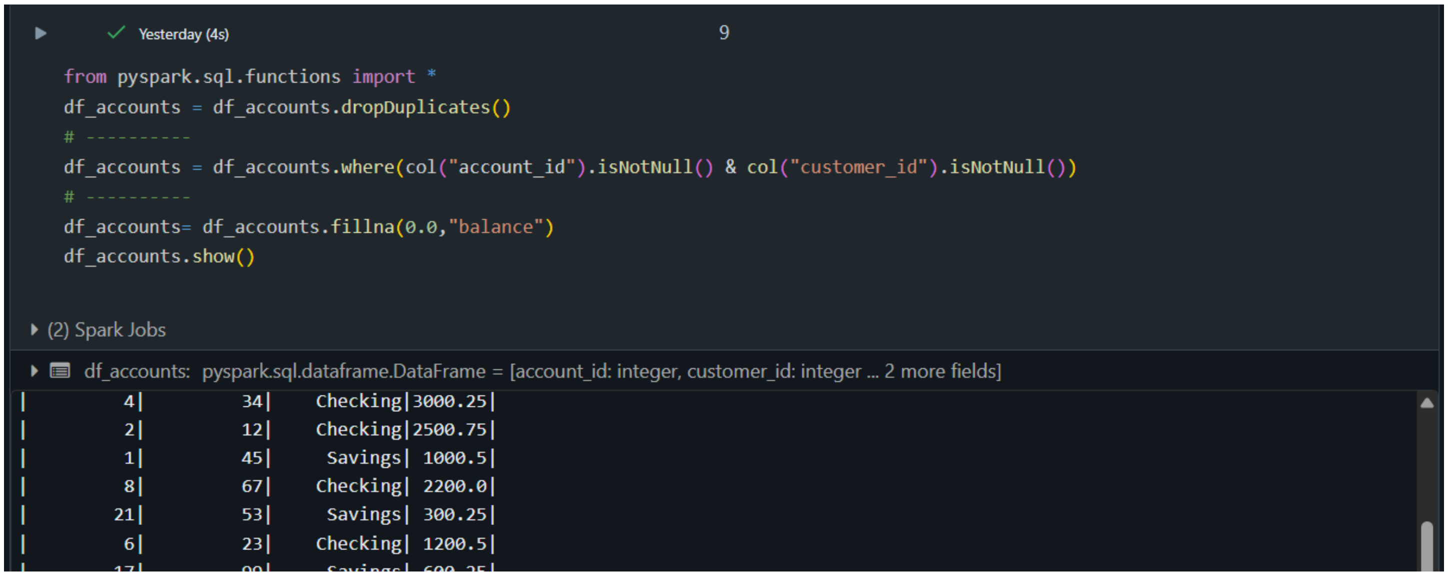
```
df_accounts = spark.read.csv("/mnt/project2/Bronze/accounts.csv", header=True,inferSchema=True)
df_accounts.printSchema()
df_accounts.show()
```

(3) Spark Jobs

df_accounts: pyspark.sql.dataframe.DataFrame = [account_id: integer, customer_id: integer ... 2 more fields]

```
|         3|         78|     Savings| 1500.0|
|         4|         34|    Checking|3000.25|
|         5|         56|     Savings|  500.0|
|         6|         23|    Checking| 1200.5|
|         7|         89|     Savings| 800.75|
|         8|         67|    Checking| 2200.0|
|         9|         14|     Savings| 900.25|
|        10|         92|    Checking| 1800.5|
|        11|          3|     Savings|1100.75|
|        12|         81|    Checking| 2700.0|
|        13|         29|     Savings|1300.25|
```

- *Data Transformation I did were :*



Yesterday (4s) 9

```
from pyspark.sql.functions import *
df_accounts = df_accounts.dropDuplicates()
# ----------
df_accounts = df_accounts.where(col("account_id").isNotNull() & col("customer_id").isNotNull())
# ---------
df_accounts= df_accounts.fillna(0.0,"balance")
df_accounts.show()
```

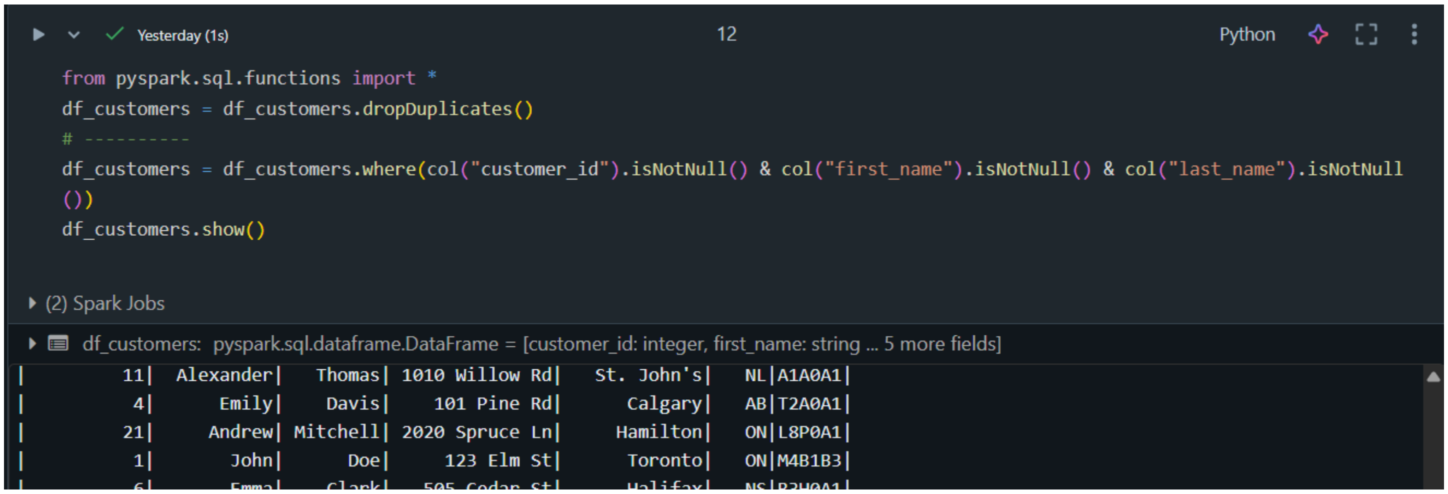
(2) Spark Jobs

df_accounts: pyspark.sql.dataframe.DataFrame = [account_id: integer, customer_id: integer ... 2 more fields]

```
|         4|         34|    Checking|3000.25|
|         2|         12|    Checking|2500.75|
|         1|         45|     Savings| 1000.5|
|         8|         67|    Checking| 2200.0|
|        21|         53|     Savings| 300.25|
|         6|         23|    Checking| 1200.5|
```

1. Dropping duplicates records
2. Filtering out records, which have non-null values of particular columns
3. Setting up default values to those column values which will be NULL.

- I've applied these transformations to all 5 Dataframes:

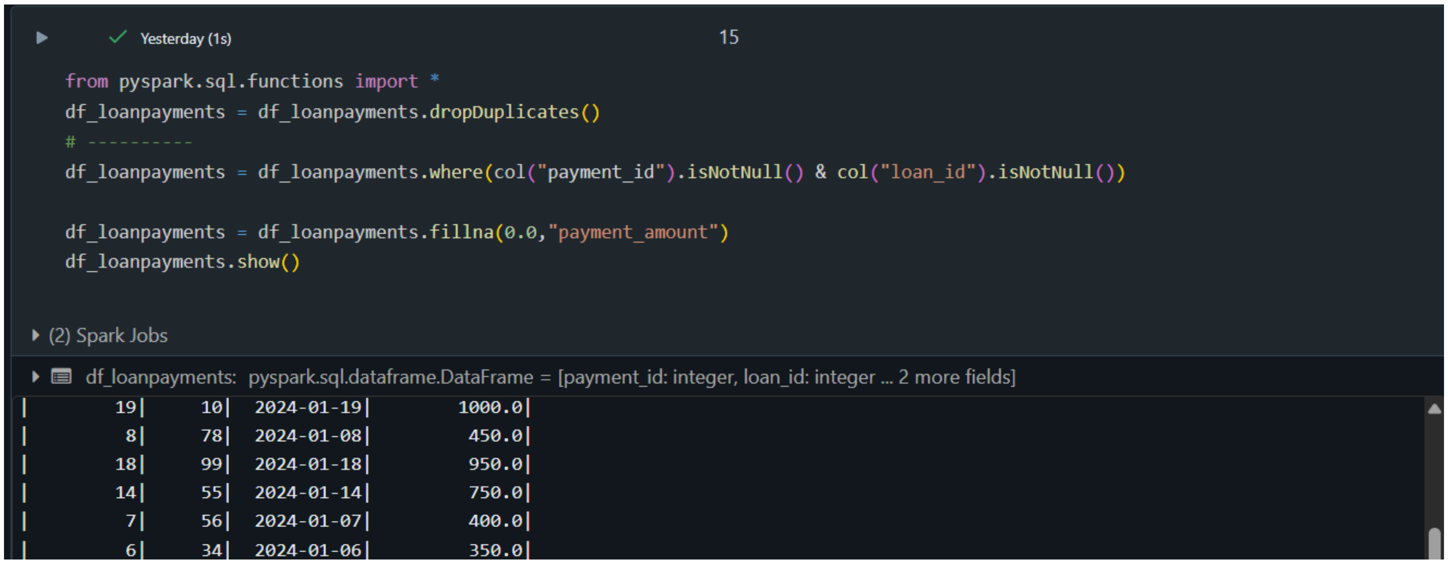

Yesterday (1s) 12 Python

```
from pyspark.sql.functions import *
df_customers = df_customers.dropDuplicates()
# ----------
df_customers = df_customers.where(col("customer_id").isNotNull() & col("first_name").isNotNull() & col("last_name").isNotNull
())
df_customers.show()
```

(2) Spark Jobs

df_customers: pyspark.sql.dataframe.DataFrame = [customer_id: integer, first_name: string ... 5 more fields]

```
|         11| Alexander|   Thomas| 1010 Willow Rd|   St. John's|    NL|A1A0A1|
|          4|     Emily|    Davis|    101 Pine Rd|      Calgary|    AB|T2A0A1|
|         21|    Andrew| Mitchell| 2020 Spruce Ln|     Hamilton|    ON|L8P0A1|
|          1|      John|      Doe|    123 Elm St|      Toronto|    ON|M4B1B3|
```

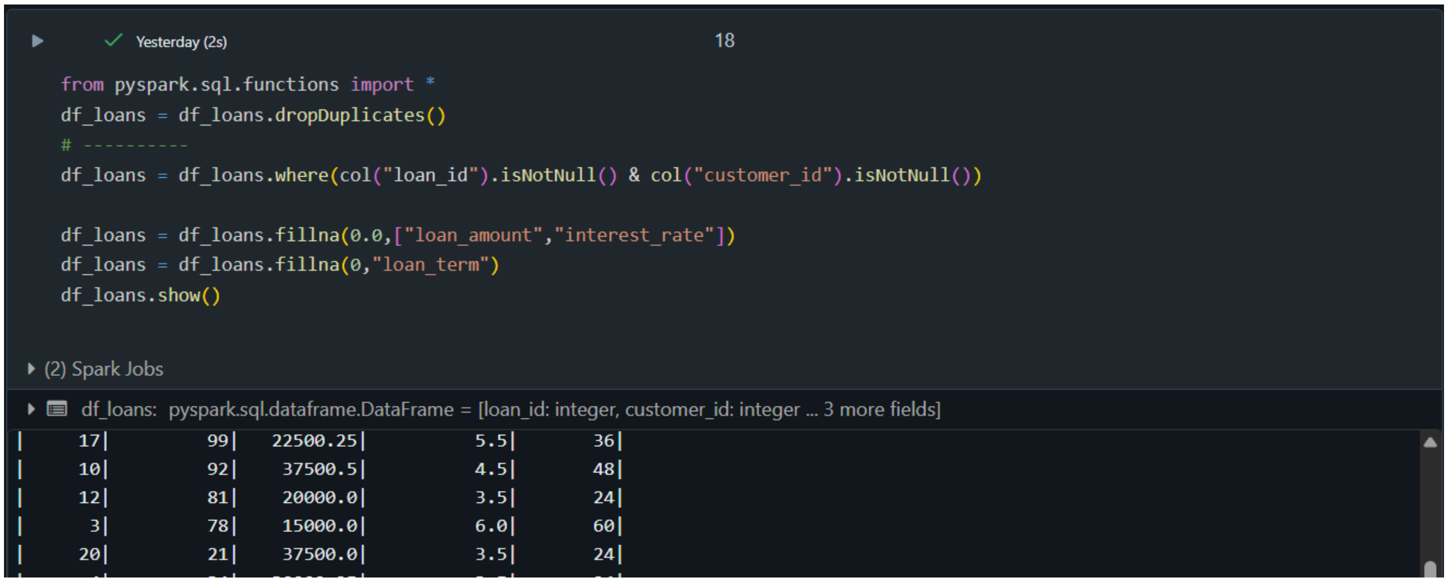

Yesterday (1s) 15

```
from pyspark.sql.functions import *
df_loanpayments = df_loanpayments.dropDuplicates()
# ----------
df_loanpayments = df_loanpayments.where(col("payment_id").isNotNull() & col("loan_id").isNotNull())

df_loanpayments = df_loanpayments.fillna(0.0,"payment_amount")
df_loanpayments.show()
```

(2) Spark Jobs

df_loanpayments: pyspark.sql.dataframe.DataFrame = [payment_id: integer, loan_id: integer ... 2 more fields]

```
|         19|     10|  2024-01-19|        1000.0|
|          8|     78|  2024-01-08|         450.0|
|         18|     99|  2024-01-18|         950.0|
|         14|     55|  2024-01-14|         750.0|
|          7|     56|  2024-01-07|         400.0|
|          6|     34|  2024-01-06|         350.0|
```

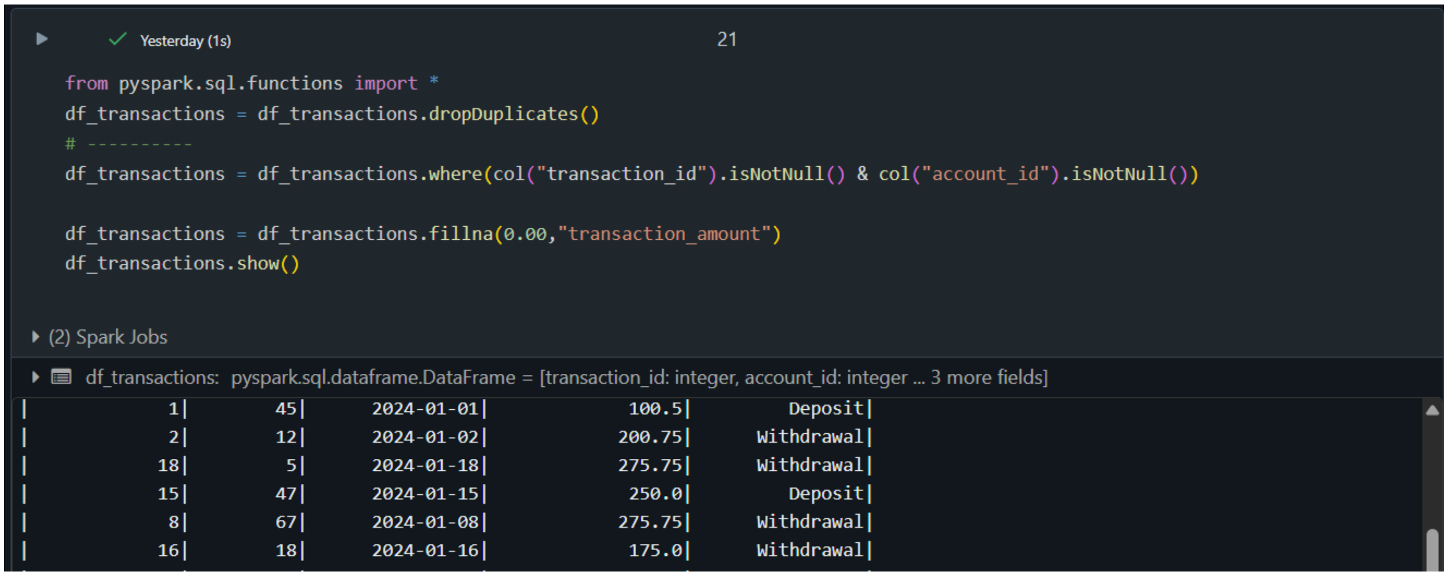

Yesterday (2s) 18

```
from pyspark.sql.functions import *
df_loans = df_loans.dropDuplicates()
# ----------
df_loans = df_loans.where(col("loan_id").isNotNull() & col("customer_id").isNotNull())

df_loans = df_loans.fillna(0.0,["loan_amount","interest_rate"])
df_loans = df_loans.fillna(0,"loan_term")
df_loans.show()
```

(2) Spark Jobs

df_loans: pyspark.sql.dataframe.DataFrame = [loan_id: integer, customer_id: integer ... 3 more fields]

```
|     17|          99|   22500.25|           5.5|       36|
|     10|          92|    37500.5|           4.5|       48|
|     12|          81|    20000.0|           3.5|       24|
|      3|          78|    15000.0|           6.0|       60|
|     20|          21|    37500.0|           3.5|       24|
```



Yesterday (1s) 21

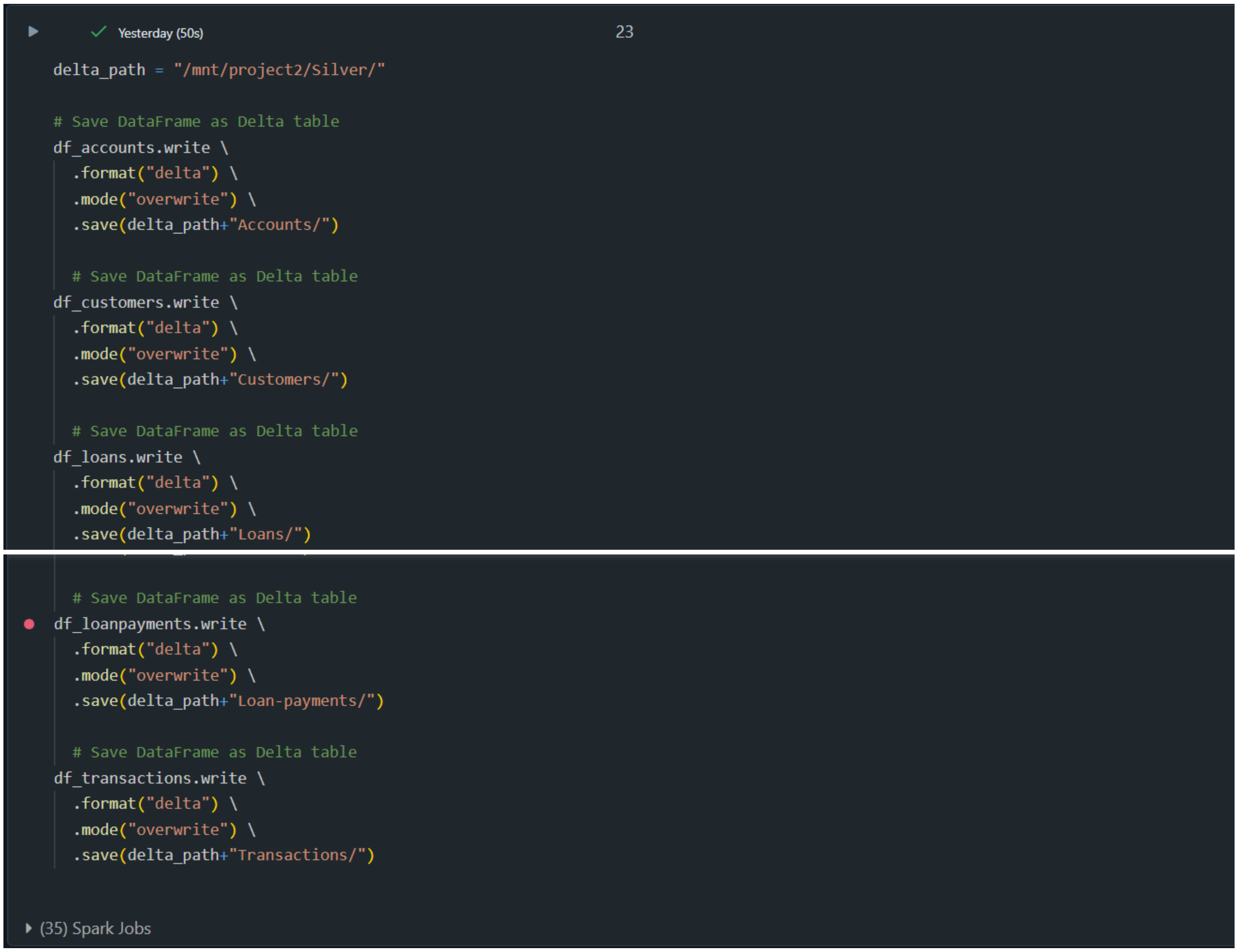
```
from pyspark.sql.functions import *
df_transactions = df_transactions.dropDuplicates()
# ----------
df_transactions = df_transactions.where(col("transaction_id").isNotNull() & col("account_id").isNotNull())

df_transactions = df_transactions.fillna(0.00,"transaction_amount")
df_transactions.show()
```

(2) Spark Jobs

df_transactions: pyspark.sql.dataframe.DataFrame = [transaction_id: integer, account_id: integer ... 3 more fields]

```
|             1|        45|    2024-01-01|             100.5|         Deposit|
|             2|        12|    2024-01-02|            200.75|      Withdrawal|
|            18|         5|    2024-01-18|            275.75|      Withdrawal|
|            15|        47|    2024-01-15|             250.0|         Deposit|
|             8|        67|    2024-01-08|            275.75|      Withdrawal|
|            16|        18|    2024-01-16|             175.0|      Withdrawal|
```

- Saving all transformed Data frames into delta format in silver layer – ADLS G2

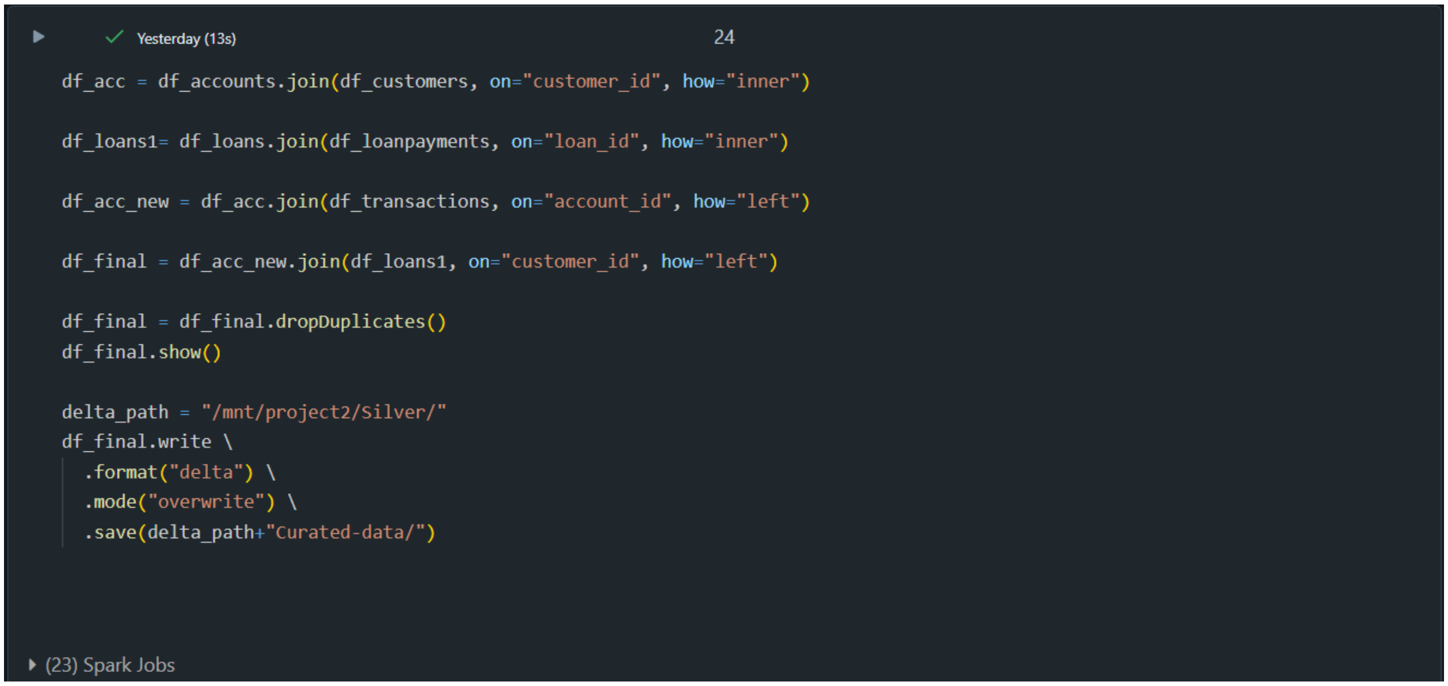


Yesterday (50s) 23

```
delta_path = "/mnt/project2/Silver/"

# Save DataFrame as Delta table
df_accounts.write \
  .format("delta") \
  .mode("overwrite") \
  .save(delta_path+"Accounts/")

  # Save DataFrame as Delta table
df_customers.write \
  .format("delta") \
  .mode("overwrite") \
  .save(delta_path+"Customers/")

  # Save DataFrame as Delta table
df_loans.write \
  .format("delta") \
  .mode("overwrite") \
  .save(delta_path+"Loans/")

  # Save DataFrame as Delta table
df_loanpayments.write \
  .format("delta") \
  .mode("overwrite") \
  .save(delta_path+"Loan-payments/")

  # Save DataFrame as Delta table
df_transactions.write \
  .format("delta") \
  .mode("overwrite") \
  .save(delta_path+"Transactions/")
```

(35) Spark Jobs

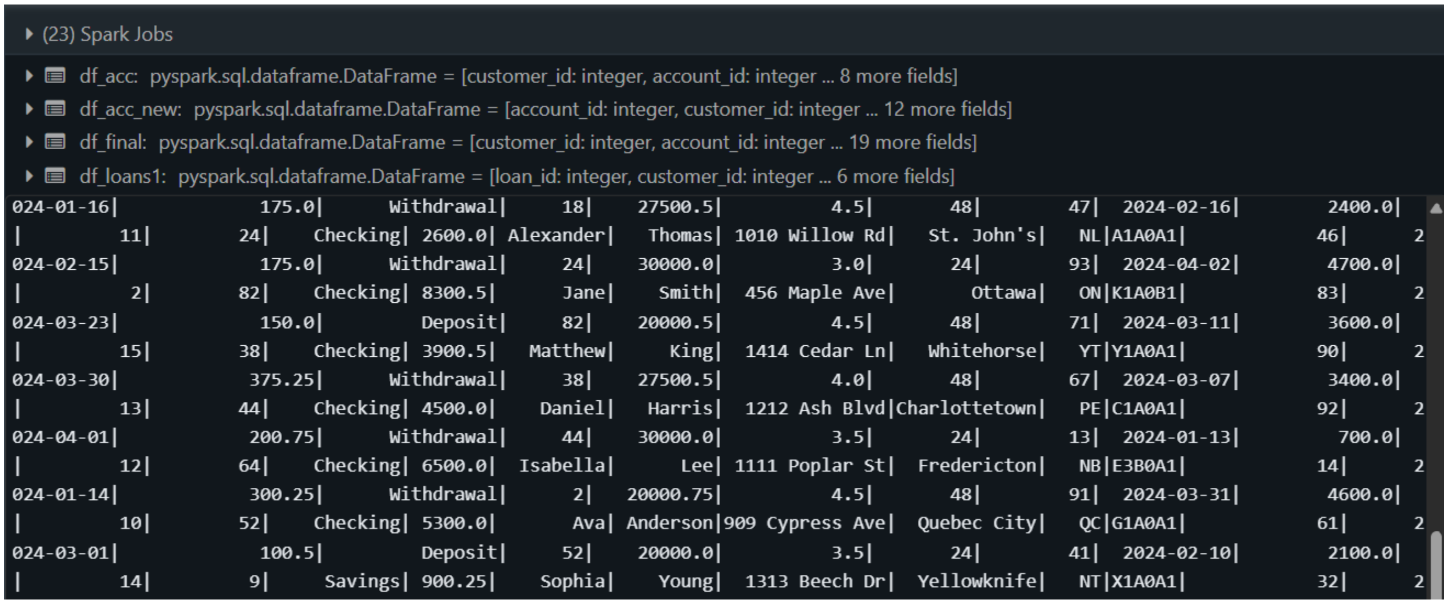
- To combine all Data frames into one common DF by using their relationships with each other, I've created a new Data frame and store it into delta table in ADLS Silver layer.



Yesterday (13s) 24

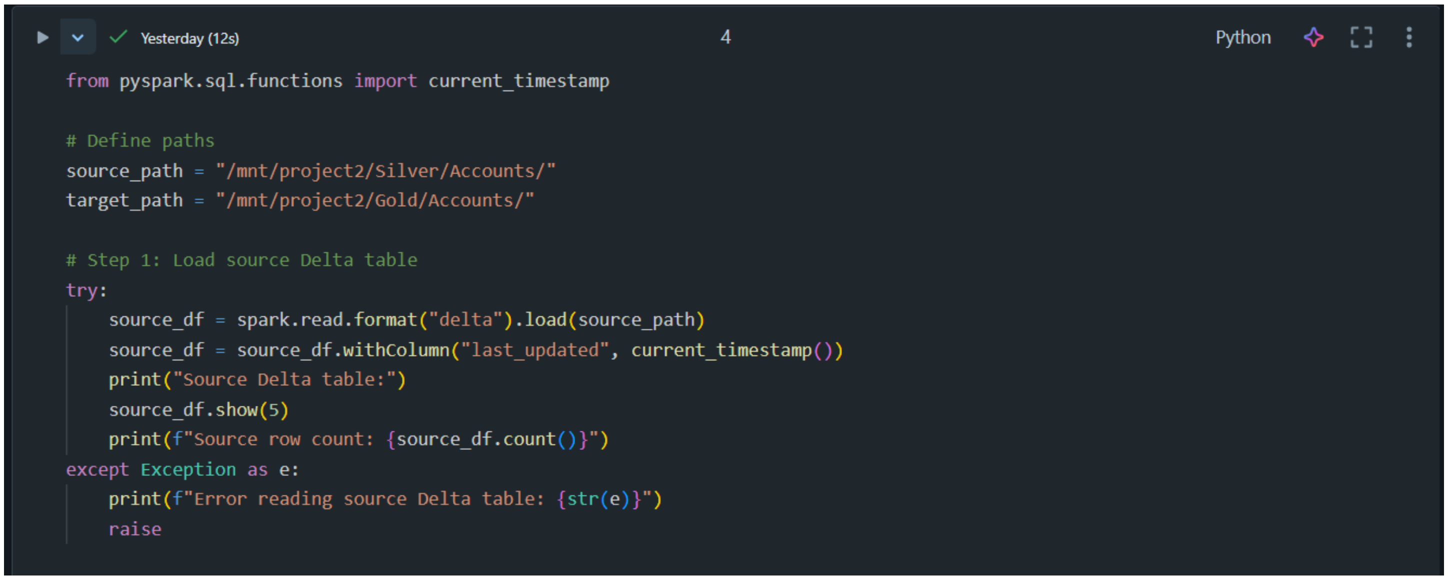
```python
df_acc = df_accounts.join(df_customers, on="customer_id", how="inner")

df_loans1= df_loans.join(df_loanpayments, on="loan_id", how="inner")

df_acc_new = df_acc.join(df_transactions, on="account_id", how="left")

df_final = df_acc_new.join(df_loans1, on="customer_id", how="left")

df_final = df_final.dropDuplicates()
df_final.show()

delta_path = "/mnt/project2/Silver/"
df_final.write \
  .format("delta") \
  .mode("overwrite") \
  .save(delta_path+"Curated-data/")
```

(23) Spark Jobs

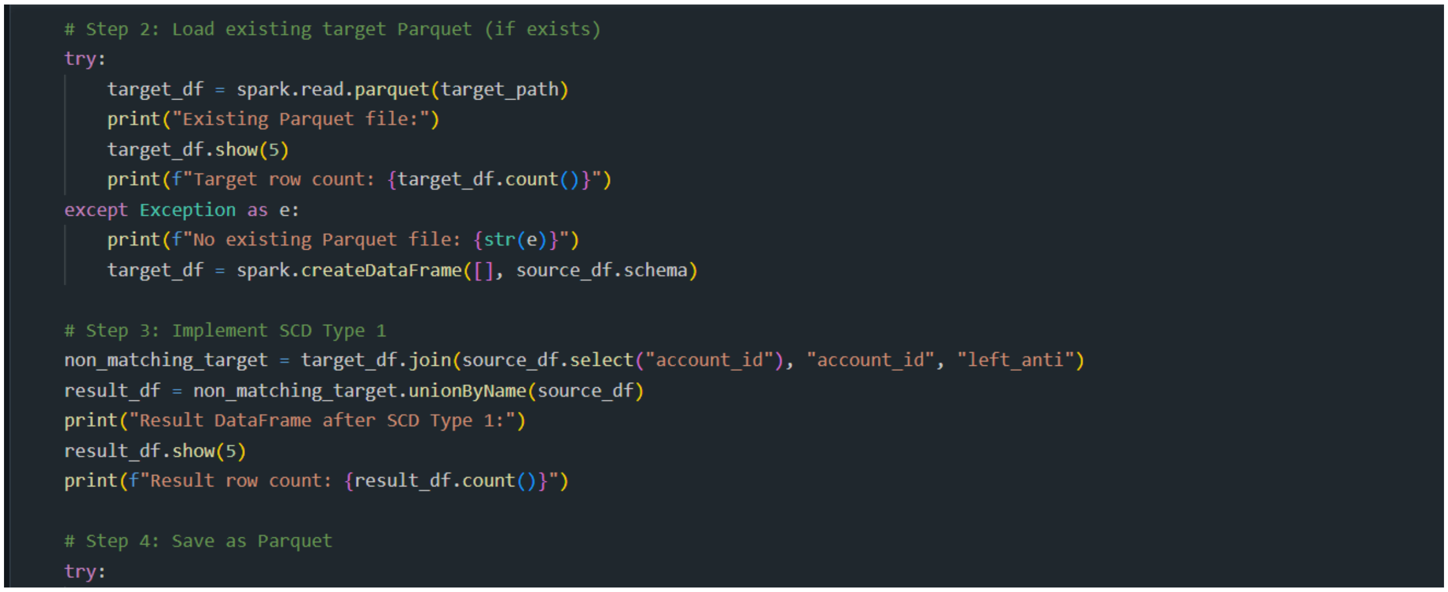


```
(23) Spark Jobs
df_acc:  pyspark.sql.dataframe.DataFrame = [customer_id: integer, account_id: integer ... 8 more fields]
df_acc_new:  pyspark.sql.dataframe.DataFrame = [account_id: integer, customer_id: integer ... 12 more fields]
df_final:  pyspark.sql.dataframe.DataFrame = [customer_id: integer, account_id: integer ... 19 more fields]
df_loans1:  pyspark.sql.dataframe.DataFrame = [loan_id: integer, customer_id: integer ... 6 more fields]
024-01-16|           175.0|      Withdrawal|     18|    27500.5|            4.5|         48|          47|  2024-02-16|           2400.0|
|        11|          24|    Checking| 2600.0| Alexander|    Thomas| 1010 Willow Rd|   St. John's|   NL|A1A0A1|            46|        2
024-02-15|           175.0|      Withdrawal|     24|    30000.0|            3.0|         24|          93|  2024-04-02|           4700.0|
|         2|          82|    Checking| 8300.5|      Jane|     Smith|  456 Maple Ave|       Ottawa|   ON|K1A0B1|            83|        2
024-03-23|           150.0|         Deposit|     82|    20000.5|            4.5|         48|          71|  2024-03-11|           3600.0|
|        15|          38|    Checking| 3900.5|   Matthew|      King|  1414 Cedar Ln|   Whitehorse|   YT|Y1A0A1|            90|        2
024-03-30|          375.25|      Withdrawal|     38|    27500.5|            4.0|         48|          67|  2024-03-07|           3400.0|
|        13|          44|    Checking| 4500.0|    Daniel|    Harris|  1212 Ash Blvd|Charlottetown|   PE|C1A0A1|            92|        2
024-04-01|          200.75|      Withdrawal|     44|    30000.0|            3.5|         24|          13|  2024-01-13|            700.0|
|        12|          64|    Checking| 6500.0|  Isabella|       Lee| 1111 Poplar St|  Fredericton|   NB|E3B0A1|            14|        2
024-01-14|          300.25|      Withdrawal|      2|   20000.75|            4.5|         48|          91|  2024-03-31|           4600.0|
|        10|          52|    Checking| 5300.0|       Ava|  Anderson|909 Cypress Ave|  Quebec City|   QC|G1A0A1|            61|        2
024-03-01|           100.5|         Deposit|     52|    20000.0|            3.5|         24|          41|  2024-02-10|           2100.0|
|        14|           9|     Savings| 900.25|    Sophia|     Young|  1313 Beech Dr|  Yellowknife|   NT|X1A0A1|            32|        2
```

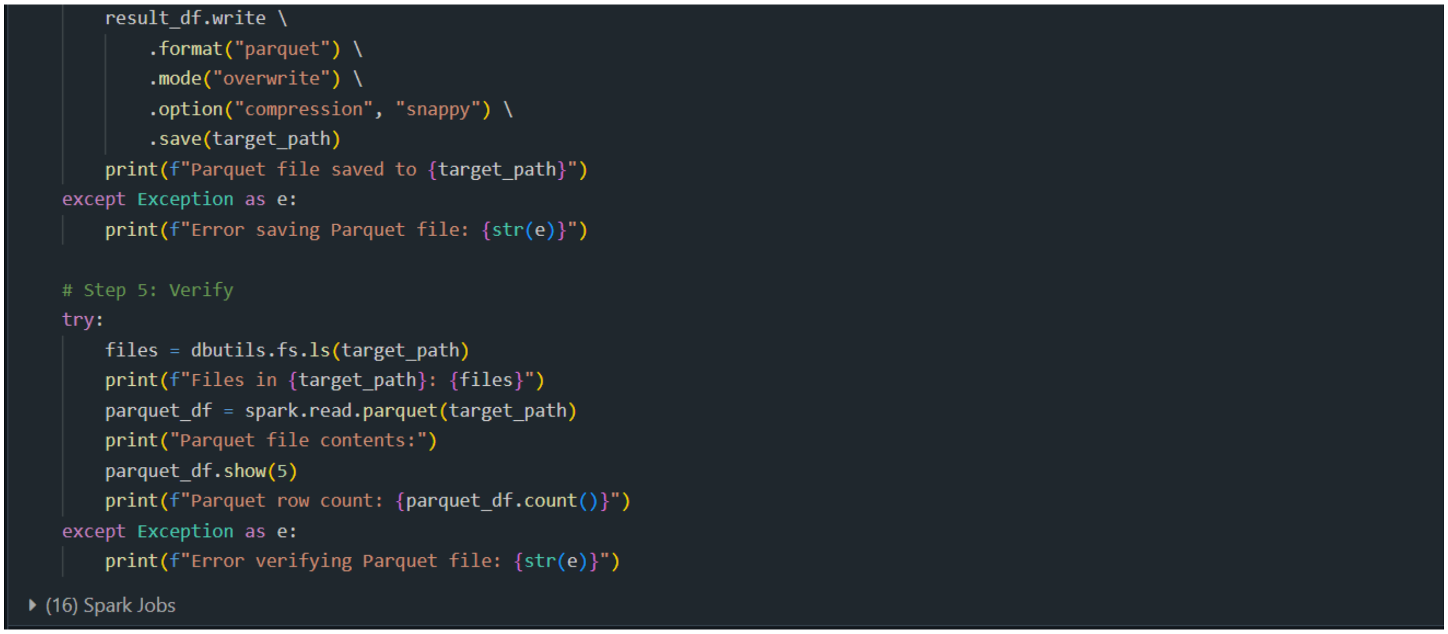
CONVERTING DATA INTO SCD TYPE 1 DIMENSION TABLES BY USING DATABRICKS:



```python
from pyspark.sql.functions import current_timestamp

# Define paths
source_path = "/mnt/project2/Silver/Accounts/"
target_path = "/mnt/project2/Gold/Accounts/"

# Step 1: Load source Delta table
try:
    source_df = spark.read.format("delta").load(source_path)
    source_df = source_df.withColumn("last_updated", current_timestamp())
    print("Source Delta table:")
    source_df.show(5)
    print(f"Source row count: {source_df.count()}")
except Exception as e:
    print(f"Error reading source Delta table: {str(e)}")
    raise
```



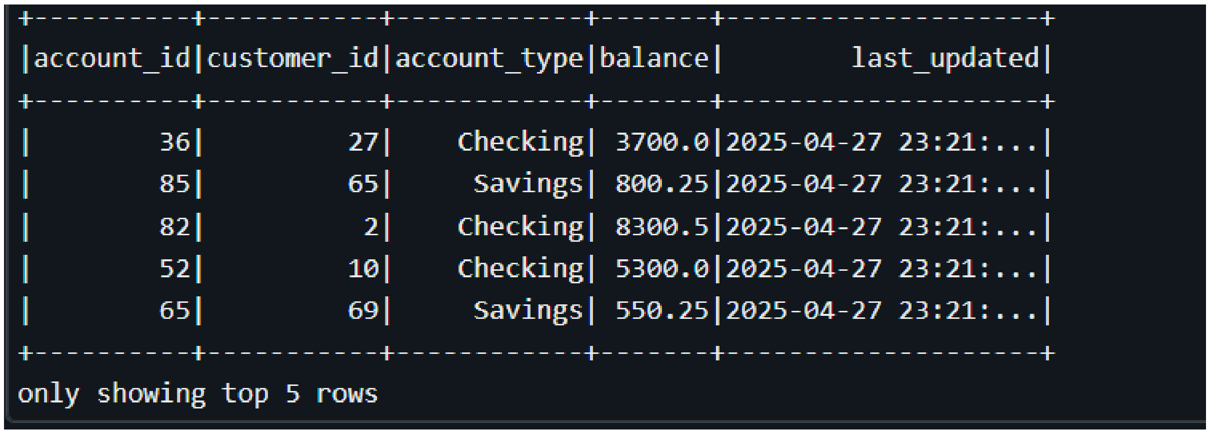
```python
# Step 2: Load existing target Parquet (if exists)
try:
    target_df = spark.read.parquet(target_path)
    print("Existing Parquet file:")
    target_df.show(5)
    print(f"Target row count: {target_df.count()}")
except Exception as e:
    print(f"No existing Parquet file: {str(e)}")
    target_df = spark.createDataFrame([], source_df.schema)

# Step 3: Implement SCD Type 1
non_matching_target = target_df.join(source_df.select("account_id"), "account_id", "left_anti")
result_df = non_matching_target.unionByName(source_df)
print("Result DataFrame after SCD Type 1:")
result_df.show(5)
print(f"Result row count: {result_df.count()}")

# Step 4: Save as Parquet
try:
```



```
        result_df.write \
            .format("parquet") \
            .mode("overwrite") \
            .option("compression", "snappy") \
            .save(target_path)
        print(f"Parquet file saved to {target_path}")
    except Exception as e:
        print(f"Error saving Parquet file: {str(e)}")

    # Step 5: Verify
    try:
        files = dbutils.fs.ls(target_path)
        print(f"Files in {target_path}: {files}")
        parquet_df = spark.read.parquet(target_path)
        print("Parquet file contents:")
        parquet_df.show(5)
        print(f"Parquet row count: {parquet_df.count()}")
    except Exception as e:
        print(f"Error verifying Parquet file: {str(e)}")
```

▸ (16) Spark Jobs

- New Data frame created :



```
+----------+-----------+------------+-------+--------------------+
|account_id|customer_id|account_type|balance|        last_updated|
+----------+-----------+------------+-------+--------------------+
|        36|         27|    Checking| 3700.0|2025-04-27 23:21:...|
|        85|         65|     Savings| 800.25|2025-04-27 23:21:...|
|        82|          2|    Checking| 8300.5|2025-04-27 23:21:...|
|        52|         10|    Checking| 5300.0|2025-04-27 23:21:...|
|        65|         69|     Savings| 550.25|2025-04-27 23:21:...|
+----------+-----------+------------+-------+--------------------+
only showing top 5 rows
```

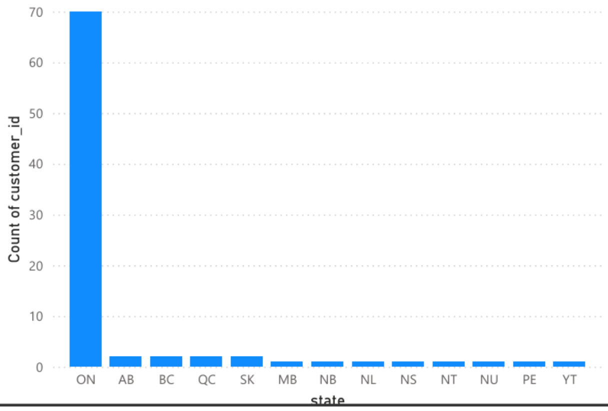
Populating Data on Power BI Dashboard:

No. of Total Customers

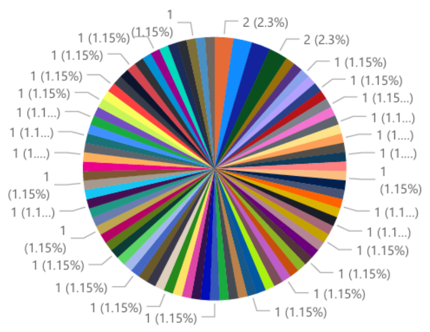
87

Count of customer_id

Number of customers State Wise



Count of customer_id by city


2 (2.3%)
2 (2.3%)
1 (1.15%)
1 (1.15...)
1 (1.1...)
1 (1....)
1 (1.1...)
1 (1.15%)
1 (1.15%)
city
Midland
North Bay
Orillia
Thunder Bay
Alliston
Angus
Aurora
Bala
Barrie
Beaverton
Belleville
Bracebridge