Data Engineering Project 2:

Transactions and Loan Data for Customers

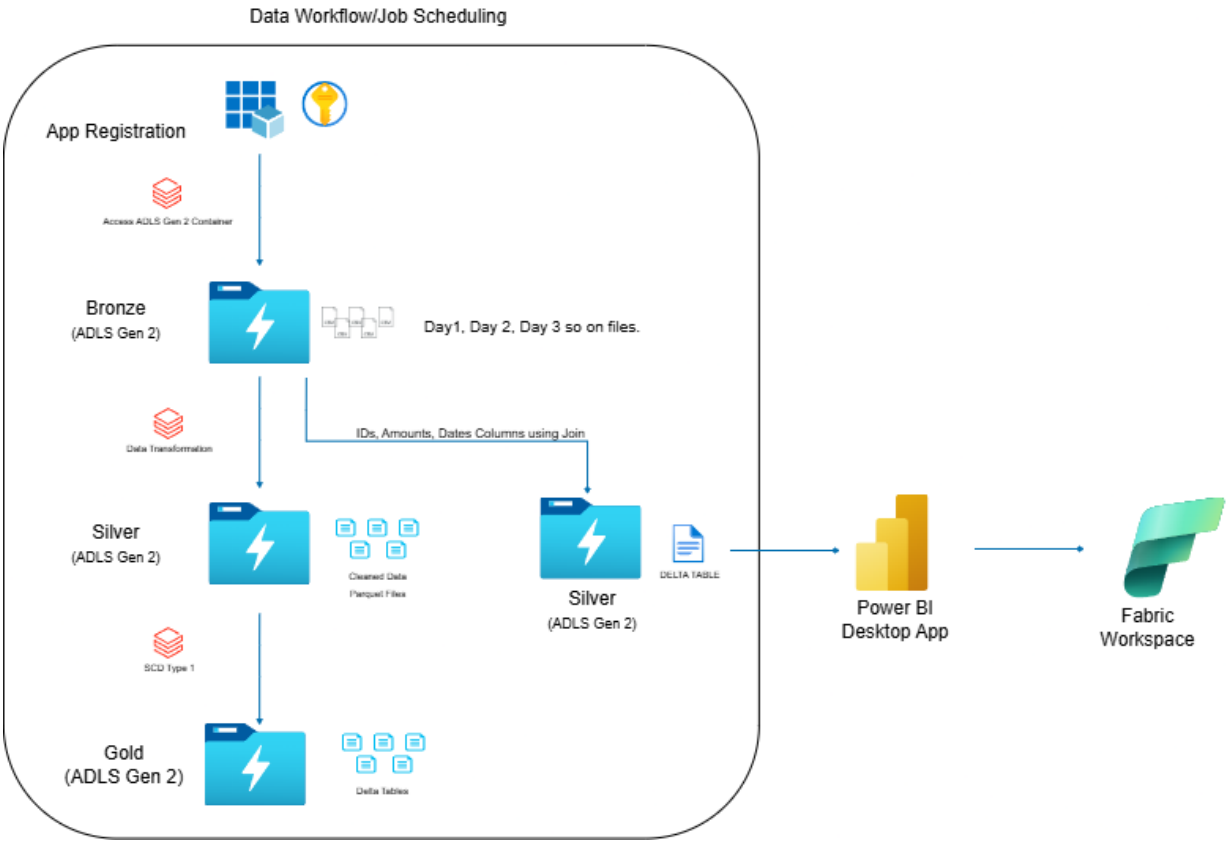
Table of Contents

Objective: 2
Project Architecture Diagram: 2
Step 1: App Registration 2
Step 2: Mount Point and Functions Notebooks 14
Step 3: Upload files into the ADLS Gen 2 Folder 17
Step 4: Bronze to Silver Notebook 19
Step 5: DELTA Table for Data Visualization 20
Step 6: Create a scheduled job for daily run 29
Step 7: Power BI Visualization on DELTA File in Silver Layer 35
Step 8: Publish it to the Fabric Workspace 38
Conclusion: 39
Points to remember: 40

Objective:

This project aims to design and implement a robust data pipeline for processing customer account data. This includes copying data from ADLS GEN2 (Bronze layer) and transforming the data in the Silver layer using Data bricks Notebooks and GOLD Storage into the SCD Type 1 Delta Table in ADLS GEN2. The pipeline aims to ensure efficient, accurate, and scalable data processing to support downstream analytics and reporting needs.

Project Architecture Diagram:



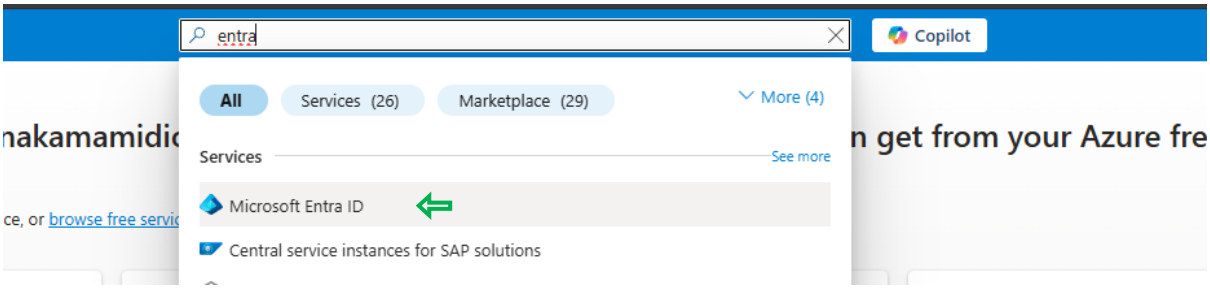
Step 1: App Registration

Go to **Microsoft Entra ID**

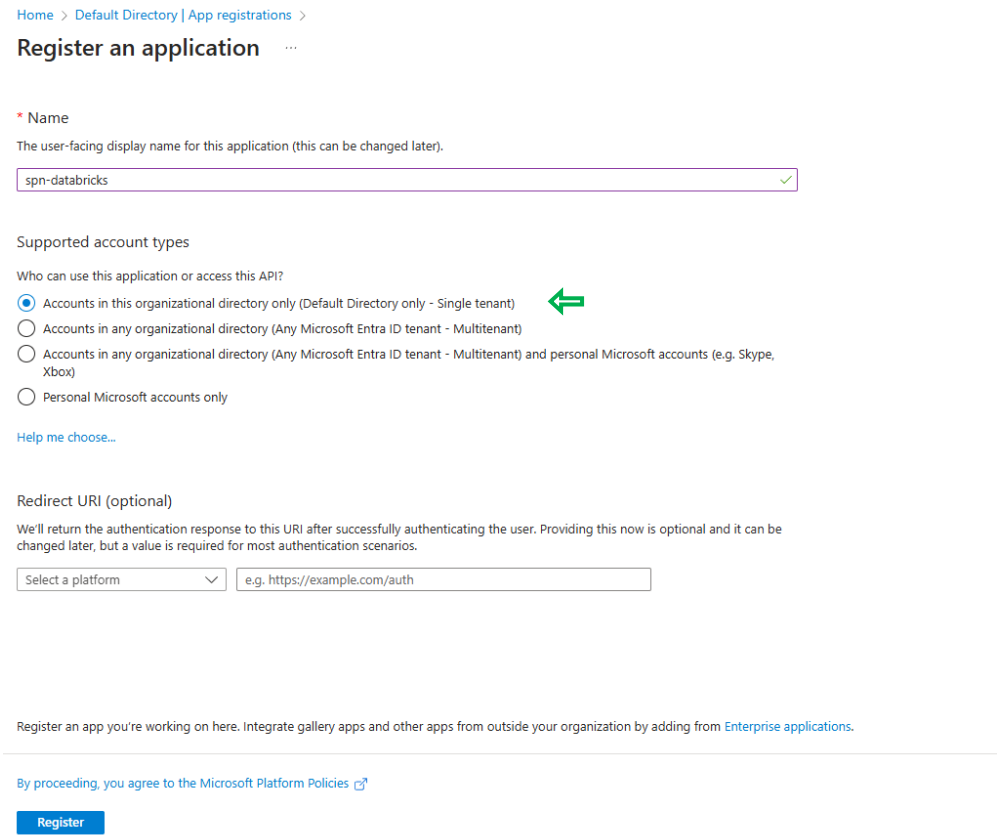
Microsoft Entra ID → Manage → App Registration → New Registration

Name: spn-databricks

Select: Single tenant

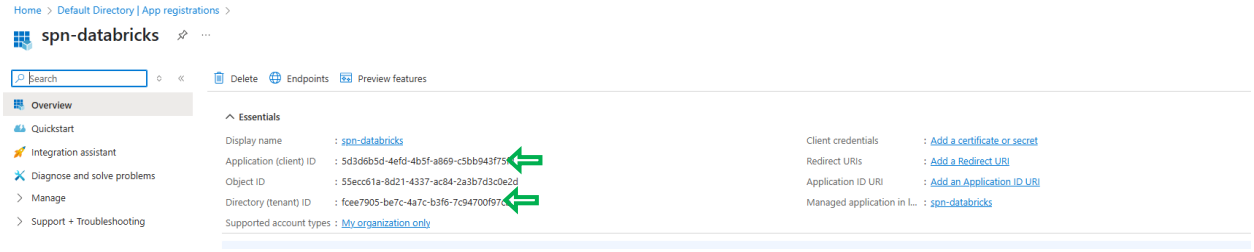


Click **Register**

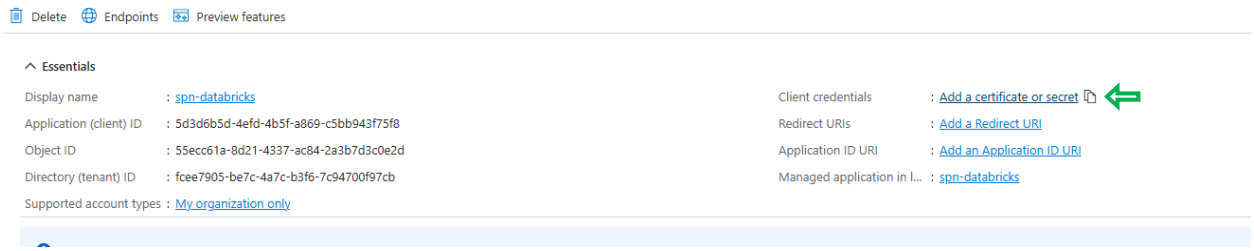


Here we need the application(client) ID and Directory(tenant) ID:

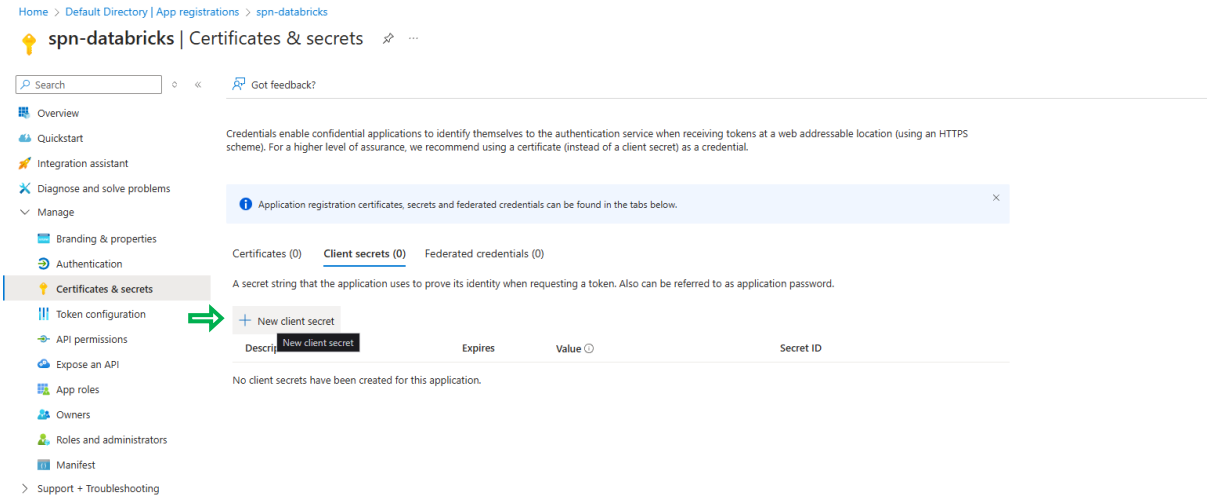
Copy these two IDS into Notepad, as we will need them later



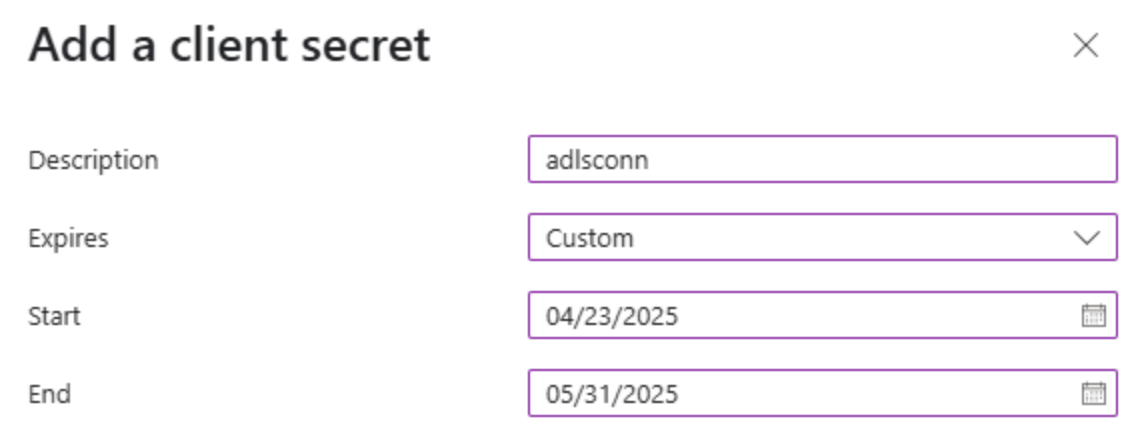
Click on **Client Credentials** as shown below



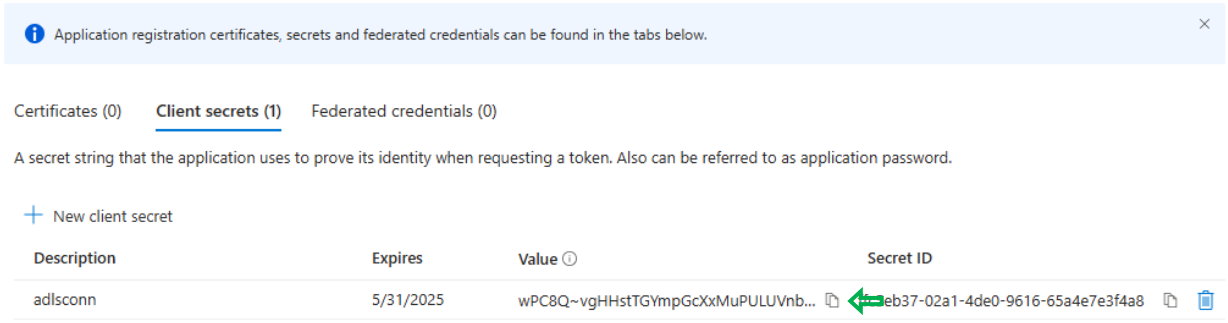
Click on New Client Secret



Write a description, Expires, Start and End as shown below

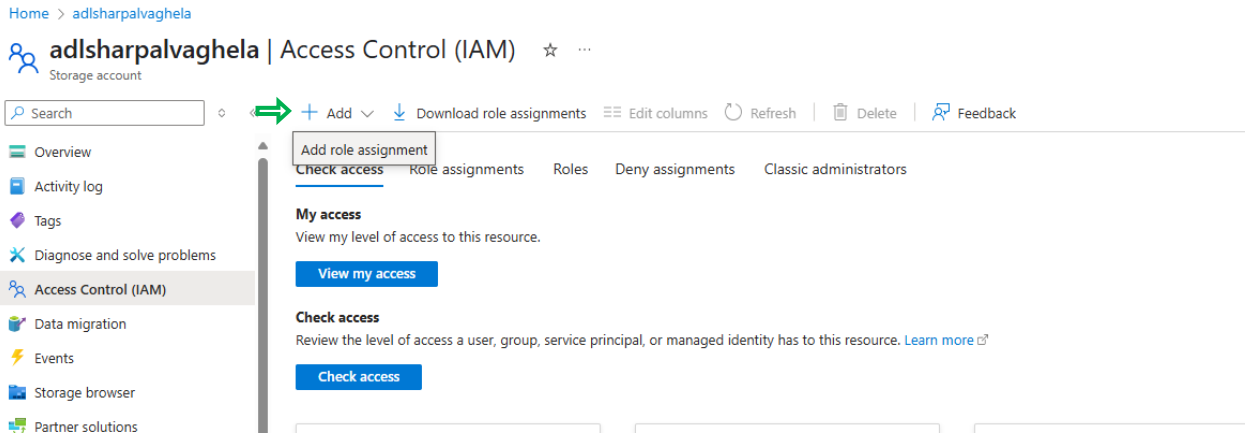


Copy the **Value** from this newly created secret, paste it into a notepad to use later on.

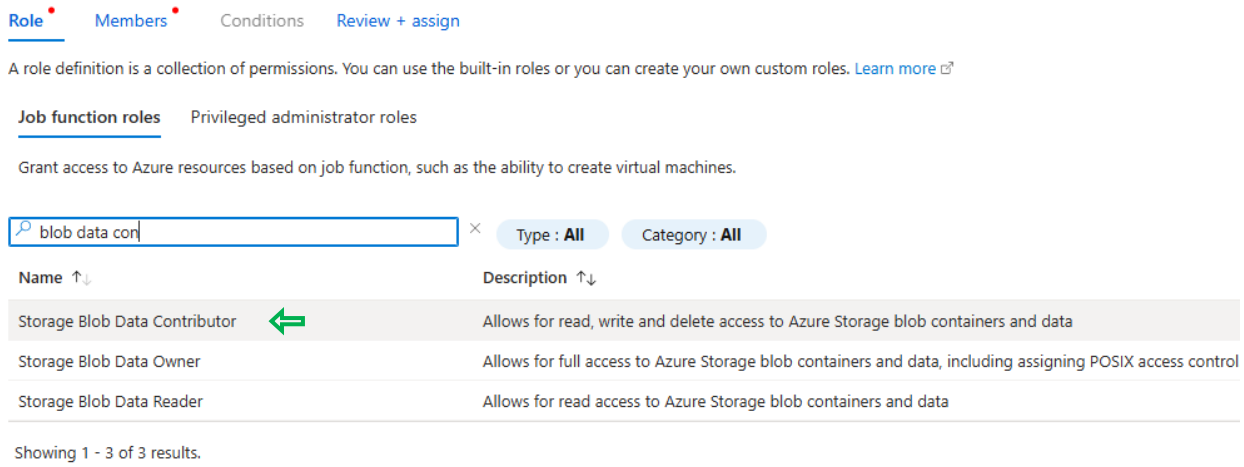


Go to the ADLS Gen 2 account:

Access Control (IAM) → Add role assignment

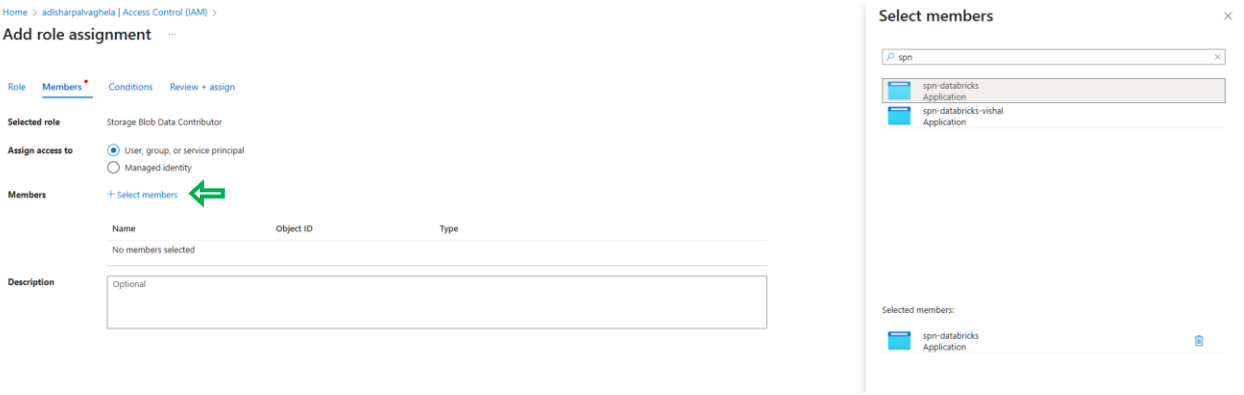


Select Storage Blob Data Contributor role

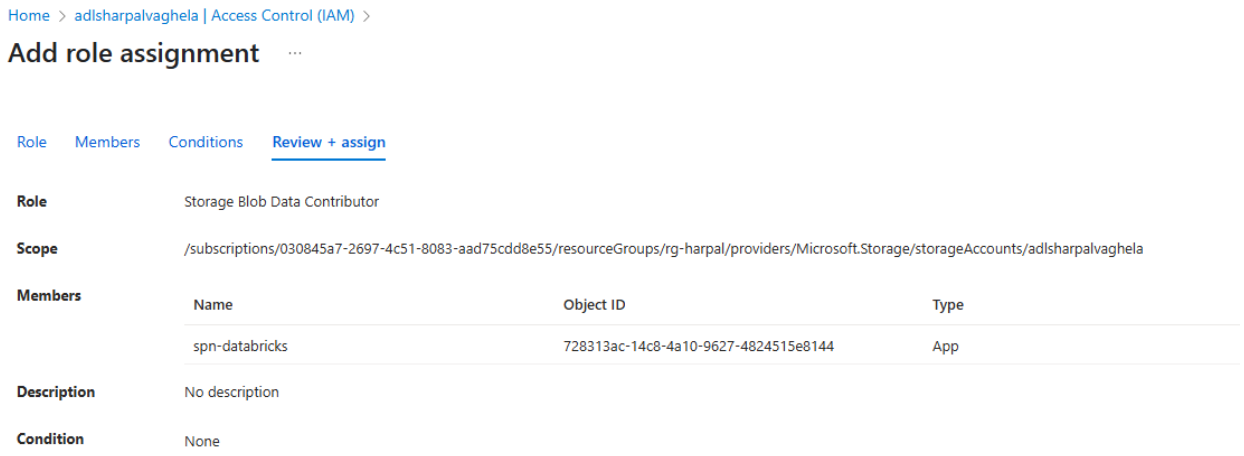


Members Tab: User, group or service principal

Select member: search spn-databricks



Review and Assign



❖ **Azure Key Vault**

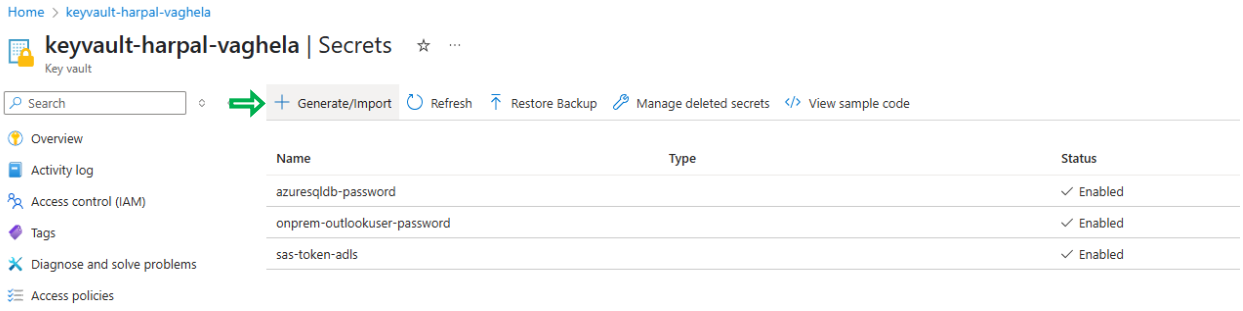
Azure Key Vault is a cloud service provided by Microsoft Azure that helps securely store and manage sensitive information such as keys, secrets, and certificates. It acts as a secure locker where you can store and control access to things like database connection strings, API keys, or passwords, which are crucial for your applications and services but should not be exposed or hard-coded in your code.

❖ **Key Features of Azure Key Vault:**

- Security: Azure Key Vault is designed to keep data secure using encryption, ensuring that sensitive information is never exposed.
- Access Control: It allows you to control who and what can access your keys and secrets. You can configure permissions specifically tailored to the needs of your organization.
- Audit Trails: It provides logging capabilities, which means you can monitor who accessed what information and when, adding an extra layer of security and compliance.

Go to **Key Vault** in the Azure Portal

Secrets → Generate/Import

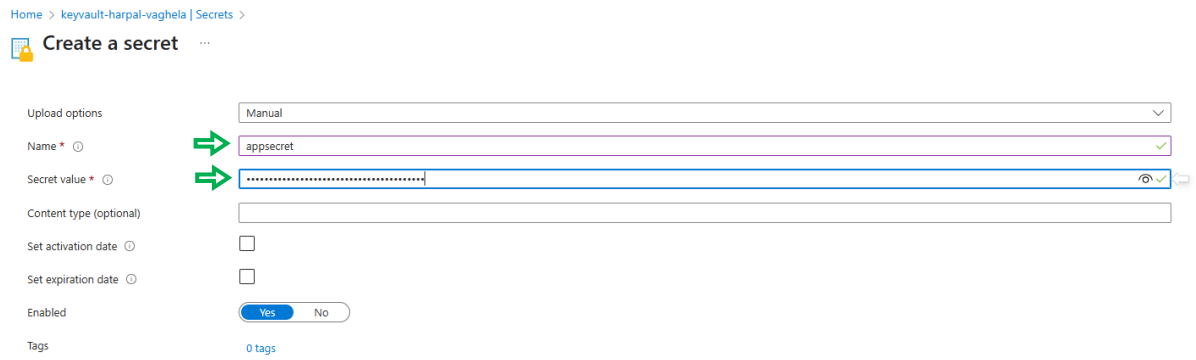


Name: appid

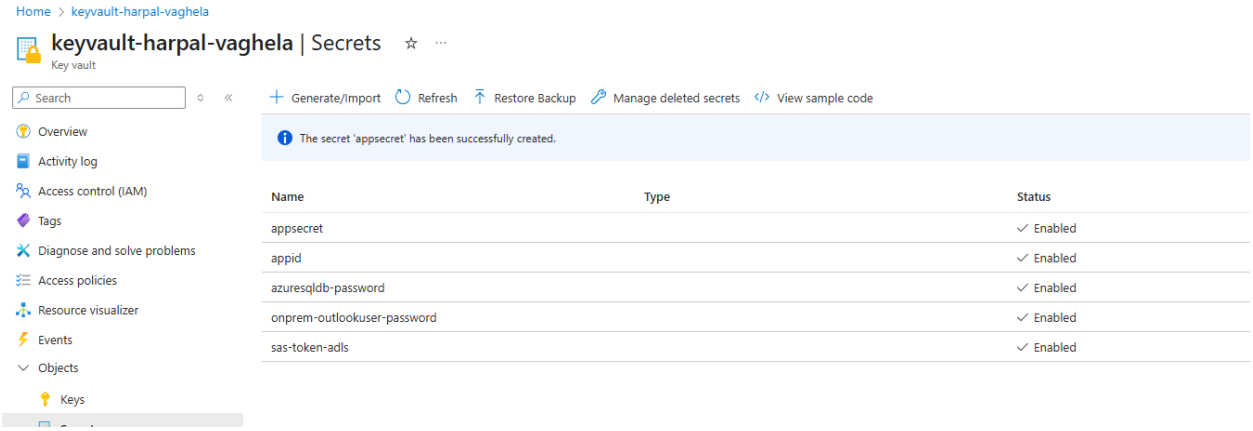
Select the appid from Notepad, and in the Secret value field, paste it there

Again, click on Generate/Import → Name: appsecret

Secret value, paste it here (client secret which we copied earlier while creating spn-databricks)

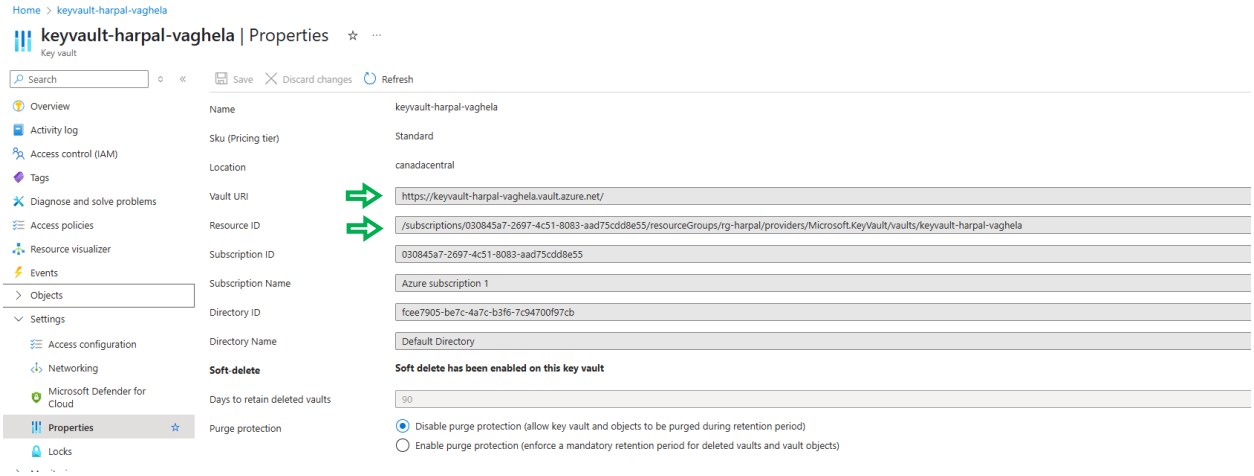


Keyvault -> Objects -> Secrets page will look like this

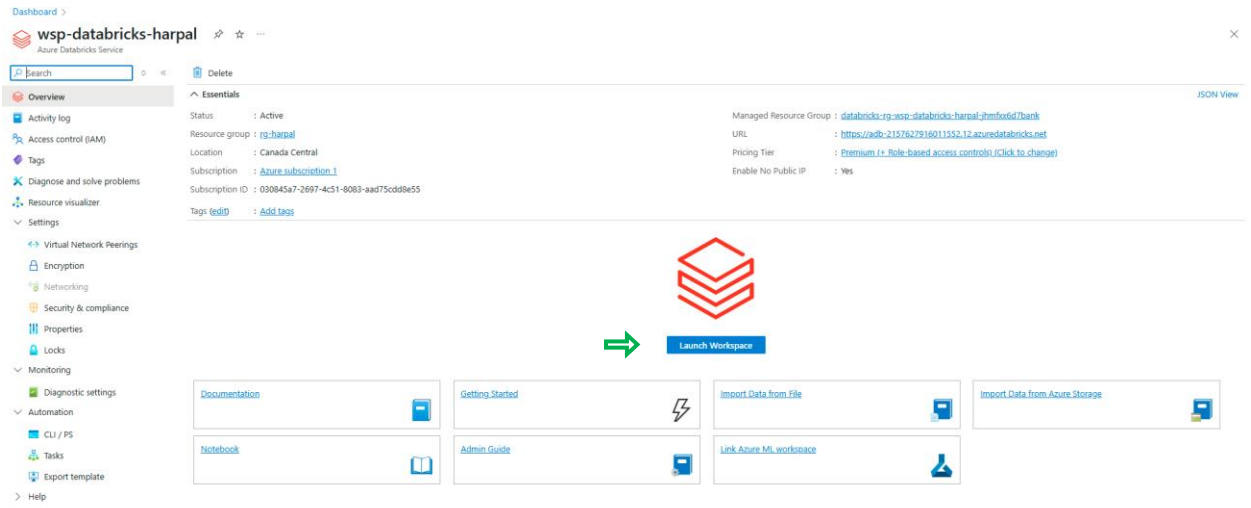


Go to Key Vault-> Properties

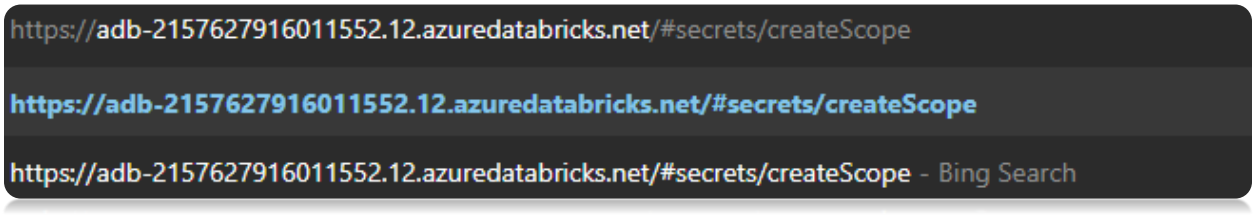
Here we need Vault URI and Resource ID, copy it to Notepad



Go to the Databricks Notebook tab in the browser



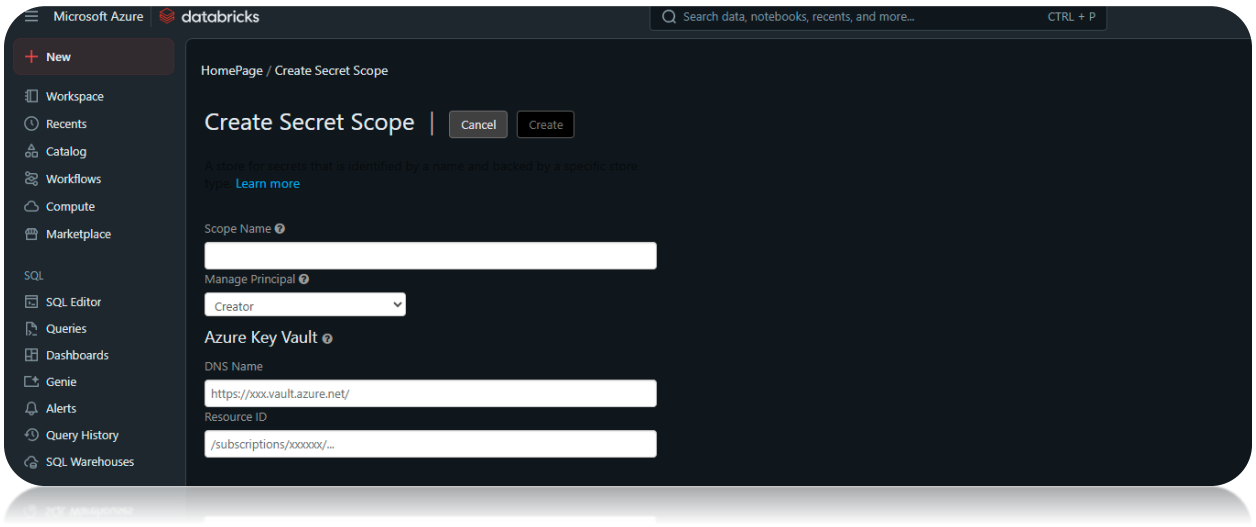
In the browser URL, remove the URL after the .net word, like this



And write this: #secrets/createScope

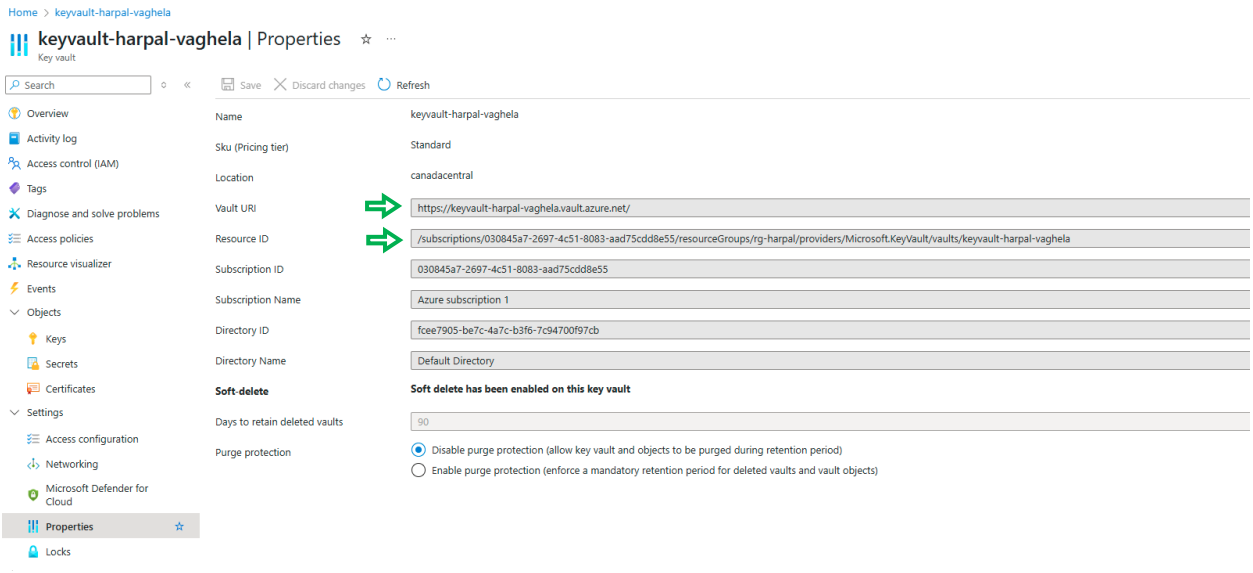
Press Enter on the keyboard

Page will be opened as shown below:

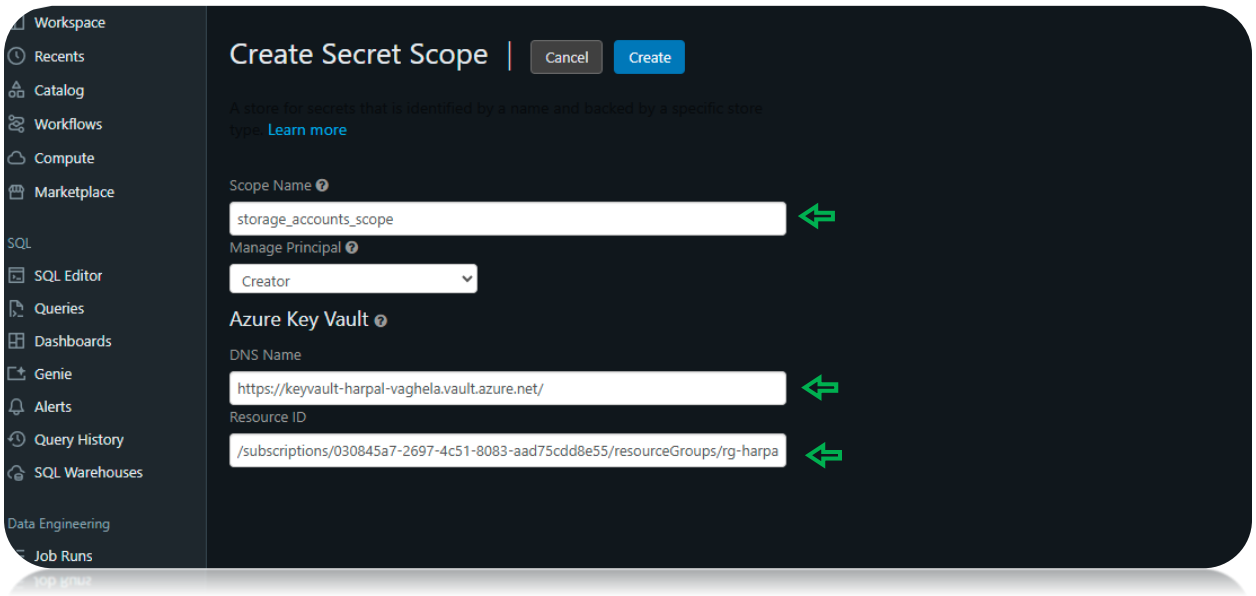


Write the Scope Name, also, we need the DNS Name and Resource ID here

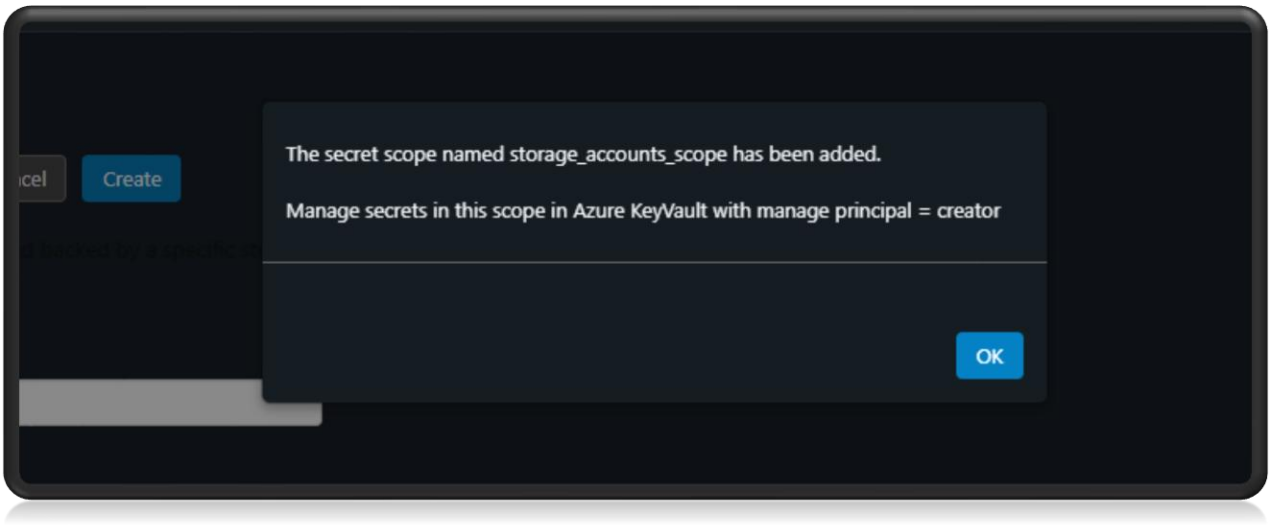
So we have to go to Key vault -> Properties tab and copy it to the scope window



Vault URI in DNS URL and Resource ID as shown below:



Click on Create



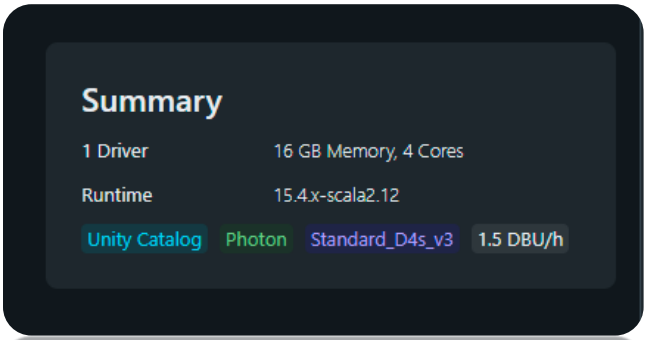
Go to Databricks Workspace:

Go to Compute -> Create Compute

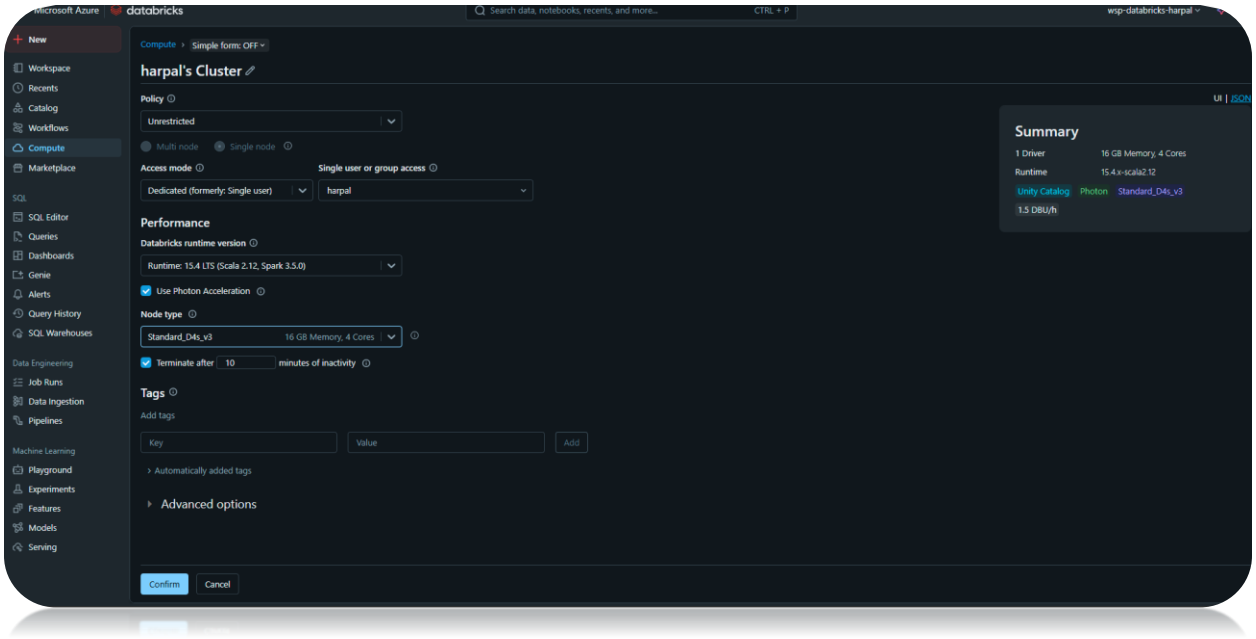
Use this configuration for creating a cluster

Policy: Unrestricted, Single Mode
Access mode: Dedicated (Single User)
Performance:

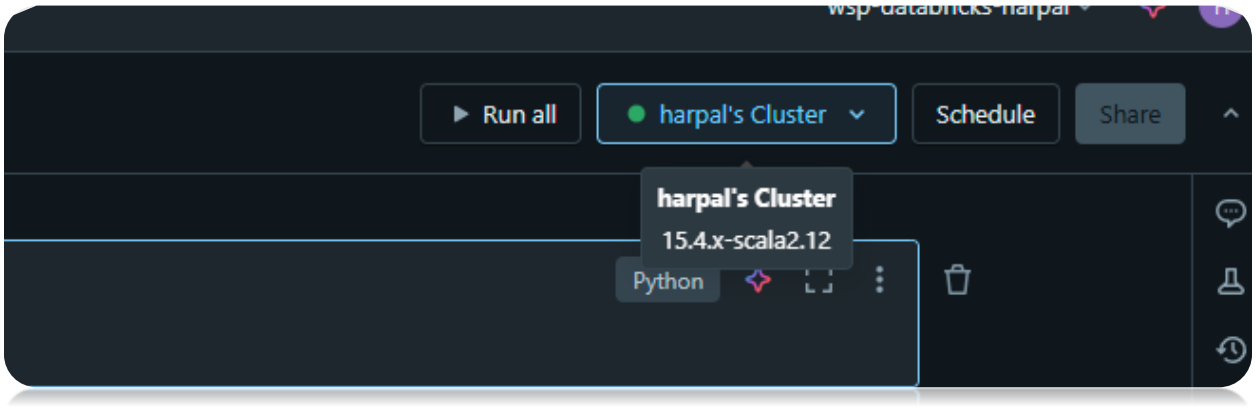
- Databricks runtime version:
  Runtime 15.4 LTS (Scala 2.12, Spark 3.5.0)
- Node Type: Standard_D4s_v3
- Terminate after 10 minutes of inactivity



Click on Create Compute



Create a new notebook and attach that cluster



Follow this URL to get the code for the mounting process

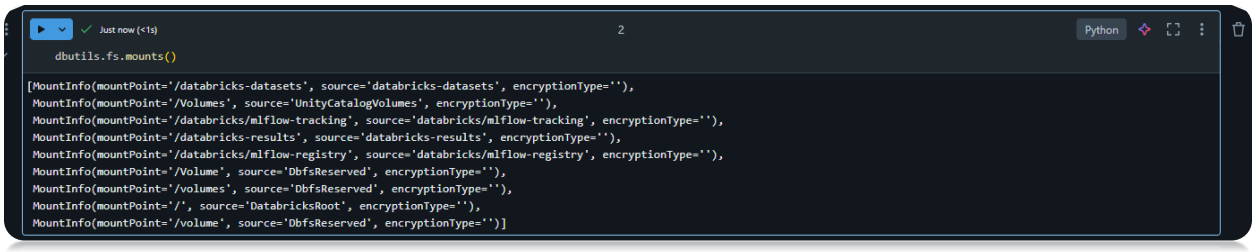
Code to mount ADLS Gen 2 in Databricks

https://learn.microsoft.com/en-us/azure/databricks/dbfs/mounts

Go to Databricks Notebook, attach and start the cluster

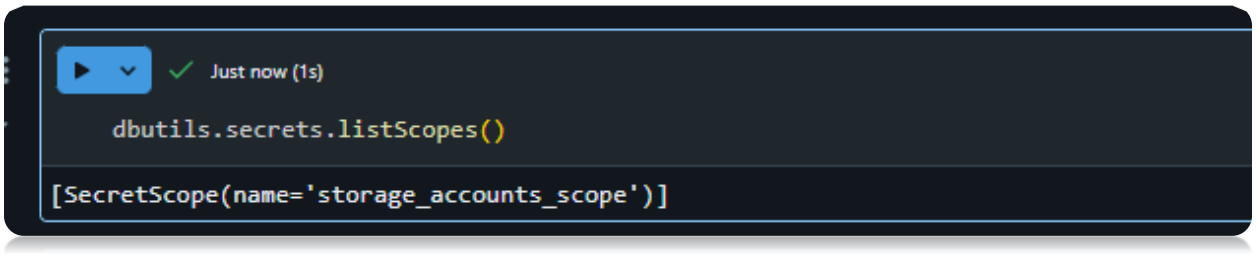
Check for all mounted locations using the command below

dbutils.fs.mounts()



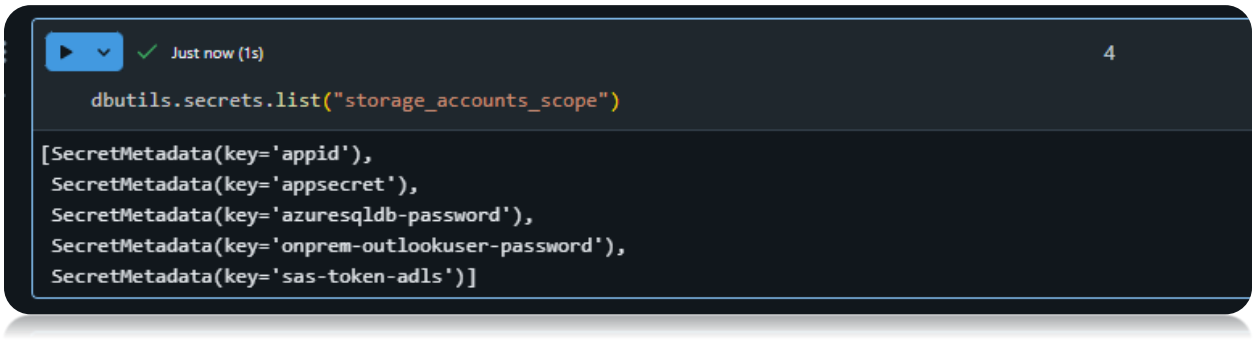
Check all secret scopes

dbutils.secrets.listScopes()

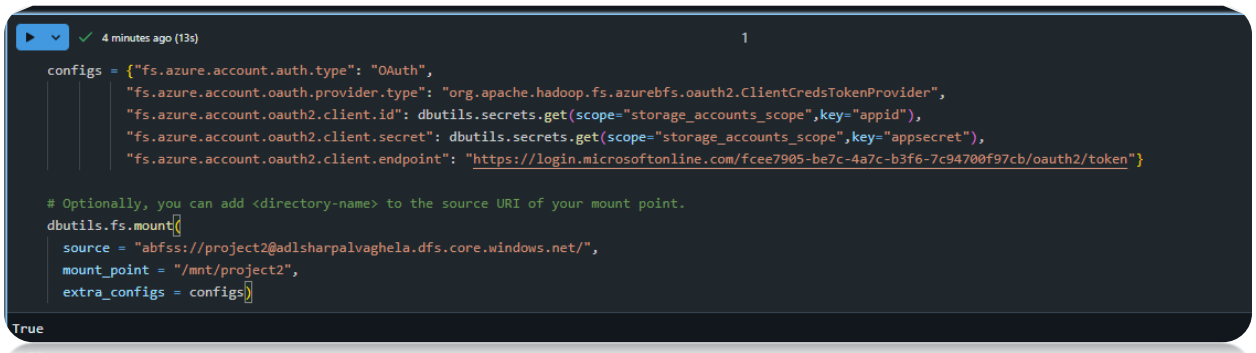


Execute this to check all secrets from the key vault

dbutils.secrets.list("storage_accounts_scope")



Make a change in that mount code and modify as below

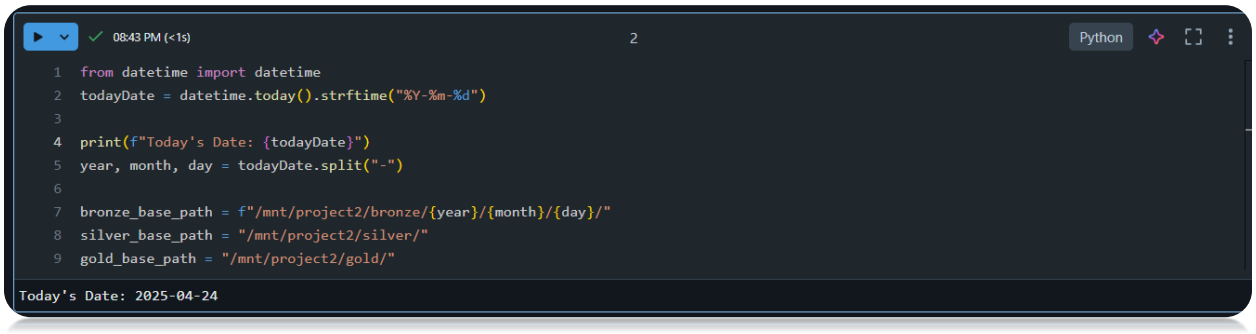


We will create a total of 3 notebook files:

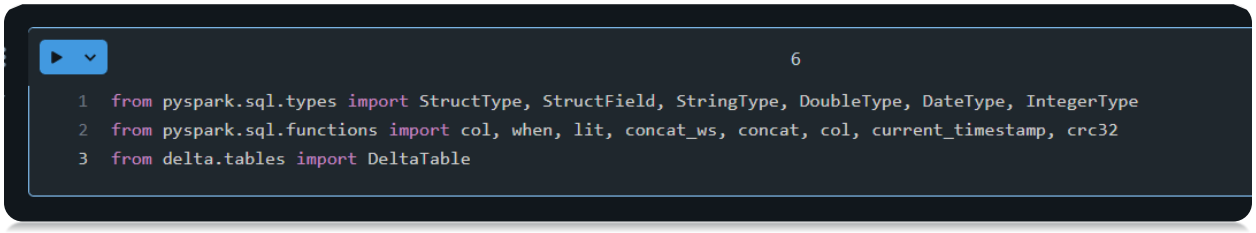
Let's go with the first one, in which we already have the mount code.

Step 2: Mount Point and Functions Notebooks

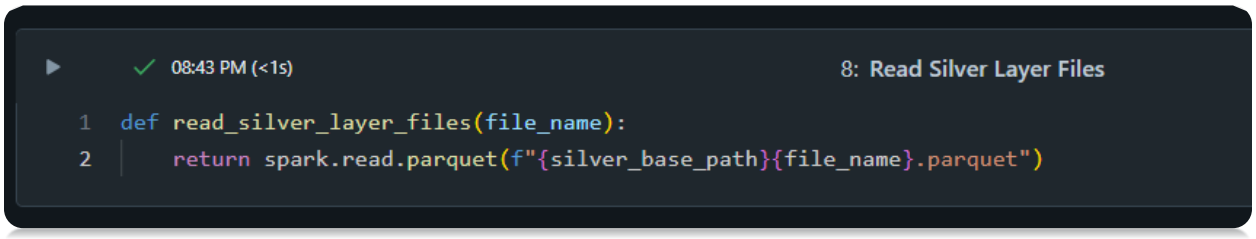
- Gets today's date in "YYYY-MM-DD" format using datetime.today().strftime.
- Splits the date into year, month, and day components.
- Builds a dynamic bronze layer path based on today's date.
- Sets static paths for silver and gold layers.



- Imports data types like StringType, DoubleType, DateType, and IntegerType to define DataFrame schemas.
- Imports functions like col, when, lit, concat_ws, concat, current_timestamp, and crc32 for data transformations.
- Imports DeltaTable to work with Delta Lasske tables for updates, merges, and upserts



- Defines a function read_silver_layer_files that takes file_name as input.
- Reads a Parquet file from the silver layer path using the given file name.
- Returns the loaded DataFrame for further processings



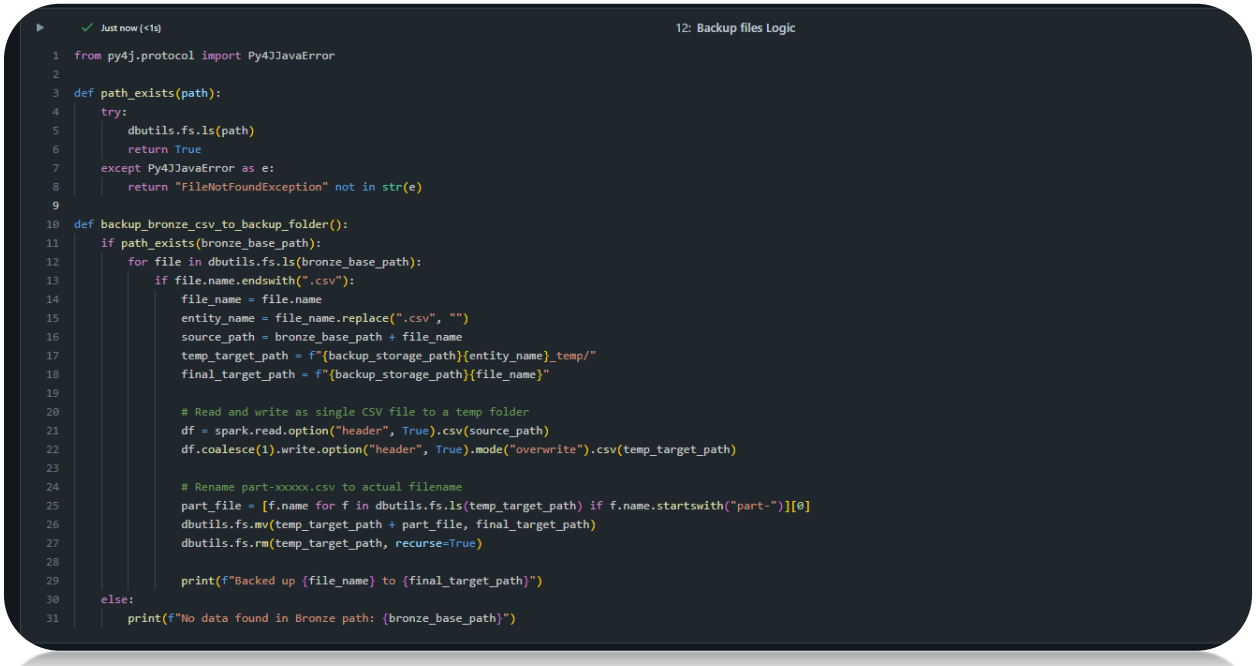
- Replaces .csv in the file name to create a base name for output.
- Writes the DataFrame as a single .parquet file by coalescing into one partition and saving to a temp path.
- Finds the actual .parquet part file generated in the temp folder.
- Moves and renames the part file to the final path in the silver layer.
- Deletes the temporary folder after moving the file to clean up.

- Defines a function get_schema that returns a specific schema based on the file name.
- Each schema is created using StructType with a list of StructField for column names and data types.
- Handles five known files: accounts.csv, customers.csv, loans.csv, loan_payments.csv, and transactions.csv.
- If an unknown file name is passed, the function raises an error to prevent incorrect schema assignment.

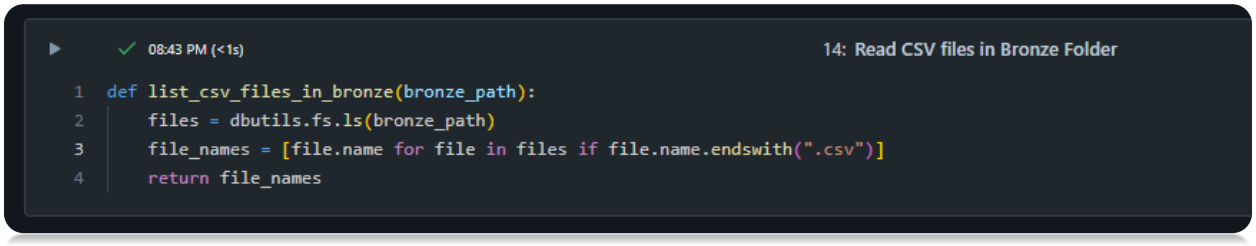
Backup File logic:

- path_exists checks if a given path exists without raising an error.
- The main function loops through .csv files in the bronze path if it exists.
- Each file is read into a DataFrame and written to a temporary backup folder as a single .csv.
- Renames the part-xxxxx.csv file to the original file name.
- Moves the renamed file to the final backup folder and deletes the temp folder.

- Prints the backup status for each file; skips if the bronze path doesn't exist.



- Defines a function list_csv_files_in_bronze to get all .csv files from tshe given bronze path.
- Uses dbutils.fs.ls to list all files in the directory.
- Filters only files that end with .csv and returns a list of matching file names.



- filter_nulls: Filters out rows where any of the specified ID columns are null.
- replace_nulls_with_defaults: Replaces nulls in specified columns with default values using a dictionary.

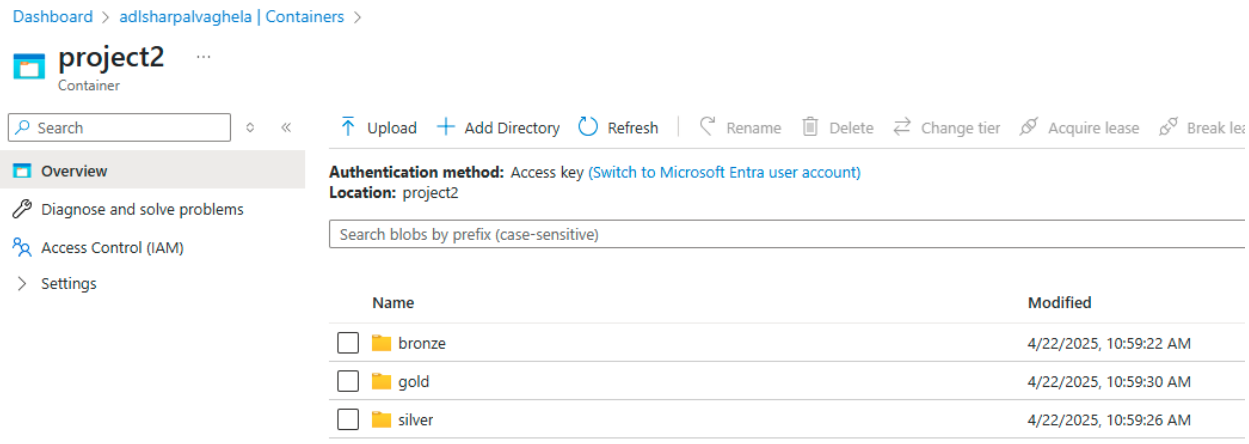
- remove_duplicates: Removes duplicate rows based on the given ID columns.s
- Each function returns a cleaned DataFrame for the next transformation step.

```
08:43 PM (<1s)                                                    16
1  from pyspark.sql.functions import col, when, lit
2
3  #------------------------------Filter Nulls-------------------------
4  def filter_nulls(df, id_cols):
5      for col_name in id_cols:
6          df = df.filter(col(col_name).isNotNull())
7      return df
8
9  #------------------------------Replace Nulls-------------------------
10 def replace_nulls_with_defaults(df, transform_cols_defaults):
11     for col_name, default_val in transform_cols_defaults.items():
12         df = df.withColumn(col_name, when(col(col_name).isNull(), lit(default_val)).otherwise(col(col_name)))
13     return df
14
15 #------------------------------Remove Duplicates-------------------------
16 def remove_duplicates(df, id_cols):
17     return df.dropDuplicates(id_cols)
18
```

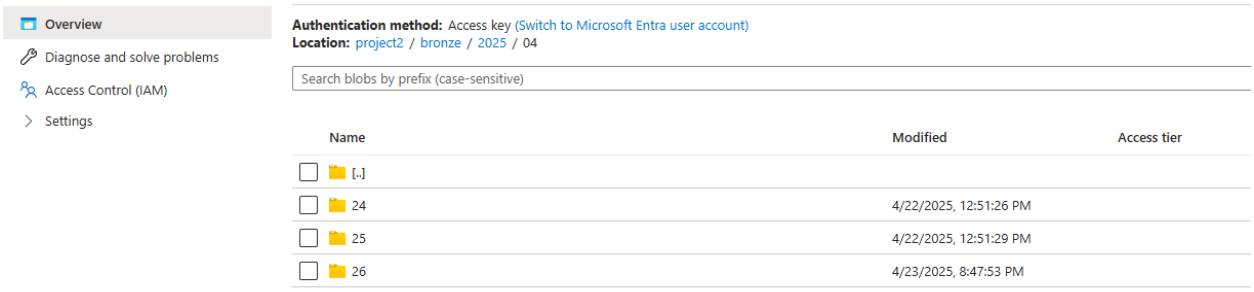
Step 3: Upload files into the ADLS Gen 2 Folder

ADLS Gen 2 folders:

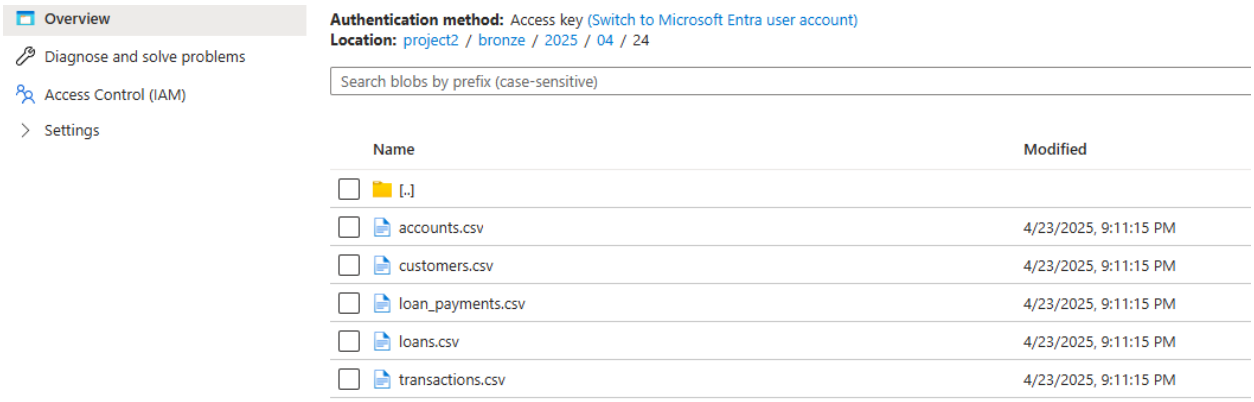
Create Bronze, Silver and Gold folders in this storage account



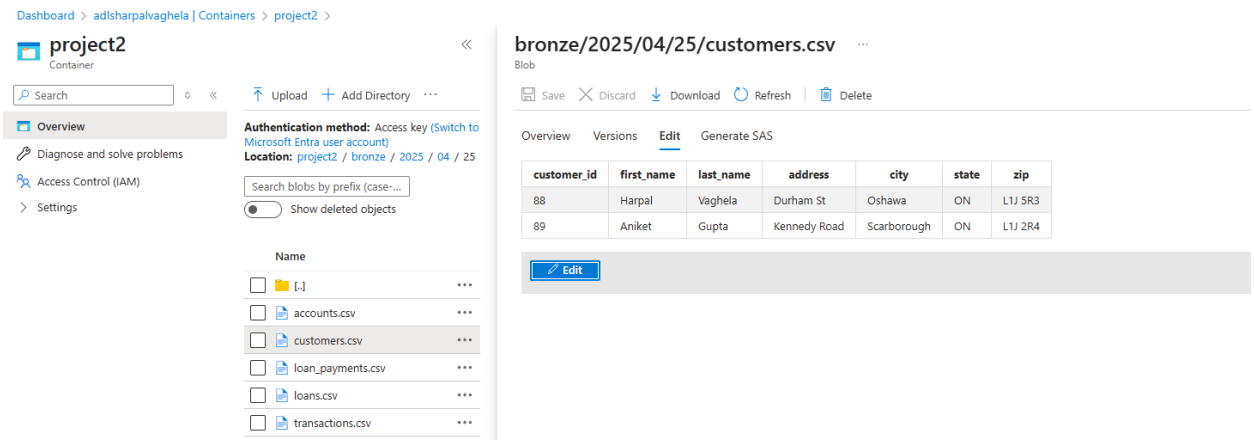
In the Bronze Folder in ADLS Gen 2, we have created a day-wise folder to upload the raw data files



24 Folder Data:



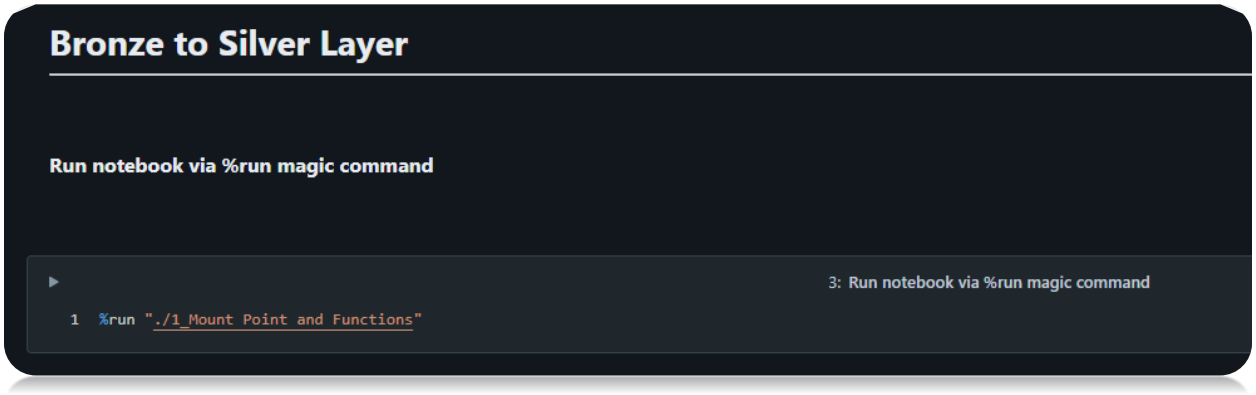
25 Folder Data with some updated values and new records (showing data for the customers CSV file):



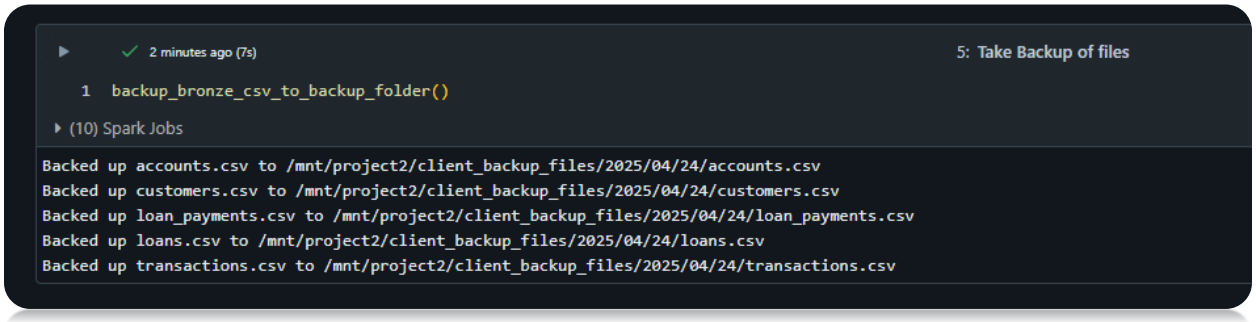
Step 4: Bronze to Silver Notebook

Let's write code into the bronze to silver notebook now (this is our second notebook)

- %run is a Databricks notebook command used to include and run another notebook's code.
- It imports all variables, functions, and logic defined in that notebook into the current one.



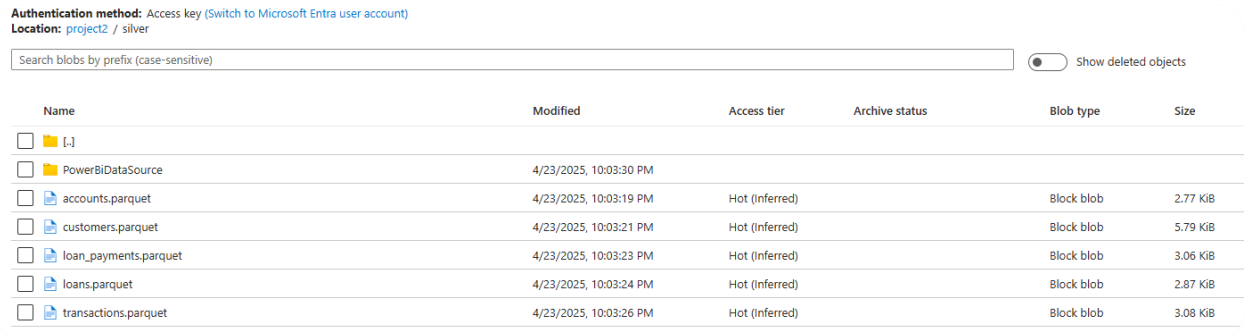
- Calling the Backup function to take all files backup to a different folder in csv files



- Retrieves a list of all .csv files from the bronze layer and prints them.
- For each file, reads it with an assigned schema and header using Spark.
- Based on the file name, defines primary ID columns and default values for null replacements.
- Applies three transformations: filters out rows with null IDs, replaces other nulls with defaults, and removes duplicates.
- Writes the cleaned data as a single .parquet file into the silver layer using the helper function.

```python
# 5: Assign Schema, Transformation, and Store into Silver Layer
file_name_list = list_csv_files_in_bronze(bronze_base_path)
print(f"File Name List : {file_name_list}")

# Loop through each file
cleaned_dfs = []

for file_name in file_name_list:

    # Get schema for the file
    schema = get_schema(file_name)

    # Read CSV with header and schema
    df = spark.read.format("csv") \
        .option("header", True) \
        .schema(schema) \
        .load(f"{bronze_base_path}{file_name}")

   # Define id columns and default values for each file
    if file_name == "customers.csv":
        id_cols = ["customer_id"]
        defaults = {"city": "Unknown", "state": "Unknown", "zip": "Unknown"}

    elif file_name == "accounts.csv":
        id_cols = ["account_id", "customer_id"]
        defaults = {"balance": 0.0}

    elif file_name == "loans.csv":
        id_cols = ["loan_id", "customer_id"]
        defaults = {"loan_amount": 0.0}

    elif file_name == "transactions.csv":
        id_cols = ["transaction_id", "account_id"]
        defaults = {"transaction_amount": 0.0, "transaction_type": "Unknown"}

    elif file_name == "loan_payments.csv":
        id_cols = ["payment_id", "loan_id"]
        defaults = {"payment_amount": 0}

    else:
        print(f"Skipping unknown file: {file_name}")
        continue

    # Apply transformations/data cleaning
    df_filtered = filter_nulls(df, id_cols)
    df_transformed = replace_nulls_with_defaults(df_filtered, defaults)
    df_final = remove_duplicates(df_transformed, id_cols)

    #write file to silver layer
    write_single_parquet_file(df_final, file_name, silver_base_path)

    #show preview
    #df_final.show(5)
```

Silver Folder data in ADLS Gen 2



Step 5: DELTA Table for Data Visualization

- Reads five cleaned .parquet files from the silver layer and loads them into individual DataFrames.
- Joins accounts with customers using customer_id.
- Joins the result with transactions using account_id.

- Further joins with loans on customer_id, then with loan_payments on loan_id.
- Combines all related data into a single DataFrame df_combined for visualization or reporting.

8: Inner Join Condition

```
#Read all files from silver layer and store in a dataframe
df_accounts = read_silver_layer_files("accounts")
df_customers = read_silver_layer_files("customers")
df_loans = read_silver_layer_files("loans")
df_loan_payments = read_silver_layer_files("loan_payments")
df_transactions = read_silver_layer_files("transactions")

df_combined = (
    df_accounts.join(df_customers, on="customer_id", how="inner")
        .join(df_transactions, on="account_id", how="inner")
        .join(df_loans, on="customer_id", how="inner")
        .join(df_loan_payments, on="loan_id", how="inner")
)

# Show result
#display(df_combined)
```

- Creates a new DataFrame df_selected_columns with only the required fields.
- Renames the selected columns using alias to more readable names.

10

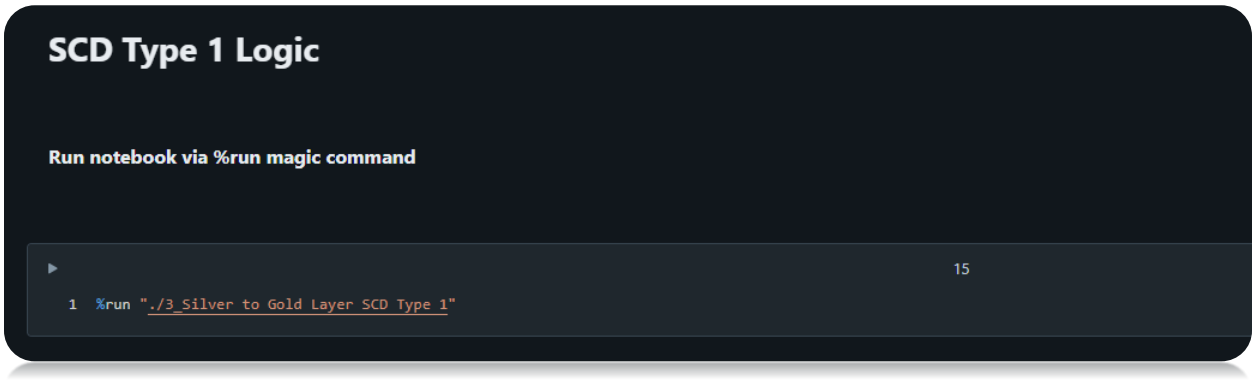
```
# Select Required Columns

df_selected_columns = df_combined.select(
    col("account_id").alias("Account_ID"),
    col("customer_id").alias("Customer_ID"),
    col("loan_id").alias("Loan_ID"),
    col("payment_id").alias("Payment_ID"),
    col("transaction_id").alias("Transaction_ID"),
    col("balance").alias("Account_Balance"),
    col("payment_amount").alias("Payment_Amount"),
    col("loan_amount").alias("Loan_Amount"),
    col("transaction_amount").alias("Transaction_Amount"),
    col("payment_date").alias("Payment_Date"),
    col("transaction_date").alias("Transaction_Date")
)
#display(df_selected_columns)
```

- Writes df_selected_columns to the silver layer in Delta format for efficient querying and updates.
- Uses mode("overwrite") to replace any existing data at the path.
- Saves the output to the folder PowerBiDataSource inside the silver layer path.

12: Store it in DELTA format in Silver Layer

```
df_selected_columns.write.format("delta").mode("overwrite").save(f"{silver_base_path}PowerBiDataSource")
```

- Runs the notebook "./3_Silver to Gold Layer SCD Type 1" inside the current notebook.
- Perform SCD Type 1 logic and write cleaned records to the gold layer.



Let's go with a 3rd notebook now

Step 5: SCD Type 1 Logic

- Executes a SQL command using %sql magic in Databricks.
- Creates a new database named bankdb under the hive_metastore if it doesn't already exist.

- Creates a Delta table named transactions in the bankdb database if it doesn't exist.
- Defines schema with fields like transaction_id, account_id, and audit columns such as CreatedDate and HashKey.
- Uses the DELTA format for efficient updates and versioning.
- Specifies the physical storage location in the gold layer: /mnt/project2/gold/transactions.

- Creates a Delta table named loan_payments in the bankdb database if it doesn't already exist.
- Defines the table schema with columns like payment_id, loan_id, payment_amount, and audit fields.
- Uses DELTA format to enable ACID transactions and efficient data updates.
- Stores the table data at the specified gold layer path: /mnt/project2/gold/loan_payments.

- Creates a Delta table named loans in the bankdb database if it doesn't already exist.
- Defines schema with loan-related fields and audit columns like CreatedDate, UpdatedBy, and HashKey.
- Uses DELTA format for optimized performance and support for upserts.
- Stores the data in the gold layer path: /mnt/project2/gold/loans.

- Creates a Delta table named customers in the bankdb database if it doesn't already exist.
- Defines schema with customer details and audit fields for tracking changes.
- Uses DELTA format to enable efficient updates and data consistency.
- Saves the table data in the gold layer at /mnt/project2/gold/customers.

- Creates a Delta table named accounts in the bankdb database if it doesn't already exist.
- Defines columns for account info along with audit fields like CreatedBy and HashKey.
- Uses DELTA format to support versioning, ACID compliance, and upserts.
- Stores the table at the gold layer path: /mnt/project2/gold/accounts.

Check for new Data and load it into the tables

- Reads the silver data and adds audit columns (CreatedDate, UpdatedDate, CreatedBy, UpdatedBy) and a HashKey for change detection using crc32.
- Performs an anti-join with the gold table to identify new or changed records (based on join_key and HashKey).
- Prepares update_set and insert_values dictionaries for the merge operation, applying updates or inserts accordingly.
- Runs a Delta Lake merge using whenMatchedUpdate and whenNotMatchedInsert to apply SCD Type 1 logic (overwrite on change, insert if new).

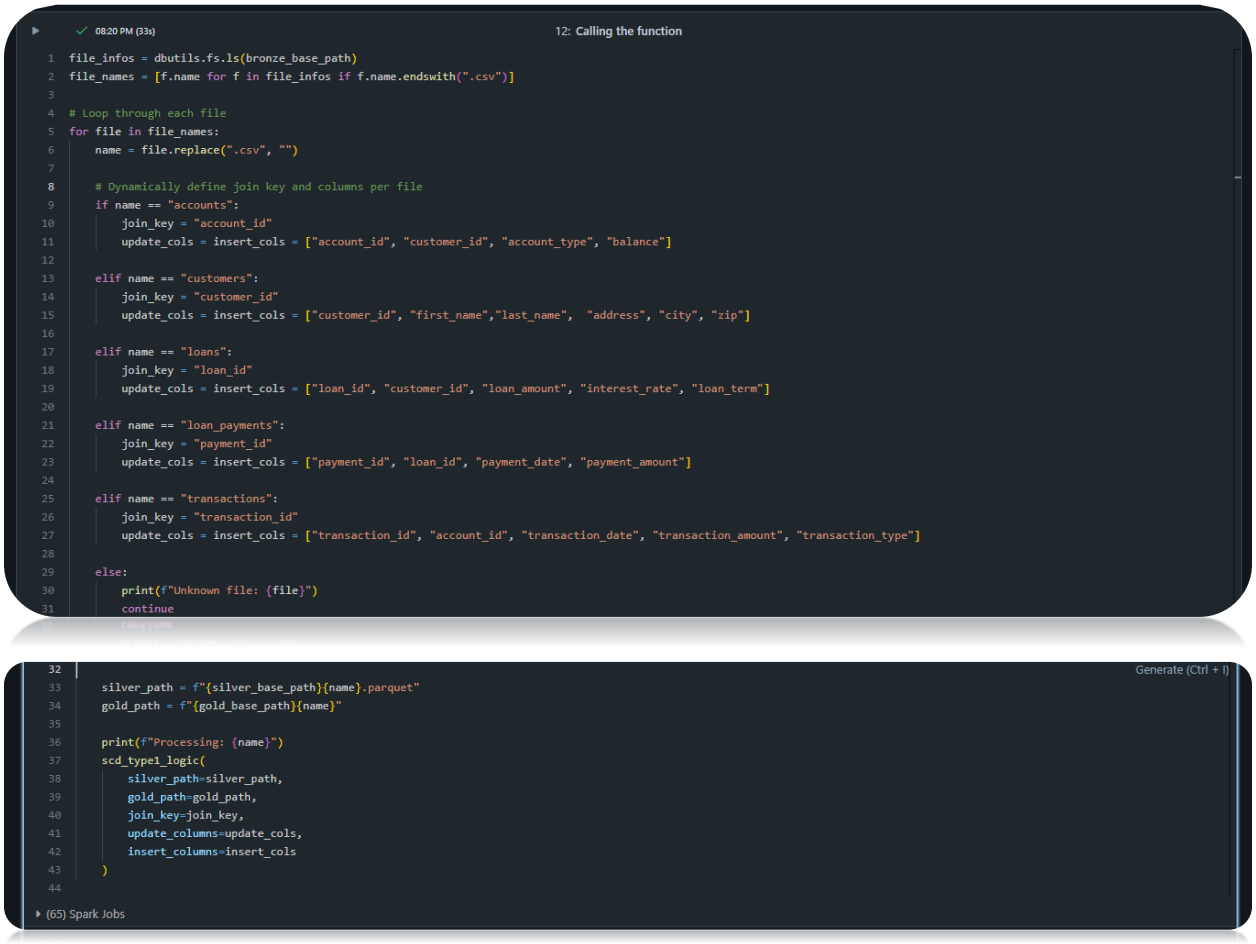
11:04 PM (<1s) 11: Check for new Data and load it into the tables

```python
def scd_type1_logic(
    silver_path: str,
    gold_path: str,
    join_key: str,
    update_columns: list,
    insert_columns: list,
    created_by: str = "databricks",
    updated_by: str = "databricks-updated"
):
    try:
        # Load Silver layer and add hash
        df_silver = spark.read.format("parquet").load(silver_path)
        from pyspark.sql.functions import current_timestamp, lit
        df_hashed = (
            df_silver
            .withColumn("HashKey", crc32(concat_ws("|", *df_silver.columns)))
            .withColumn("CreatedDate", current_timestamp())
            .withColumn("UpdatedDate", current_timestamp())
            .withColumn("CreatedBy", lit(created_by))
            .withColumn("UpdatedBy", lit(created_by))
        )

        # Load existing Gold table
        target_table = DeltaTable.forPath(spark, gold_path)

        # Anti-join to find new/changed rows
        df_src = (
        df_hashed.alias("src")
        .join(
            target_table.toDF().alias("tgt"),
            col(f"src.{join_key}") == col(f"tgt.{join_key}"),
            "left"
        )
        .filter(
            (col(f"tgt.{join_key}").isNull()) |  # New records
            (col("src.HashKey") != col("tgt.HashKey"))  # Changed records
        )
        .select("src.*")
        )
```

```python
        if df_src.count() > 0:
            # Update dictionary
            update_set = {colname: f"source.{colname}" for colname in update_columns}
            update_set["UpdatedDate"] = current_timestamp()
            update_set["UpdatedBy"] = lit(updated_by)
            update_set["HashKey"] = "source.HashKey"

            # Insert dictionary
            insert_values = {colname: f"source.{colname}" for colname in insert_columns}
            insert_values.update({
                "CreatedDate": "source.CreatedDate",
                "UpdatedDate": current_timestamp(),
                "CreatedBy": "source.CreatedBy",
                "UpdatedBy": lit(created_by),
                "HashKey": "source.HashKey"
            })

            # Perform merge
            target_table.alias("target").merge(
                df_src.alias("source"),
                f"target.{join_key} = source.{join_key}"
            ).whenMatchedUpdate(
                set = update_set
            ).whenNotMatchedInsert(
                values = insert_values
            ).execute()

            print(f" --> Merge completed for: {gold_path}")
        else:
            print(f"No changes found for: {gold_path}")

    except Exception as e:
        print(f"Error in SCD Type 1 process: {str(e)}")
```

Calling the function: scd_type1_logic

- Lists all .csv files from the bronze path and extracts their base names (e.g., accounts, customers).
- For each known file type, sets the join_key and relevant columns for update and insert operations.
- Builds full paths for silver (input Parquet file) and gold (Delta table output) layers dynamically.
- Calls scd_type1_logic function for each dataset to apply SCD Type 1 logic and update the gold layer.
- Skips any unknown file types and logs a message for awareness.



```
08:20 PM (33s)                                                    12: Calling the function

file_infos = dbutils.fs.ls(bronze_base_path)
file_names = [f.name for f in file_infos if f.name.endswith(".csv")]

# Loop through each file
for file in file_names:
    name = file.replace(".csv", "")

    # Dynamically define join key and columns per file
    if name == "accounts":
        join_key = "account_id"
        update_cols = insert_cols = ["account_id", "customer_id", "account_type", "balance"]

    elif name == "customers":
        join_key = "customer_id"
        update_cols = insert_cols = ["customer_id", "first_name","last_name",  "address", "city", "zip"]

    elif name == "loans":
        join_key = "loan_id"
        update_cols = insert_cols = ["loan_id", "customer_id", "loan_amount", "interest_rate", "loan_term"]

    elif name == "loan_payments":
        join_key = "payment_id"
        update_cols = insert_cols = ["payment_id", "loan_id", "payment_date", "payment_amount"]

    elif name == "transactions":
        join_key = "transaction_id"
        update_cols = insert_cols = ["transaction_id", "account_id", "transaction_date", "transaction_amount", "transaction_type"]

    else:
        print(f"Unknown file: {file}")
        continue
```



```
                                                                                  Generate (Ctrl + I)
    silver_path = f"{silver_base_path}{name}.parquet"
    gold_path = f"{gold_base_path}{name}"

    print(f"Processing: {name}")
    scd_type1_logic(
        silver_path=silver_path,
        gold_path=gold_path,
        join_key=join_key,
        update_columns=update_cols,
        insert_columns=insert_cols
    )

(65) Spark Jobs
```

Displaying the output of the accounts table:

```
                                                                    13

df_gold_accounts = spark.read.format("delta").option("header", True).load(f"/mnt/project2/gold/accounts")
print("---------------------------------accounts.csv file Data---------------------------------")
display(df_gold_accounts)
```

Displaying the output of the customers table:

```
14
df_gold_customers = spark.read.format("delta").option("header", True).load(f"{gold_base_path}customers")
print("--------------------------------customers.csv file Data--------------------------------")
display(df_gold_customers)
```

Displaying the output of the loans table:

```
15
df_gold_loans = spark.read.format("delta").option("header", True).load(f"{gold_base_path}loans")
print("--------------------------------loans.csv file Data--------------------------------------")
display(df_gold_loans)
```

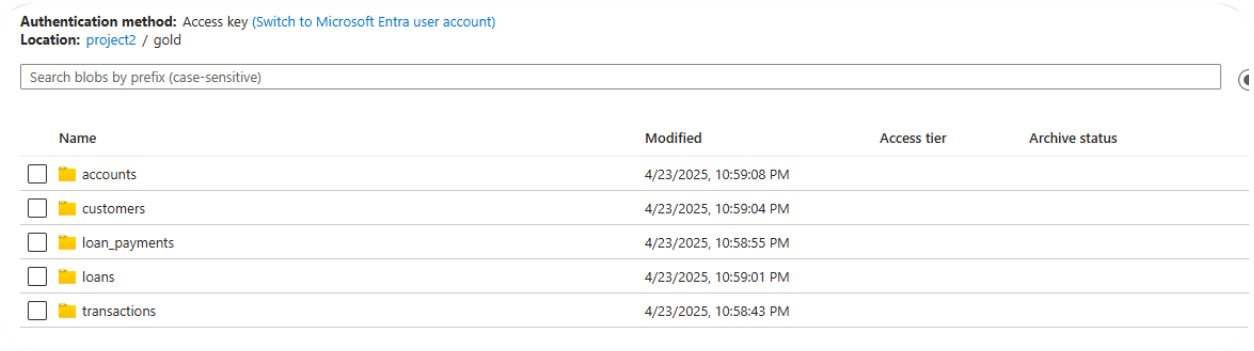
Displaying the output of the loan_payments table:

```
16
df_gold_loan_payments = spark.read.format("delta").option("header", True).load(f"{gold_base_path}loan_payments")
print("-----------------------------loan_payments.csv file Data--------------------------------")
display(df_gold_loan_payments)
```

Displaying the output of the transactions table:

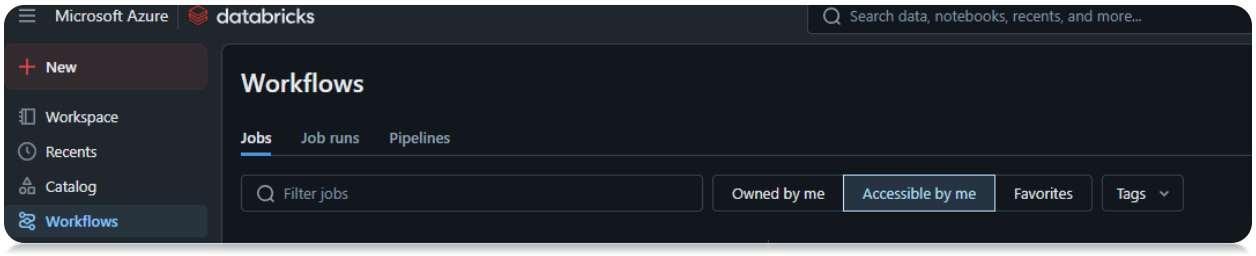
```
17
df_gold_transactions = spark.read.format("delta").option("header", True).load(f"{gold_base_path}transactions")
print("-------------------------------transactions.csv file Data---------------------------------")
display(df_gold_transactions)
```

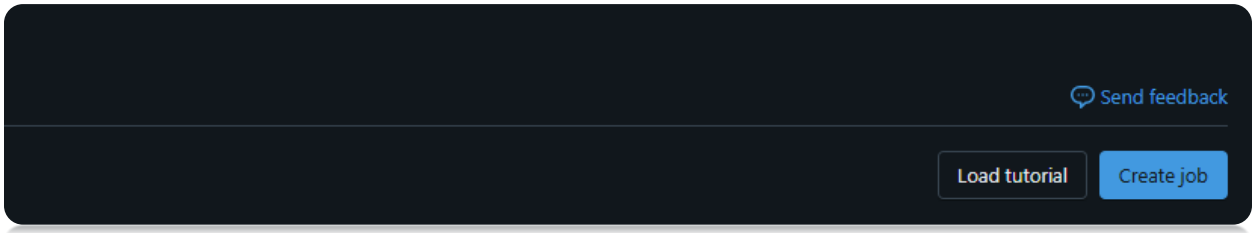
Gold Folder in ADLS Gen 2:



Step 6: Create a scheduled job for daily run

Go to the Workflow tab -> Jobs -> Create Job



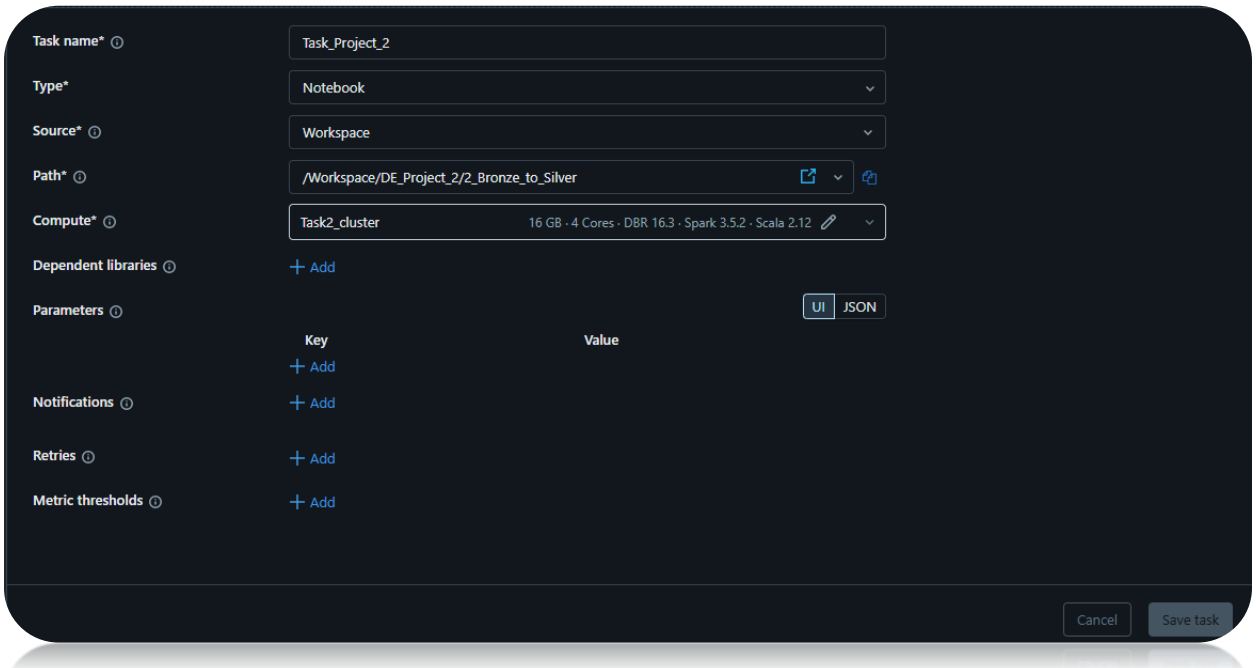


Create a new task:

Mention this information

- Task Name: Name of this new task
- Type: Notebook
- Source Workspace (Select the main notebook which we want to run on a schedule trigger)
- Path: Path of that notebook
- Computer: Select a job compute cluster/Create a new one based on the requirement

Click on **Save Task**



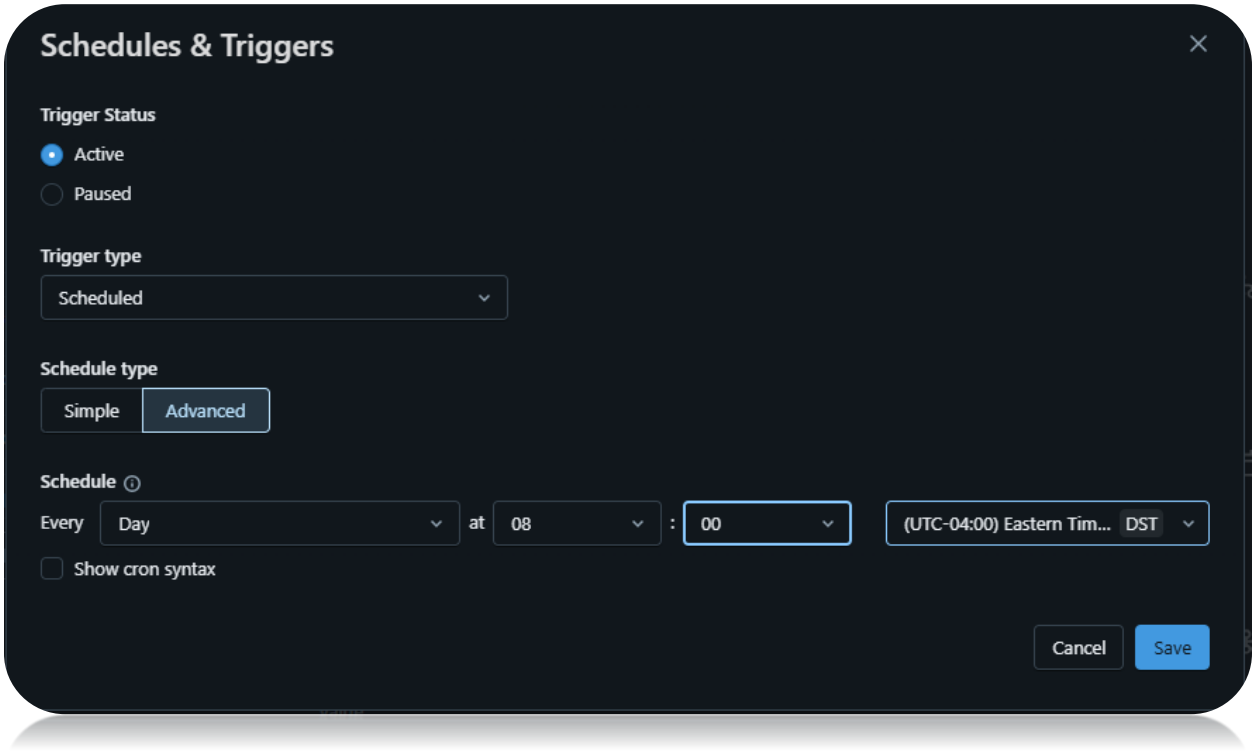
On the right-side panel, click on add schedule trigger and go to advanced options

Here we have selected

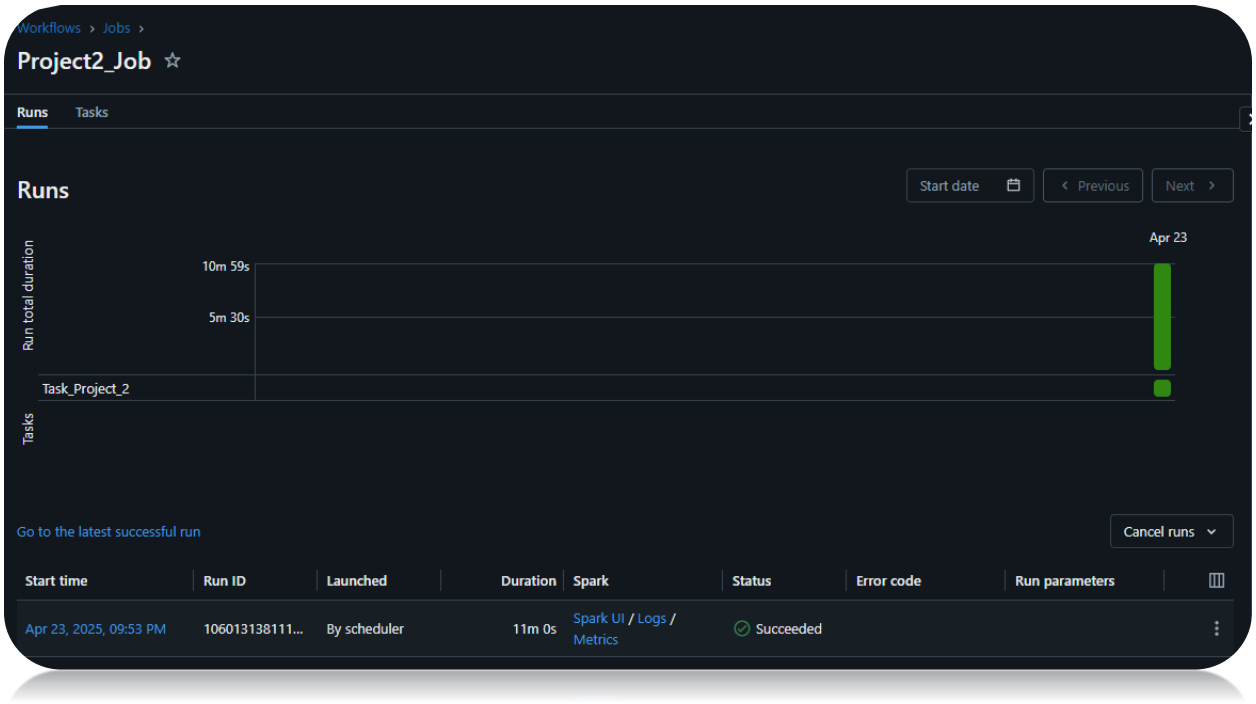
Trigger type: Schedule

Schedule type: Advanced

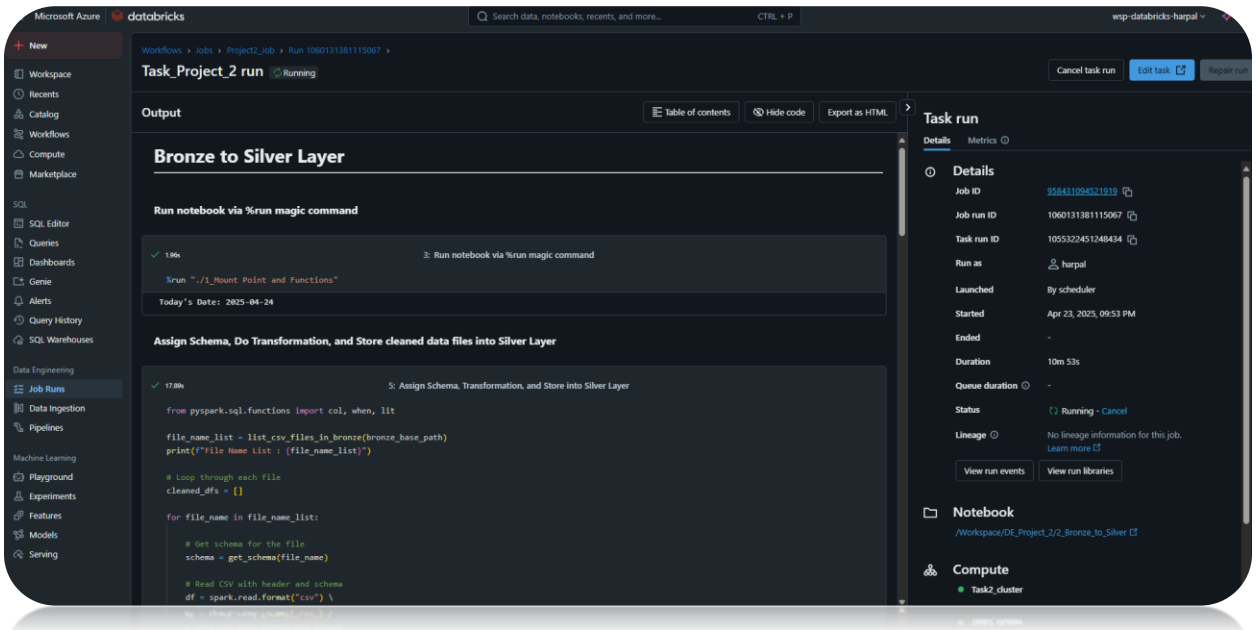
Schedule: Every day 8 AM (UTC -4:00 Time zone) Eastern Time (US and Canada)

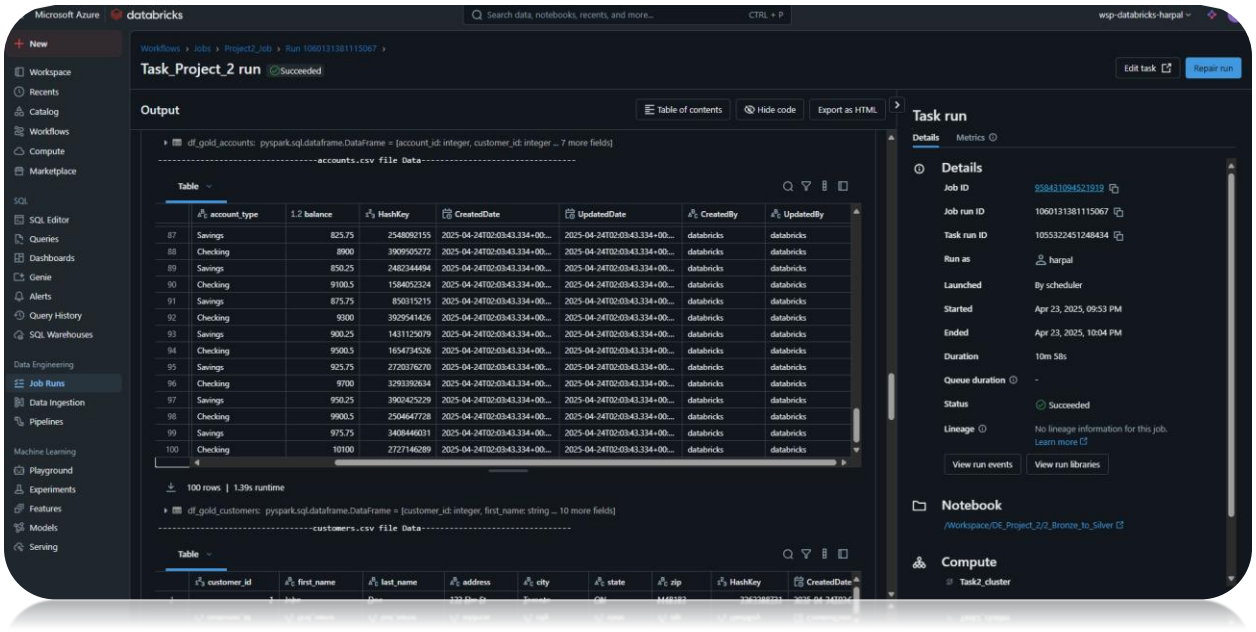


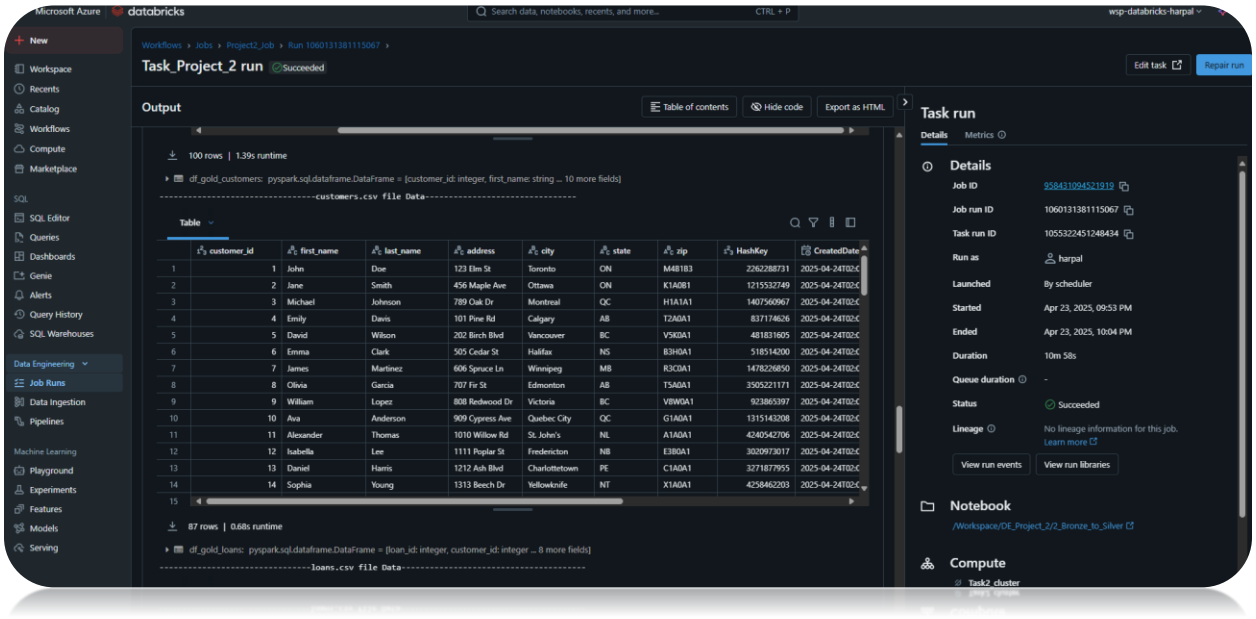
Once the job runs by the scheduler, it will be shown in the Runs tab in the workflow section:



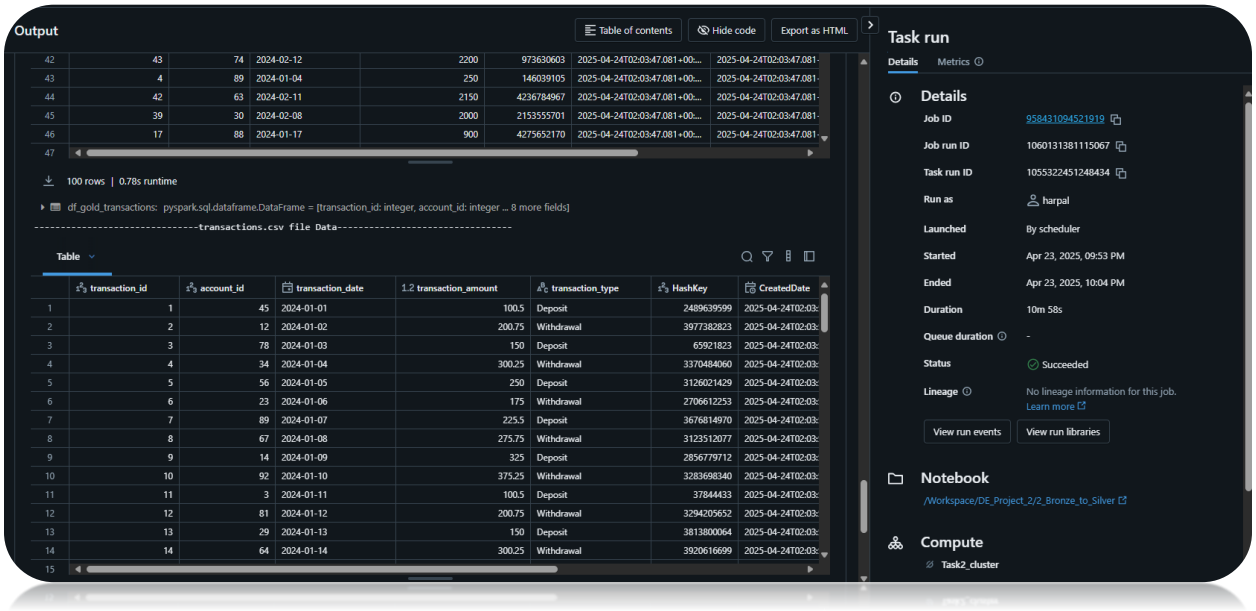
When the job runs on schedule time we can check the process side by side, by clicking on the start time in the Job runs tab:




Microsoft Azure
databricks
Search data, notebooks, recents, and more... CTRL + P
wsp-databricks-harpal
Workflows › Jobs › Project2_Job › Run 1060131381115067 ›
Task_Project_2 run
Succeeded
Edit task
Repair run
Output
Table of contents
Hide code
Export as HTML
df_gold_accounts: pyspark.sql.dataframe.DataFrame = [account_id: integer, customer_id: integer ... 7 more fields]
accounts.csv file Data
Table
account_type | 1.2 balance | HashKey | CreatedDate | UpdatedDate | CreatedBy | UpdatedBy
87 Savings 825.75 2548092155 2025-04-24T02:03:43.334+00... 2025-04-24T02:03:43.334+00... databricks databricks
88 Checking 8900 3909505272 2025-04-24T02:03:43.334+00... 2025-04-24T02:03:43.334+00... databricks databricks
89 Savings 850.25 2482344494 2025-04-24T02:03:43.334+00... 2025-04-24T02:03:43.334+00... databricks databricks
90 Checking 9100.5 1584052324 2025-04-24T02:03:43.334+00... 2025-04-24T02:03:43.334+00... databricks databricks
91 Savings 875.75 850315215 2025-04-24T02:03:43.334+00... 2025-04-24T02:03:43.334+00... databricks databricks
92 Checking 9300 3929541426 2025-04-24T02:03:43.334+00... 2025-04-24T02:03:43.334+00... databricks databricks
93 Savings 900.25 1431125079 2025-04-24T02:03:43.334+00... 2025-04-24T02:03:43.334+00... databricks databricks
94 Checking 9500.5 1654734526 2025-04-24T02:03:43.334+00... 2025-04-24T02:03:43.334+00... databricks databricks
95 Savings 925.75 2720376270 2025-04-24T02:03:43.334+00... 2025-04-24T02:03:43.334+00... databricks databricks
96 Checking 9700 3293392634 2025-04-24T02:03:43.334+00... 2025-04-24T02:03:43.334+00... databricks databricks
97 Savings 950.25 3902425229 2025-04-24T02:03:43.334+00... 2025-04-24T02:03:43.334+00... databricks databricks
98 Checking 9900.5 2504647728 2025-04-24T02:03:43.334+00... 2025-04-24T02:03:43.334+00... databricks databricks
99 Savings 975.75 3408446031 2025-04-24T02:03:43.334+00... 2025-04-24T02:03:43.334+00... databricks databricks
100 Checking 10100 2727146289 2025-04-24T02:03:43.334+00... 2025-04-24T02:03:43.334+00... databricks databricks
100 rows | 1.39s runtime
df_gold_customers: pyspark.sql.dataframe.DataFrame = [customer_id: integer, first_name: string ... 10 more fields]
customers.csv file Data
Table
customer_id | first_name | last_name | address | city | state | zip | HashKey | CreatedDate
Task run
Details
Metrics
Details
Job ID 958431094521919
Job run ID 1060131381115067
Task run ID 1055322451248434
Run as harpal
Launched By scheduler
Started Apr 23, 2025, 09:53 PM
Ended Apr 23, 2025, 10:04 PM
Duration 10m 58s
Queue duration -
Status Succeeded
Lineage No lineage information for this job. Learn more
View run events
View run libraries
Notebook
/Workspace/DE_Project_2/2_Bronze_to_Silver
Compute
Task2_cluster

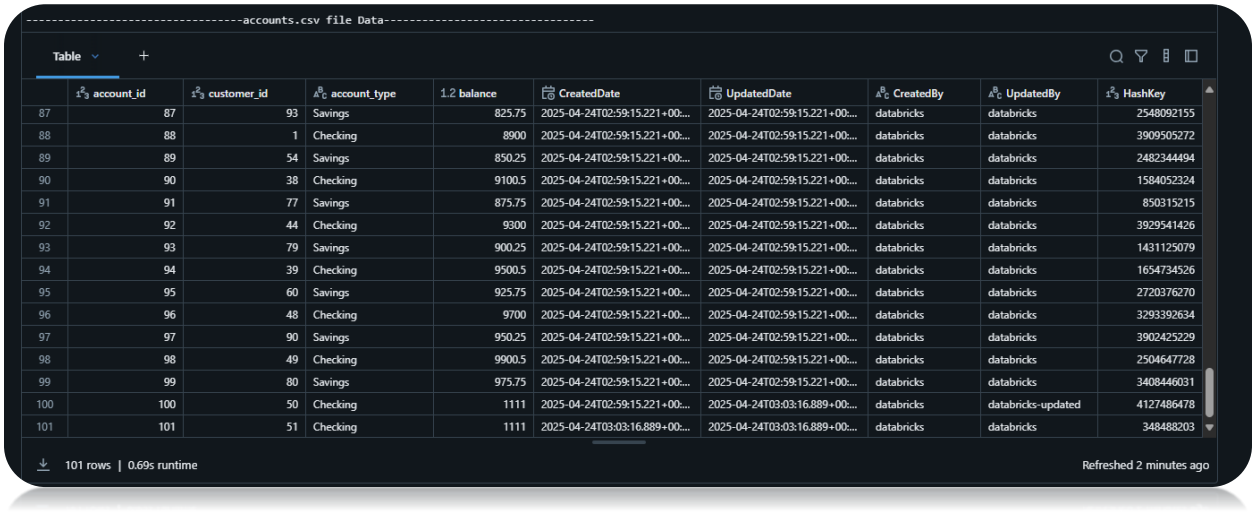

Microsoft Azure
databricks
Search data, notebooks, recents, and more... CTRL + P
wsp-databricks-harpal
Workflows › Jobs › Project2_Job › Run 1060131381115067 ›
Task_Project_2 run
Succeeded
Edit task
Repair run
Output
Table of contents
Hide code
Export as HTML
100 rows | 1.39s runtime
df_gold_customers: pyspark.sql.dataframe.DataFrame = [customer_id: integer, first_name: string ... 10 more fields]
customers.csv file Data
Table
customer_id | first_name | last_name | address | city | state | zip | HashKey | CreatedDate
1 1 John Doe 123 Elm St Toronto ON M4B1B3 2262288731 2025-04-24T02:0
2 2 Jane Smith 456 Maple Ave Ottawa ON K1A0B1 1215532749 2025-04-24T02:0
3 3 Michael Johnson 789 Oak Dr Montreal QC H1A1A1 1407560967 2025-04-24T02:0
4 4 Emily Davis 101 Pine Rd Calgary AB T2A0A1 837174626 2025-04-24T02:0
5 5 David Wilson 202 Birch Blvd Vancouver BC V5K0A1 481831605 2025-04-24T02:0
6 6 Emma Clark 505 Cedar St Halifax NS B3H0A1 518514200 2025-04-24T02:0
7 7 James Martinez 606 Spruce Ln Winnipeg MB R3C0A1 1478226850 2025-04-24T02:0
8 8 Olivia Garcia 707 Fir St Edmonton AB T5A0A1 3505221171 2025-04-24T02:0
9 9 William Lopez 808 Redwood Dr Victoria BC V8W0A1 923865397 2025-04-24T02:0
10 10 Ava Anderson 909 Cypress Ave Quebec City QC G1A0A1 1315143208 2025-04-24T02:0
11 11 Alexander Thomas 1010 Willow Rd St. John's NL A1A0A1 4240542706 2025-04-24T02:0
12 12 Isabella Lee 1111 Poplar St Fredericton NB E3B0A1 3020973017 2025-04-24T02:0
13 13 Daniel Harris 1212 Ash Blvd Charlottetown PE C1A0A1 3271877955 2025-04-24T02:0
14 14 Sophia Young 1313 Beech Dr Yellowknife NT X1A0A1 4258462203 2025-04-24T02:0
87 rows | 0.68s runtime
df_gold_loans: pyspark.sql.dataframe.DataFrame = [loan_id: integer, customer_id: integer ... 8 more fields]
loans.csv file Data
Task run
Details
Metrics
Details
Job ID 958431094521919
Job run ID 1060131381115067
Task run ID 1055322451248434
Run as harpal
Launched By scheduler
Started Apr 23, 2025, 09:53 PM
Ended Apr 23, 2025, 10:04 PM
Duration 10m 58s
Queue duration -
Status Succeeded
Lineage No lineage information for this job. Learn more
View run events
View run libraries
Notebook
/Workspace/DE_Project_2/2_Bronze_to_Silver
Compute
Task2_cluster

At the end Job cluster will be terminated by itself.

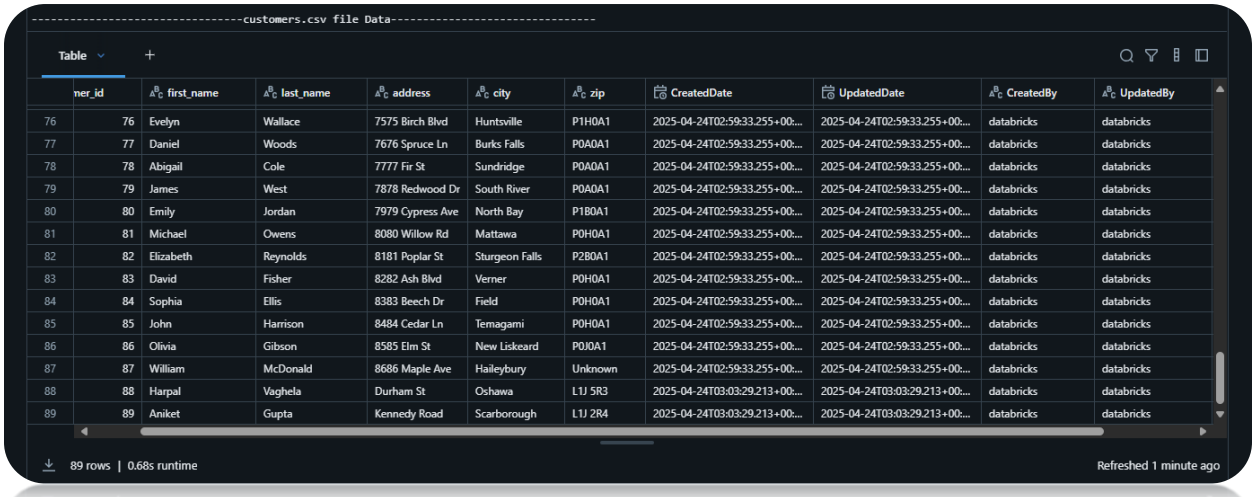


Day 2 Schedule Job run output:

Accounts:

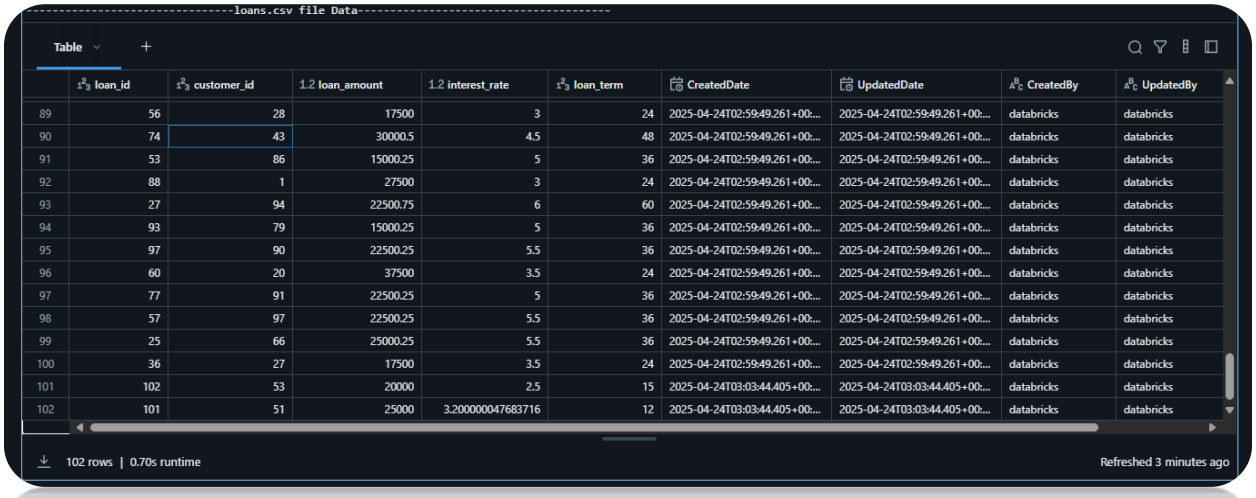


Customers:



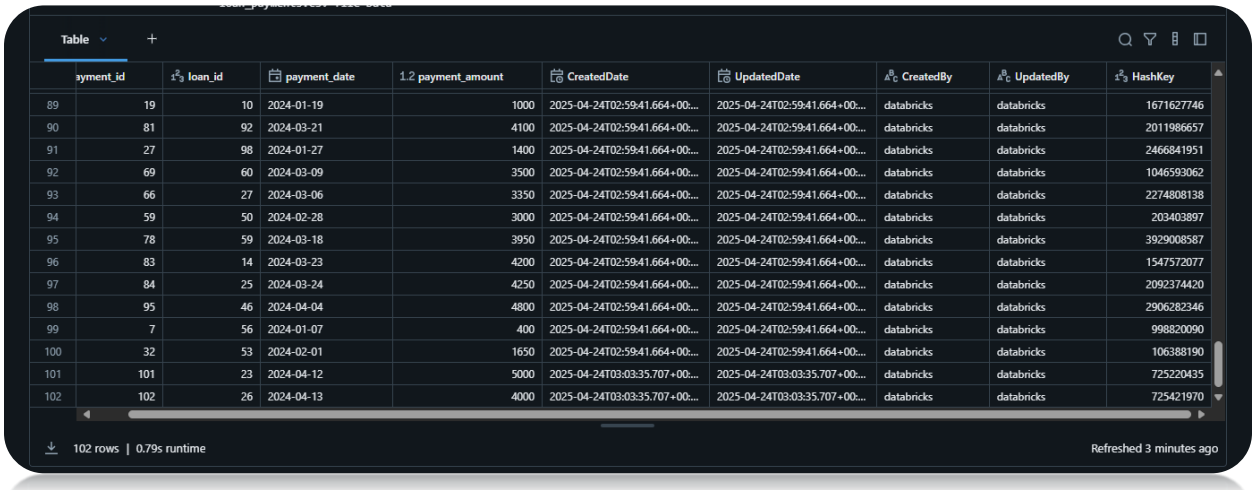
	ner_id	first_name	last_name	address	city	zip	CreatedDate	UpdatedDate	CreatedBy	UpdatedBy
76	76	Evelyn	Wallace	7575 Birch Blvd	Huntsville	P1H0A1	2025-04-24T02:59:33.255+00:...	2025-04-24T02:59:33.255+00:...	databricks	databricks
77	77	Daniel	Woods	7676 Spruce Ln	Burks Falls	P0A0A1	2025-04-24T02:59:33.255+00:...	2025-04-24T02:59:33.255+00:...	databricks	databricks
78	78	Abigail	Cole	7777 Fir St	Sundridge	P0A0A1	2025-04-24T02:59:33.255+00:...	2025-04-24T02:59:33.255+00:...	databricks	databricks
79	79	James	West	7878 Redwood Dr	South River	P0A0A1	2025-04-24T02:59:33.255+00:...	2025-04-24T02:59:33.255+00:...	databricks	databricks
80	80	Emily	Jordan	7979 Cypress Ave	North Bay	P1B0A1	2025-04-24T02:59:33.255+00:...	2025-04-24T02:59:33.255+00:...	databricks	databricks
81	81	Michael	Owens	8080 Willow Rd	Mattawa	P0H0A1	2025-04-24T02:59:33.255+00:...	2025-04-24T02:59:33.255+00:...	databricks	databricks
82	82	Elizabeth	Reynolds	8181 Poplar St	Sturgeon Falls	P2B0A1	2025-04-24T02:59:33.255+00:...	2025-04-24T02:59:33.255+00:...	databricks	databricks
83	83	David	Fisher	8282 Ash Blvd	Verner	P0H0A1	2025-04-24T02:59:33.255+00:...	2025-04-24T02:59:33.255+00:...	databricks	databricks
84	84	Sophia	Ellis	8383 Beech Dr	Field	P0H0A1	2025-04-24T02:59:33.255+00:...	2025-04-24T02:59:33.255+00:...	databricks	databricks
85	85	John	Harrison	8484 Cedar Ln	Temagami	P0H0A1	2025-04-24T02:59:33.255+00:...	2025-04-24T02:59:33.255+00:...	databricks	databricks
86	86	Olivia	Gibson	8585 Elm St	New Liskeard	P0J0A1	2025-04-24T02:59:33.255+00:...	2025-04-24T02:59:33.255+00:...	databricks	databricks
87	87	William	McDonald	8686 Maple Ave	Haileybury	Unknown	2025-04-24T02:59:33.255+00:...	2025-04-24T02:59:33.255+00:...	databricks	databricks
88	88	Harpal	Vaghela	Durham St	Oshawa	L1J 5R3	2025-04-24T03:03:29.213+00:...	2025-04-24T03:03:29.213+00:...	databricks	databricks
89	89	Aniket	Gupta	Kennedy Road	Scarborough	L1J 2R4	2025-04-24T03:03:29.213+00:...	2025-04-24T03:03:29.213+00:...	databricks	databricks

Loans:



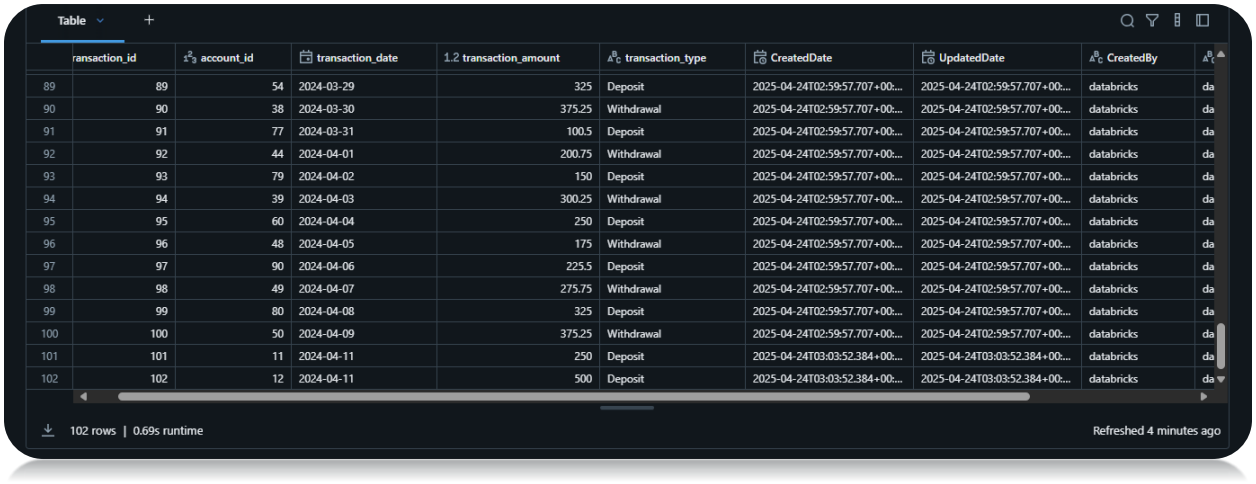
	loan_id	customer_id	loan_amount	interest_rate	loan_term	CreatedDate	UpdatedDate	CreatedBy	UpdatedBy
89	56	28	17500	3	24	2025-04-24T02:59:49.261+00:...	2025-04-24T02:59:49.261+00:...	databricks	databricks
90	74	43	30000.5	4.5	48	2025-04-24T02:59:49.261+00:...	2025-04-24T02:59:49.261+00:...	databricks	databricks
91	53	86	15000.25	5	36	2025-04-24T02:59:49.261+00:...	2025-04-24T02:59:49.261+00:...	databricks	databricks
92	88	1	27500	3	24	2025-04-24T02:59:49.261+00:...	2025-04-24T02:59:49.261+00:...	databricks	databricks
93	27	94	22500.75	6	60	2025-04-24T02:59:49.261+00:...	2025-04-24T02:59:49.261+00:...	databricks	databricks
94	93	79	15000.25	5	36	2025-04-24T02:59:49.261+00:...	2025-04-24T02:59:49.261+00:...	databricks	databricks
95	97	90	22500.25	5.5	36	2025-04-24T02:59:49.261+00:...	2025-04-24T02:59:49.261+00:...	databricks	databricks
96	60	20	37500	3.5	24	2025-04-24T02:59:49.261+00:...	2025-04-24T02:59:49.261+00:...	databricks	databricks
97	77	91	22500.25	5	36	2025-04-24T02:59:49.261+00:...	2025-04-24T02:59:49.261+00:...	databricks	databricks
98	57	97	22500.25	5.5	36	2025-04-24T02:59:49.261+00:...	2025-04-24T02:59:49.261+00:...	databricks	databricks
99	25	66	25000.25	5.5	36	2025-04-24T02:59:49.261+00:...	2025-04-24T02:59:49.261+00:...	databricks	databricks
100	36	27	17500	3.5	24	2025-04-24T02:59:49.261+00:...	2025-04-24T02:59:49.261+00:...	databricks	databricks
101	102	53	20000	2.5	15	2025-04-24T03:03:44.405+00:...	2025-04-24T03:03:44.405+00:...	databricks	databricks
102	101	51	25000	3.200000047683716	12	2025-04-24T03:03:44.405+00:...	2025-04-24T03:03:44.405+00:...	databricks	databricks

Loan Payments:



	ayment_id	loan_id	payment_date	payment_amount	CreatedDate	UpdatedDate	CreatedBy	UpdatedBy	HashKey
89	19	10	2024-01-19	1000	2025-04-24T02:59:41.664+00:...	2025-04-24T02:59:41.664+00:...	databricks	databricks	1671627746
90	81	92	2024-03-21	4100	2025-04-24T02:59:41.664+00:...	2025-04-24T02:59:41.664+00:...	databricks	databricks	2011986657
91	27	98	2024-01-27	1400	2025-04-24T02:59:41.664+00:...	2025-04-24T02:59:41.664+00:...	databricks	databricks	2466841951
92	69	60	2024-03-09	3500	2025-04-24T02:59:41.664+00:...	2025-04-24T02:59:41.664+00:...	databricks	databricks	1046593062
93	66	27	2024-03-06	3350	2025-04-24T02:59:41.664+00:...	2025-04-24T02:59:41.664+00:...	databricks	databricks	2274808138
94	59	50	2024-02-28	3000	2025-04-24T02:59:41.664+00:...	2025-04-24T02:59:41.664+00:...	databricks	databricks	203403897
95	78	59	2024-03-18	3950	2025-04-24T02:59:41.664+00:...	2025-04-24T02:59:41.664+00:...	databricks	databricks	3929008587
96	83	14	2024-03-23	4200	2025-04-24T02:59:41.664+00:...	2025-04-24T02:59:41.664+00:...	databricks	databricks	1547572077
97	84	25	2024-03-24	4250	2025-04-24T02:59:41.664+00:...	2025-04-24T02:59:41.664+00:...	databricks	databricks	2092374420
98	95	46	2024-04-04	4800	2025-04-24T02:59:41.664+00:...	2025-04-24T02:59:41.664+00:...	databricks	databricks	2906282346
99	7	56	2024-01-07	400	2025-04-24T02:59:41.664+00:...	2025-04-24T02:59:41.664+00:...	databricks	databricks	998820090
100	32	53	2024-02-01	1650	2025-04-24T02:59:41.664+00:...	2025-04-24T02:59:41.664+00:...	databricks	databricks	106388190
101	101	23	2024-04-12	5000	2025-04-24T03:03:35.707+00:...	2025-04-24T03:03:35.707+00:...	databricks	databricks	725220435
102	102	26	2024-04-13	4000	2025-04-24T03:03:35.707+00:...	2025-04-24T03:03:35.707+00:...	databricks	databricks	725421970

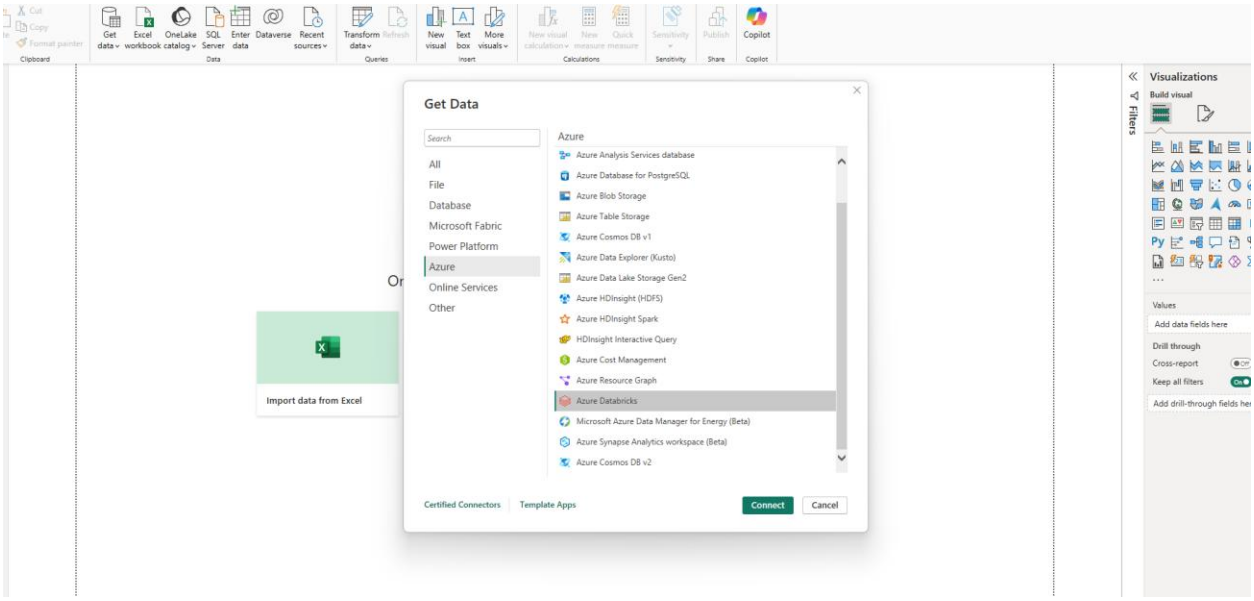
Transactions:



	ransaction_id	account_id	transaction_date	1.2 transaction_amount	transaction_type	CreatedDate	UpdatedDate	CreatedBy	
89	89	54	2024-03-29	325	Deposit	2025-04-24T02:59:57.707+00:...	2025-04-24T02:59:57.707+00:...	databricks	da
90	90	38	2024-03-30	375.25	Withdrawal	2025-04-24T02:59:57.707+00:...	2025-04-24T02:59:57.707+00:...	databricks	da
91	91	77	2024-03-31	100.5	Deposit	2025-04-24T02:59:57.707+00:...	2025-04-24T02:59:57.707+00:...	databricks	da
92	92	44	2024-04-01	200.75	Withdrawal	2025-04-24T02:59:57.707+00:...	2025-04-24T02:59:57.707+00:...	databricks	da
93	93	79	2024-04-02	150	Deposit	2025-04-24T02:59:57.707+00:...	2025-04-24T02:59:57.707+00:...	databricks	da
94	94	39	2024-04-03	300.25	Withdrawal	2025-04-24T02:59:57.707+00:...	2025-04-24T02:59:57.707+00:...	databricks	da
95	95	60	2024-04-04	250	Deposit	2025-04-24T02:59:57.707+00:...	2025-04-24T02:59:57.707+00:...	databricks	da
96	96	48	2024-04-05	175	Withdrawal	2025-04-24T02:59:57.707+00:...	2025-04-24T02:59:57.707+00:...	databricks	da
97	97	90	2024-04-06	225.5	Deposit	2025-04-24T02:59:57.707+00:...	2025-04-24T02:59:57.707+00:...	databricks	da
98	98	49	2024-04-07	275.75	Withdrawal	2025-04-24T02:59:57.707+00:...	2025-04-24T02:59:57.707+00:...	databricks	da
99	99	80	2024-04-08	325	Deposit	2025-04-24T02:59:57.707+00:...	2025-04-24T02:59:57.707+00:...	databricks	da
100	100	50	2024-04-09	375.25	Withdrawal	2025-04-24T02:59:57.707+00:...	2025-04-24T02:59:57.707+00:...	databricks	da
101	101	11	2024-04-11	250	Deposit	2025-04-24T03:03:52.384+00:...	2025-04-24T03:03:52.384+00:...	databricks	da
102	102	12	2024-04-11	500	Deposit	2025-04-24T03:03:52.384+00:...	2025-04-24T03:03:52.384+00:...	databricks	da

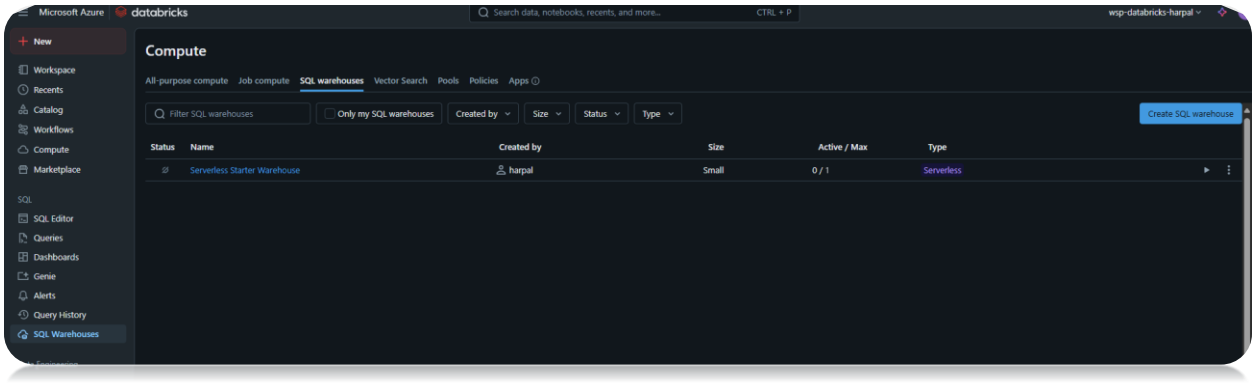
102 rows | 0.69s runtime Refreshed 4 minutes ago

Step 7: Power BI Visualization on DELTA File in Silver Layer

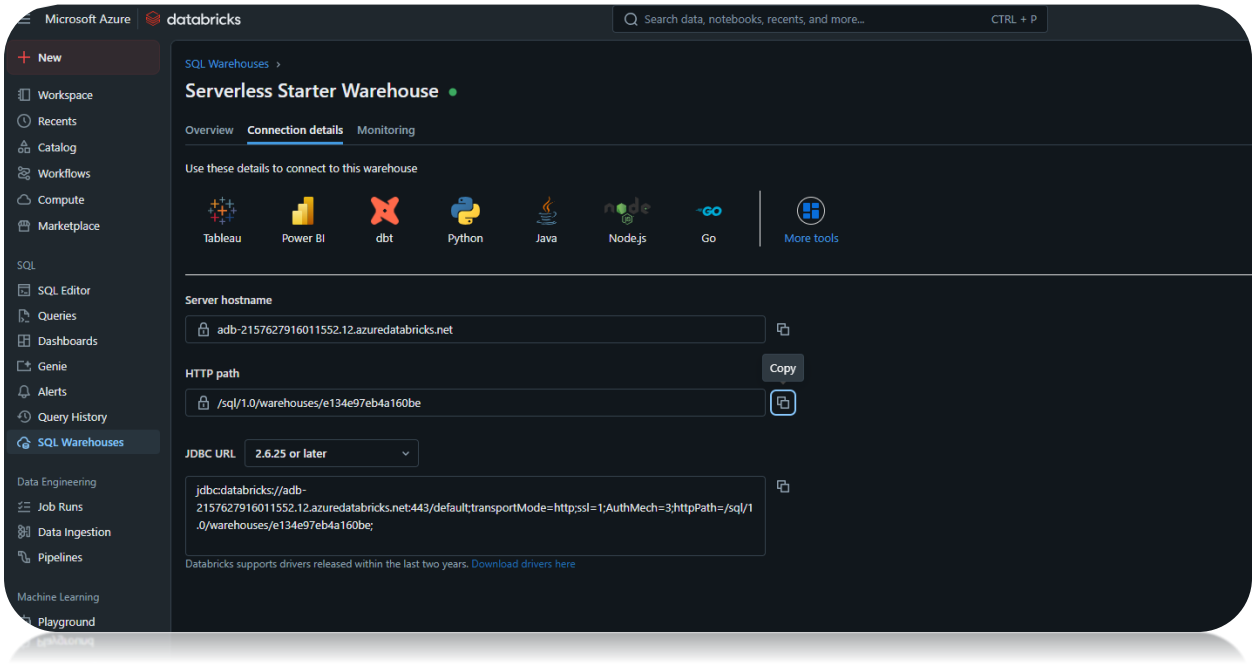


Click on Connect

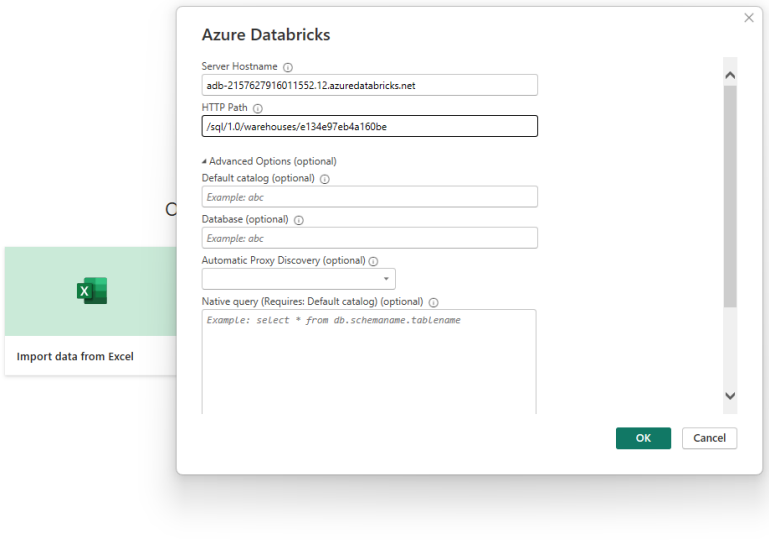
Now, to Databricks SQL Warehouse, start the warehouse



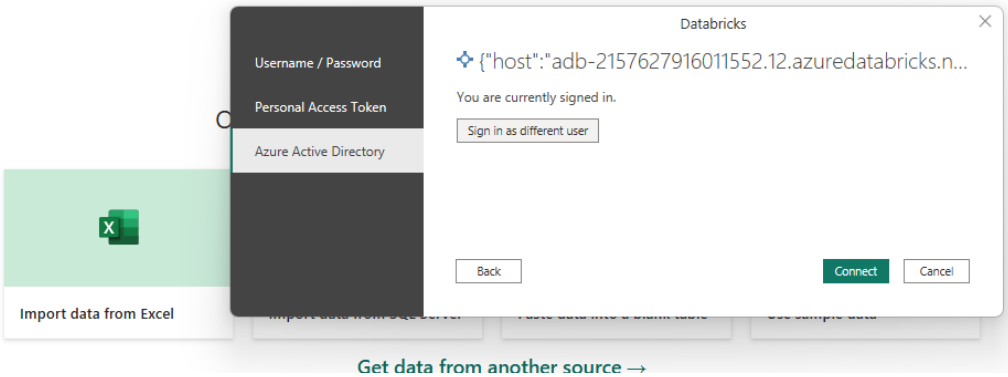
Copy Hostname and HTTP Path



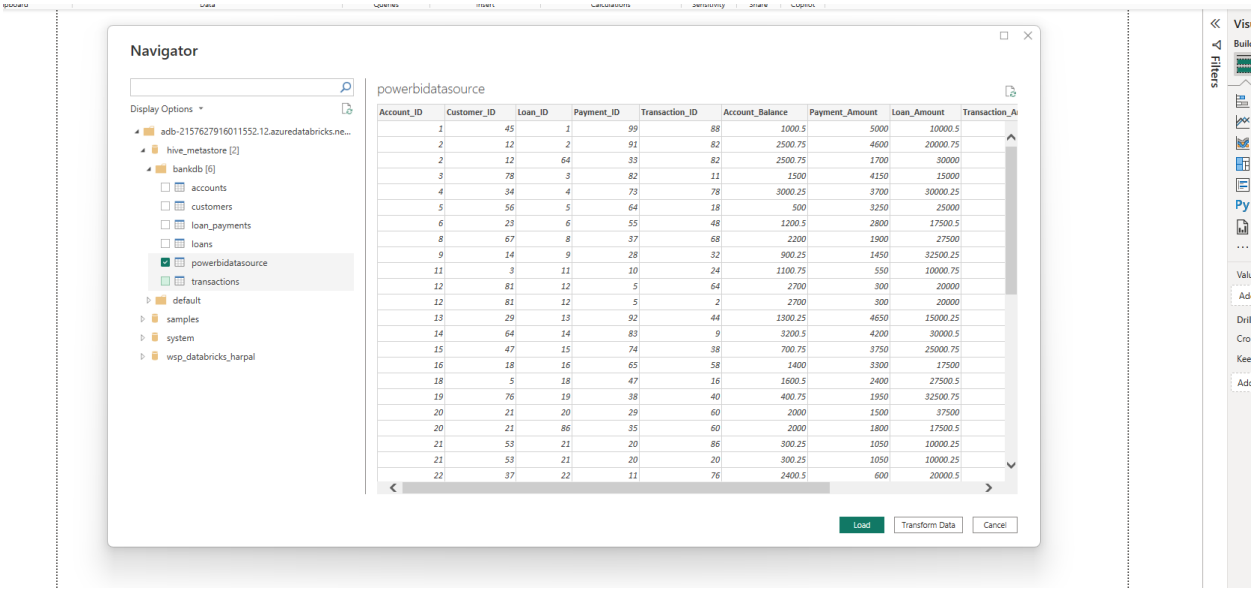
Paste that Hostname and HTTP Path in Power BI window as shown below:



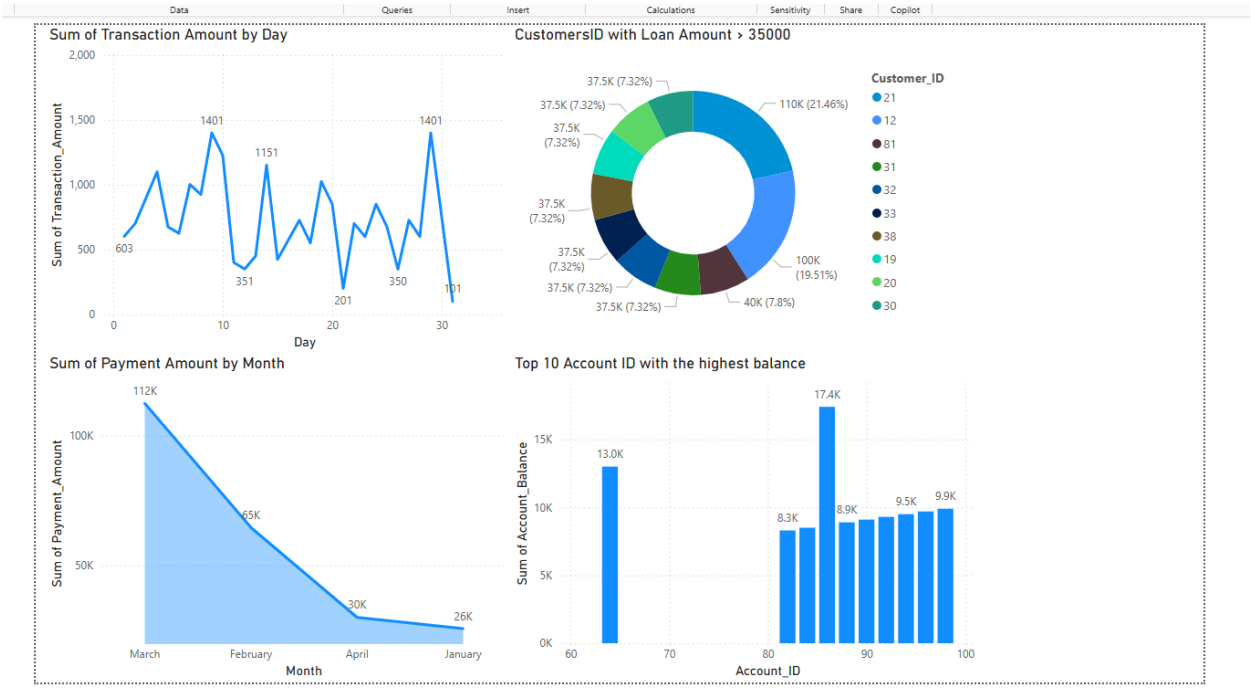
Click on Connect



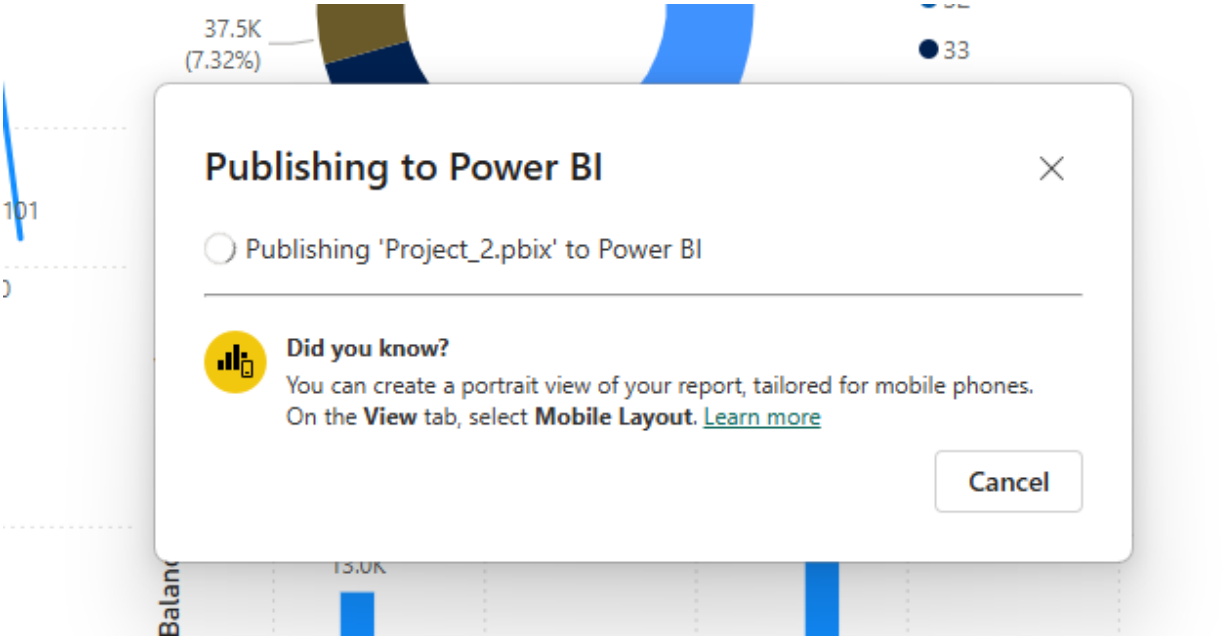
Select the table



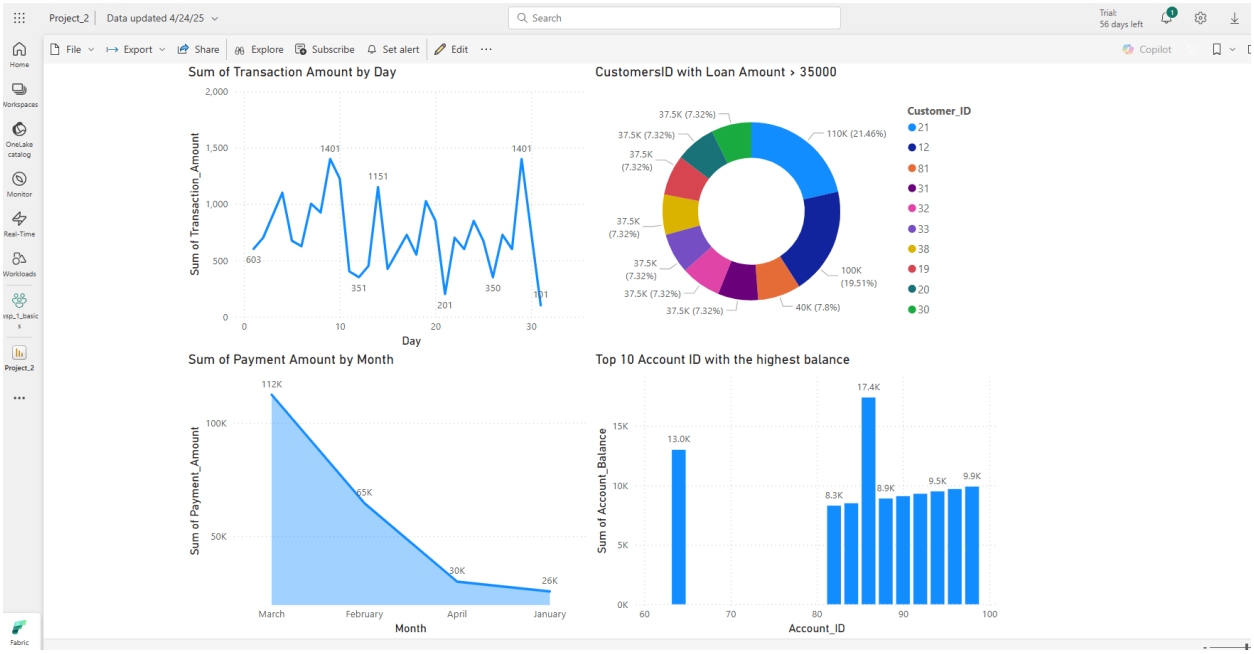
Create few visuals like this



Step 8: Publish it to the Fabric Workspace



Open the Fabric Workspace and check the visuals there:



Conclusion:

This project successfully designed and implemented a robust data pipeline in Databricks to process customer account, loan, transaction, and payment data. Using ADLS Gen2, Databricks notebooks, and Delta/Parquet formats, we established a layered architecture (Bronze → Silver → Gold) for scalable transformation, SCD handling, and analytics. Final outputs were integrated with Power BI, enabling powerful visual reporting and business insights.

Points to remember:

- Parquet files can be overwritten using .mode("overwrite"), but they don't support merge or versioning like Delta.
- Use Delta format when you need SCD Type 1, incremental processing, or Power BI connectivity, it supports ACID transactions and time travel.
- Use .saveAsTable("db.table") for **managed Delta tables**, but never provide a file path with it.
- When overwriting **CSV or Parquet** files, use .coalesce(1) to avoid multiple part-*.csv files if you want a single output.
- To rename output from Spark (e.g., to accounts.csv), **write to a temp folder, then rename part-*.csv**, and remove the temp folder.
- Implementing **SCD Type 1** logic requires MERGE INTO with Delta format, Parquet files can't handle this.
- Use **functions** like transform_dataframe() and load_to_silver() to modularize and reuse your transformation logic across different files.
- Always create **date-based backup folders** like /client_backup_files/YYYY/MM/DD/ to keep historical snapshots organized.
- In Power BI, we can **connect to Delta tables via Databricks SQL Warehouses**, and use **refresh** to update visuals after new data loads.
- When reading .csv files, always use .schema() instead of inferSchema=True to avoid type mismatches and ensure consistent transformations.
- Implement reusable utility functions for: schema loading, SCD Type 1 logic, safe reads, backup routines, and path building. This saves a lot of code duplication.
- For Power BI, use sum, average, and count aggregations in visuals (not raw tables), and apply filters like "Top N" or "greater than" directly in the Filters pane for performance and clarity.