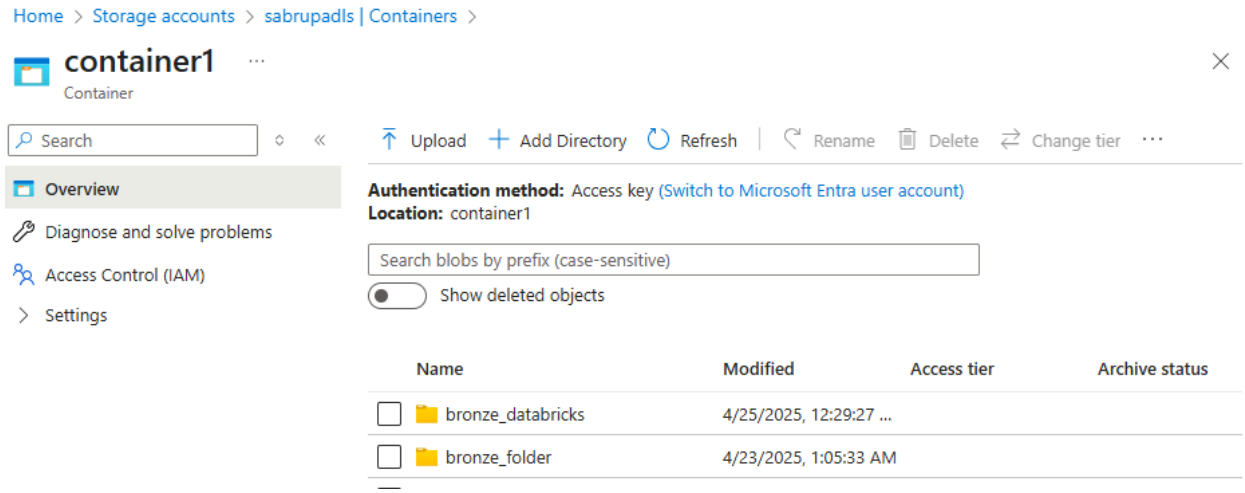
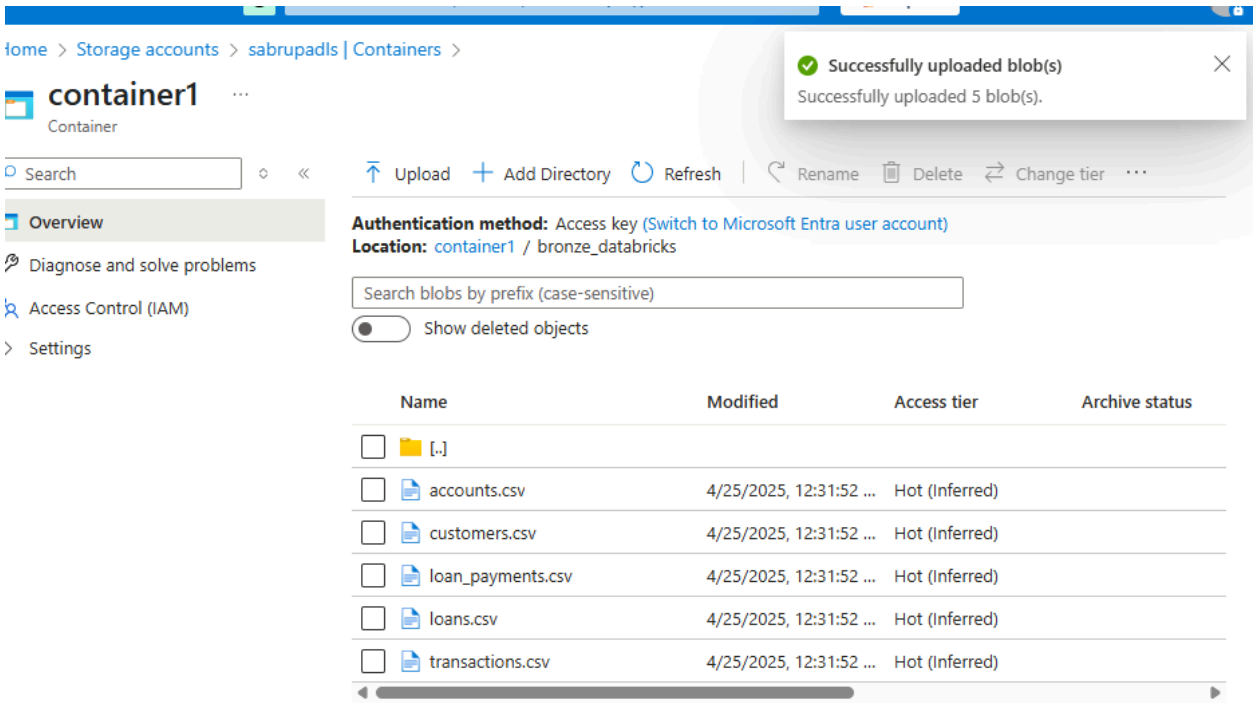
Project 2

1. For the **bronze layer**, we have to store 5 files from the local machine to storage account. For that I have created a new directory named **bronze_databricks.**



And uploaded all the files manually.



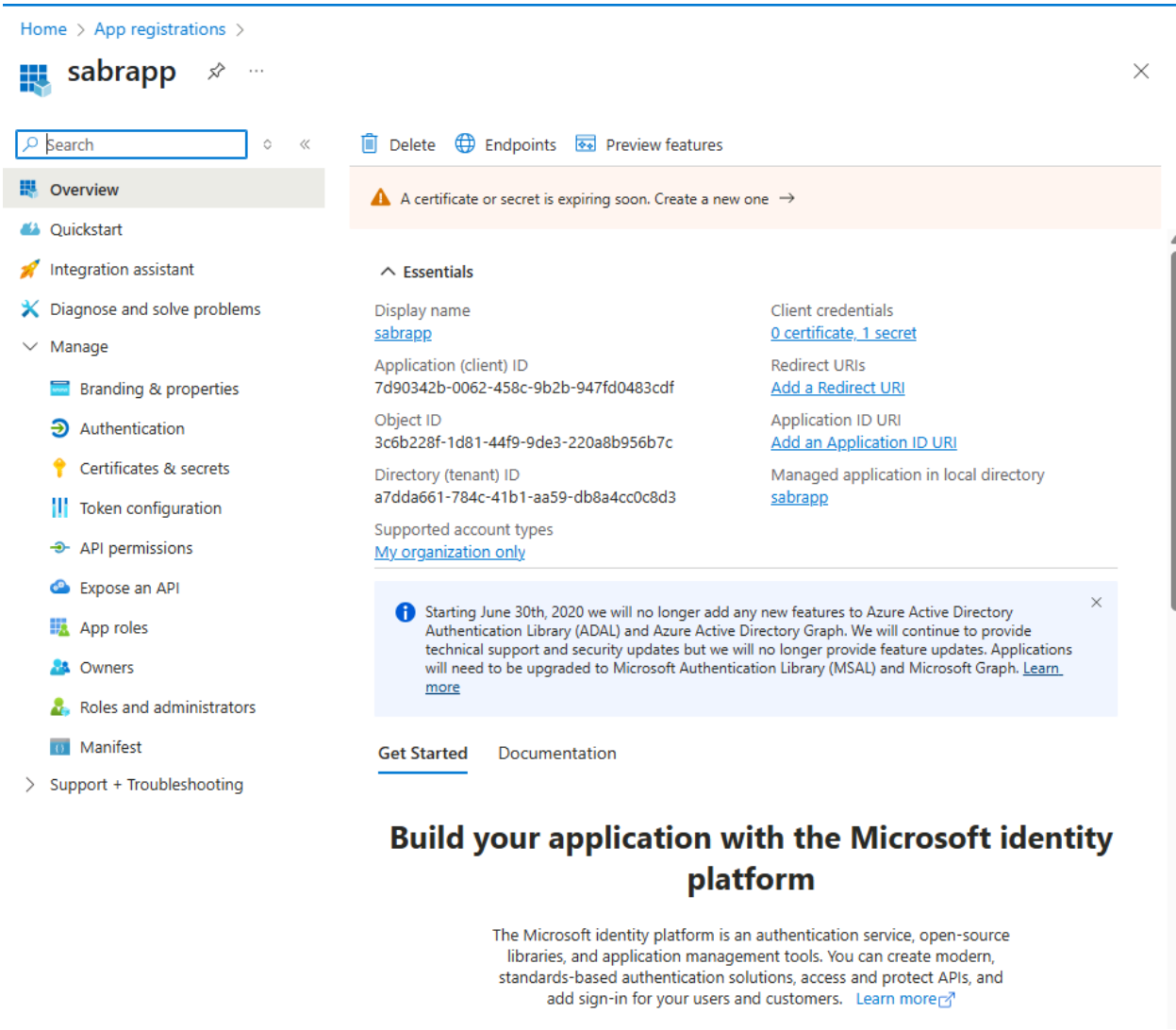
Now all the files are in csv format and in a single folder.

2. **The second step is the silver layer where I performed two tasks.** One is to fetch all each of the file and perform some transformation on them to clean and store in parquet format in a new directory.

Also have to join all the files based on the primary key and **store in delta** format.

To establish the connection between ADLS and the databricks, I used **service principle:**

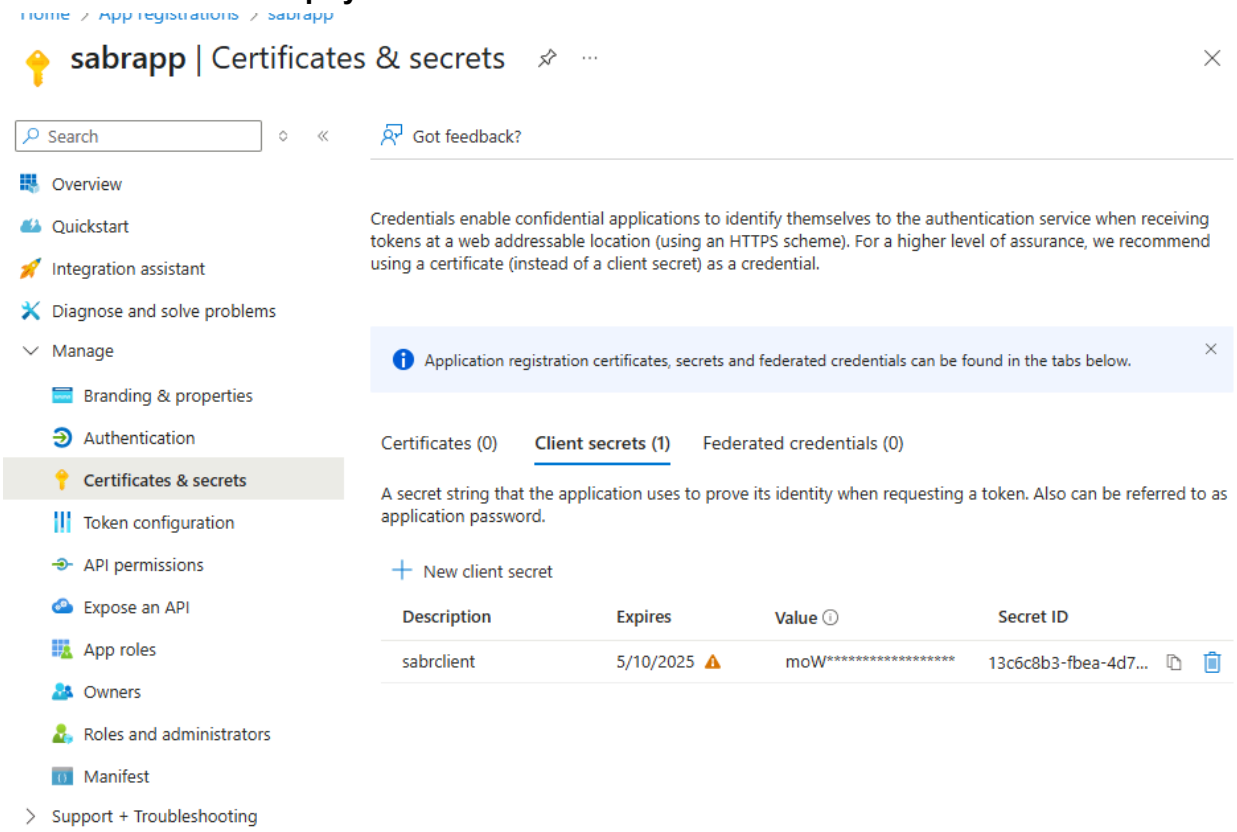
- **First of all, a new registration is created under Microsoft Entra ID-> Manage-> App Registration**



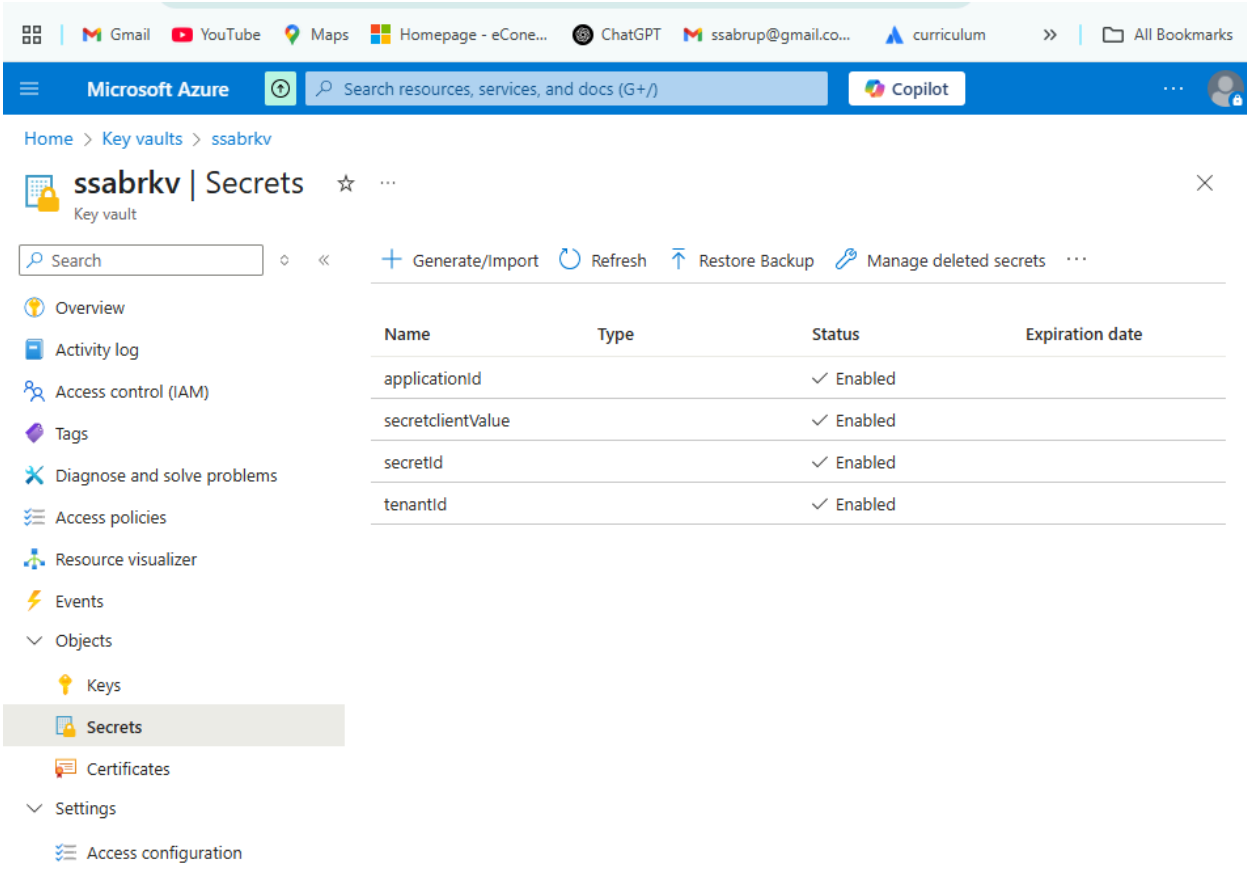
As in the SC new-app I have created.

- **Secondly, we need to copy Application ID.**

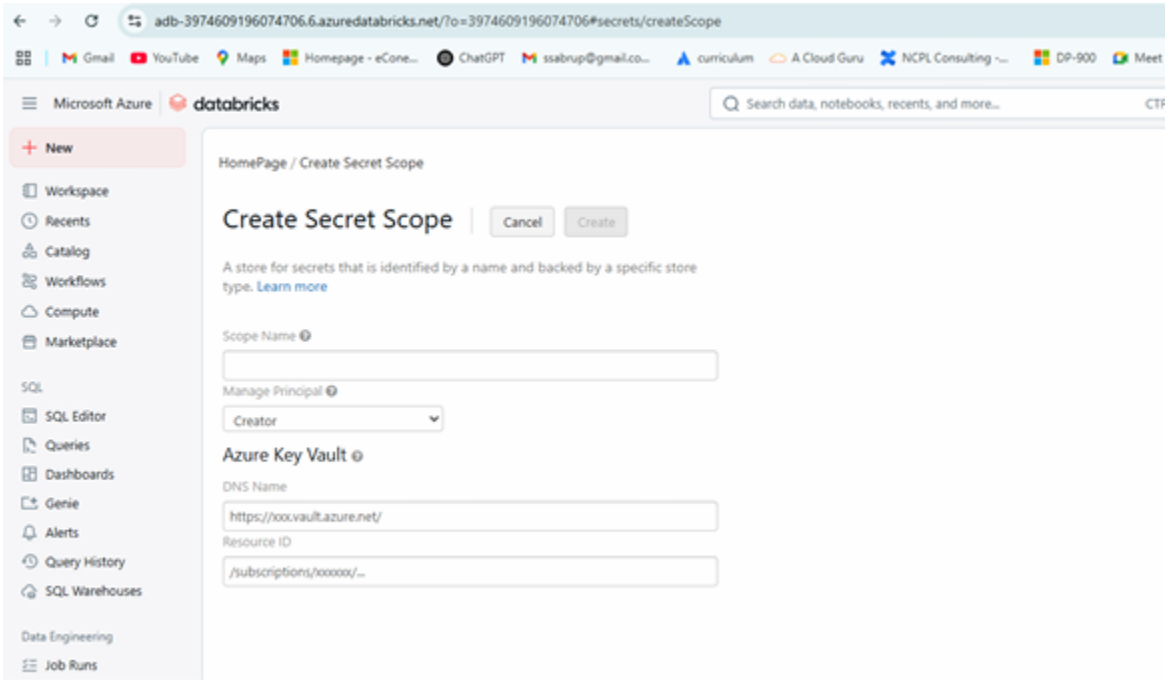
- **After that under Certificates and Secrets, we have to create new client secret with custom expiry date.**



- **As here I have created "sabrclient". And copy the value and the secret ID for further use while connecting.**
- **Further, key Vaults have to be created to hide the IDs**



As I have created "ssabrkv" key Vault with app-id, app-value and secretid.



- **Here DNS Name and the Resource ID we can get from the "Properties" tab**

in Key Vault:

Name	ssabrkv
Sku (Pricing tier)	Standard
Location	canadacentral
Vault URI	https://ssabrkv.vault.azure.net/
Resource ID	/subscriptions/d74c8e4f-2be5-4
Subscription ID	d74c8e4f-2be5-4e1d-9321-ecca
Subscription Name	Azure subscription 1
Directory ID	a7dda661-784c-41b1-aa59-db8
Directory Name	Default Directory
Soft-delete	**Soft delete has been enabled on**
Days to retain deleted vaults	90
Purge protection	(•) Disable purge protection (allo purged during retention peri ( ) Enable purge protection (enf for deleted vaults and vault o

- **Then in databricks run the following command to check the scope:**

```
12:08 AM (<1s)                    1
dbutils.secrets.listScopes()
```

```
[SecretScope(name='sabradlsscope'),
 SecretScope(name='sabrscope'),
 SecretScope(name='sabrscope2')]
```

- **Now, it's time to connect the storage account with the databricks using mounting. For that I have created a mount:**

12:17 AM (14s) 3 Python

```python
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": "7d90342b-0062-458c-9b2b-947fd0483cdf",
  "fs.azure.account.oauth2.client.secret":
  "moW8Q~XNrZC1dGpj1hwOXCL2CuP5dBl02aI9yc9m",
  "fs.azure.account.oauth2.client.endpoint": "https://login.microsoftonline.com/
  a7dda661-784c-41b1-aa59-db8a4cc0c8d3/oauth2/token"
}


dbutils.fs.mount(
  source = "abfss://container1@sabrupadls.dfs.core.windows.net/",
  mount_point = "/mnt/container1",
  extra_configs = configs
)
```

```
True
```

- **The below SC showing all the files in bronze_databricks folder:**

Just now (4s) 4 Python

```python
dbutils.fs.ls("/mnt/container1/bronze_databricks")
```

```
[FileInfo(path='dbfs:/mnt/container1/bronze_databricks/accounts.csv', name='accounts.cs
v', size=2331, modificationTime=1745555512000),
 FileInfo(path='dbfs:/mnt/container1/bronze_databricks/customers.csv', name='customers.c
sv', size=4603, modificationTime=1745555512000),
 FileInfo(path='dbfs:/mnt/container1/bronze_databricks/loan_payments.csv', name='loan_pa
yments.csv', size=2613, modificationTime=1745555512000),
 FileInfo(path='dbfs:/mnt/container1/bronze_databricks/loans.csv', name='loans.csv', siz
e=2340, modificationTime=1745555512000),
 FileInfo(path='dbfs:/mnt/container1/bronze_databricks/transactions.csv', name='transact
ions.csv', size=3513, modificationTime=1745555512000)]
```

- **Here I created 5 dataframes to store the data from the 5 files:**

Just now (3s) 5 Python

```python
df_accounts=spark.read.format("csv").option("header", "true").option
("inferSchema", "true").load("/mnt/container1/bronze_folder/accounts.csv")

df_customers=spark.read.format("csv").option("header", "true").option
("inferSchema", "true").load("/mnt/container1/bronze_folder/customers.csv")

df_loan_payments=spark.read.format("csv").option("header", "true").option
("inferSchema", "true").load("/mnt/container1/bronze_folder/loan_payments.csv")


df_loans=spark.read.format("csv").option("header", "true").option("inferSchema",
"true").load("/mnt/container1/bronze_folder/loans.csv")

df_transactions=spark.read.format("csv").option("header", "true").option
("inferSchema", "true").load("/mnt/container1/bronze_folder/transactions.csv")
```

(10) Spark Jobs

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

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

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

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

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

+ Code + Text

- **If I display customers dataframe, it has null values:**

74		Graham	7373 Oak Dr	Bala	ON	P0C0A1
75		Sullivan	7474 Pine Rd	Bracebridge	ON	P1L0A1
76		Wallace	7575 Birch Blvd	Huntsville	ON	P1H0A1
77		Woods	7676 Spruce Ln	Burks Falls	ON	P0A0A1
78		Cole	7777 Fir St	Sundridge	ON	P0A0A1
79		West	7878 Redwood Dr	South River	ON	P0A0A1
80		Jordan	7979 Cypress Ave	North Bay	ON	P1B0A1
81		Owens	8080 Willow Rd	Mattawa	ON	P0H0A1
82		Reynolds	8181 Poplar St	Sturgeon Falls	ON	P2B0A1
83		Fisher	8282 Ash Blvd	Verner	ON	P0H0A1
84		Ellis	8383 Beech Dr	Field	ON	P0H0A1
85		Harrison	8484 Cedar Ln	Temagami	ON	P0H0A1
86		Gibson	8585 Elm St	New Liskeard	ON	P0J0A1
87		McDonald	8686 Maple Ave	Haileybury	null	null

- **The following command removed the null values. Actually it removes the whole row.**

```
df_customers_clean = df_customers.na.drop()
display(df_customers_clean)
```

01:15 AM (<1s) 7 Python

```
df_customers_clean = df_customers.na.drop()
display(df_customers_clean)
```

(1) Spark Jobs

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

Table

	customer_id	first_name	last_name	address	city
73	73	Andrew	Hamilton	7272 Maple Ave	Gravenhurs
74	74	Harper	Graham	7373 Oak Dr	Bala
75	75	Joshua	Sullivan	7474 Pine Rd	Bracebridge
76	76	Evelyn	Wallace	7575 Birch Blvd	Huntsville
77	77	Daniel	Woods	7676 Spruce Ln	Burks Falls
78	78	Abigail	Cole	7777 Fir St	Sundridge
79	79	James	West	7878 Redwood Dr	South River
80	80	Emily	Jordan	7979 Cypress Ave	North Bay
81	81	Michael	Owens	8080 Willow Rd	Mattawa
82	82	Elizabeth	Reynolds	8181 Poplar St	Sturgeon Fa
83	83	David	Fisher	8282 Ash Blvd	Verner
84	84	Sophia	Ellis	8383 Beech Dr	Field
85	85	John	Harrison	8484 Cedar Ln	Temagami
86	86	Olivia	Gibson	8585 Elm St	New Liskea

The below SC showing the commands to remove all the null values from all the 5 dataframes.

Just now (<1s) 9 Python

```
df_customers_clean = df_customers.na.drop()
df_accounts_clean = df_accounts.na.drop()
df_loan_payments_clean = df_loan_payments.na.drop()
df_loans_clean = df_loans.na.drop()
df_transactions_clean = df_transactions.na.drop()
```

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

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

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

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

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

Also to remove the duplicates:

1 minute ago (<1s) 6 Python

```
df_customers_clean = df_customers.na.drop().dropDuplicates()
df_accounts_clean = df_accounts.na.drop().dropDuplicates()
df_loan_payments_clean = df_loan_payments.na.drop().dropDuplicates()
df_loans_clean = df_loans.na.drop().dropDuplicates()
df_transactions_clean = df_transactions.na.drop().dropDuplicates()
```

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

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

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

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

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

After cleaning the data, we have to write back to storage account but in parquet format.

Just now (6s) 8 Python

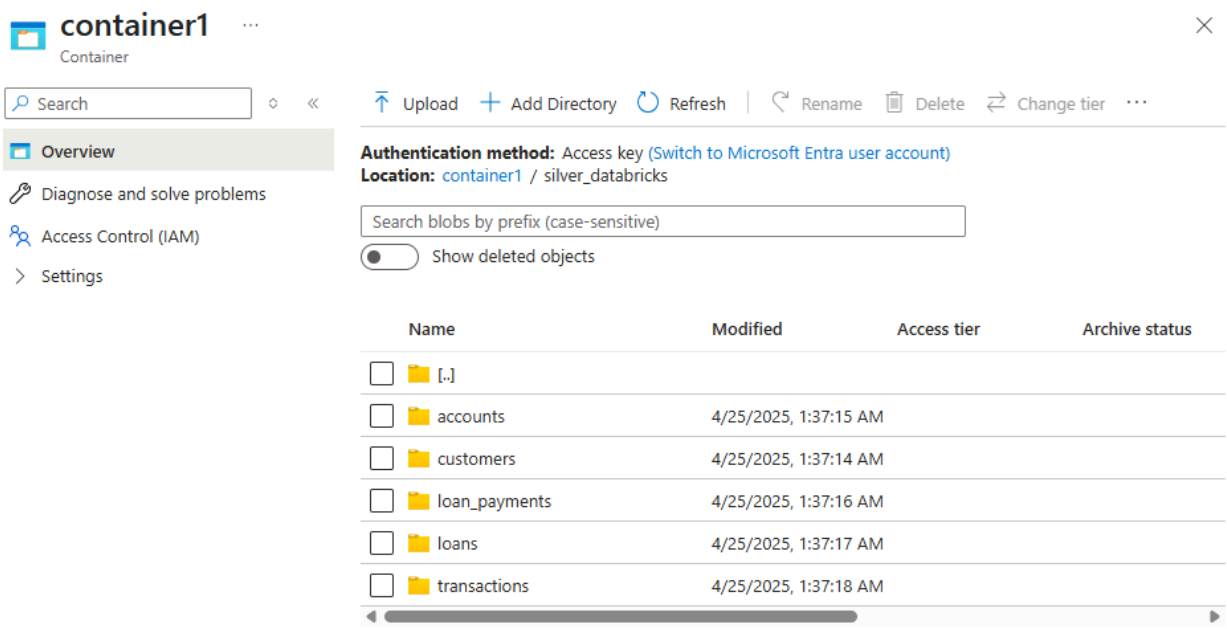
```python
df_customers_clean.write.mode("overwrite").parquet("/mnt/container1/silver_databricks/customers")
df_accounts_clean.write.mode("overwrite").parquet("/mnt/container1/silver_databricks/accounts")
df_loan_payments_clean.write.mode("overwrite").parquet("/mnt/container1/silver_databricks/loan_payments")
df_loans_clean.write.mode("overwrite").parquet("/mnt/container1/silver_databricks/loans")
df_transactions_clean.write.mode("overwrite").parquet("/mnt/container1/silver_databricks/transactions")
```

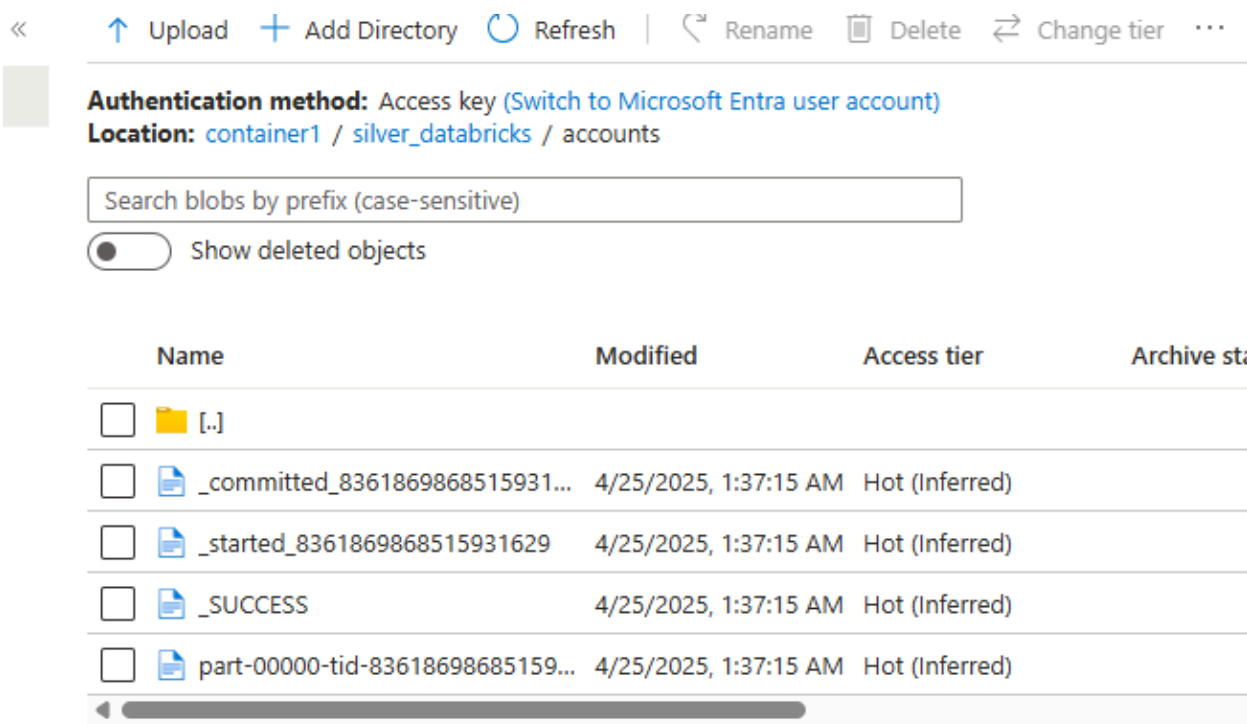
(10) Spark Jobs

Here's the silver_databricks folder:

Name	Modified	Access tier	Archive s
bronze_databricks	4/25/2025, 12:29:27 ...		
bronze_folder	4/23/2025, 1:05:33 AM		
bronzeFiles	4/23/2025, 11:54:08 ...		
silver_databricks	4/25/2025, 1:37:14 AM		

All the 5 files in parquet format:


container1
Container
Search
Overview
Diagnose and solve problems
Access Control (IAM)
Settings
Upload
Add Directory
Refresh
Rename
Delete
Change tier
Authentication method: Access key (Switch to Microsoft Entra user account)
Location: container1 / silver_databricks
Search blobs by prefix (case-sensitive)
Show deleted objects
Name
Modified
Access tier
Archive status
[..]
accounts 4/25/2025, 1:37:15 AM
customers 4/25/2025, 1:37:14 AM
loan_payments 4/25/2025, 1:37:16 AM
loans 4/25/2025, 1:37:17 AM
transactions 4/25/2025, 1:37:18 AM


Upload
Add Directory
Refresh
Rename
Delete
Change tier
Authentication method: Access key (Switch to Microsoft Entra user account)
Location: container1 / silver_databricks / accounts
Search blobs by prefix (case-sensitive)
Show deleted objects
Name
Modified
Access tier
Archive sta
[..]
_committed_8361869868515931... 4/25/2025, 1:37:15 AM Hot (Inferred)
_started_8361869868515931629 4/25/2025, 1:37:15 AM Hot (Inferred)
_SUCCESS 4/25/2025, 1:37:15 AM Hot (Inferred)
part-00000-tid-83618698685159... 4/25/2025, 1:37:15 AM Hot (Inferred)

Next step is to join all the files on the common column and store in Delta format in ADLS storage account.
The steps are below:

1. **As the customers, accounts and loansn all have common column "customer_id". Thus, can be joined.**
2. **This joined df can be joined with transaction since both have account_id common.**
3. **Lastly, loan_payment can be joined on loan_id.**

The below Screen Short shows the commands:

```python
# Now join with transactions on account_id
joined_df = joined_df.join(
    df_transactions_clean, on="account_id", how="inner"
)


# Join with loans on customer_id
joined_df = joined_df.join(
    df_loans_clean, on="customer_id", how="inner"
)


# Finally, join with loan_payments on loan_id
joined_df = joined_df.join(
    df_loan_payments_clean, on="loan_id", how="inner"
)
display(joined_df)
```

(10) Spark Jobs

joined_df: pyspark.sql.dataframe.DataFrame = [loan_id: integer, customer_id: integer ... 19 more fields]

Table +

	loan_id	customer_id	account_id	first_name	last_name	address	city	state	zip	account_type	balance
1	30	32	30	Sophia	Morris	3131 Oak Dr	Belleville	ON	K8N0A1	Checking	3100.5
2	4	34	4	Olivia	Reed	3333 Birch Blvd	Orillia	ON	L3V0A1	Checking	3000.25
3	26	25	26	Daniel	Campbell	2424 Willow Rd	St. Catharines	ON	L2R0A1	Checking	2800.5
4	74	43	74	Joseph	Cox	4242 Cedar Ln	Aurora	ON	L4G0A1	Checking	7500.5
5	11	3	11	Michael	Johnson	789 Oak Dr	Montreal	QC	H1A1A1	Savings	1100.75
6	40	19	40	Christopher	Baker	1818 Pine Rd	Thunder Bay	ON	P7A0A1	Checking	4100

Now this joined data we have to store in the form of Delta.

Just now (13s) 11 Python

```python
joined_df.write.format("delta").mode("overwrite").save("/mnt/container1/silver_databricks/joined_data")
```

Generate (Ctrl + I)

(13) Spark Jobs

Here's the successfully running command

Got joined_data folder:

Name	Modified	Access tier	Archive status	Blob type
accounts	4/25/2025, 1:37:15 AM			
customers	4/25/2025, 1:37:14 AM			
joined_data	4/27/2025, 12:27:59 AM			
loan_payments	4/25/2025, 1:37:16 AM			
loans	4/25/2025, 1:37:17 AM			
transactions	4/25/2025, 1:37:18 AM			

Containers >

r1 ...

Upload | Add Directory | Refresh | Rename | Delete | Change tier | Acquire lease | Break lease | Give feedback

Authentication method: Access key (Switch to Microsoft Entra user account)
Location: container1 / silver_databricks / joined_data

Search blobs by prefix (case-sensitive)

Show deleted objects

Name	Modified	Access tier	Archive status	Blob type	Size	Lease state
_delta_log	4/27/2025, 12:27:59 AM					-
part-00000-442fec65-e7bb-49d6-9891-4f1af34dcce1.c000.snappy.parquet	4/27/2025, 12:28:02 AM	Hot (Inferred)		Block blob	12.3 KiB	Available

Step 3: Storing the joined data as SCD Type 1

1. **First of all, we have to create a SCD Type1 table.**

12:18 AM (8s) 13 SQL

```sql
%sql
CREATE TABLE IF NOT EXISTS hive_metastore.default.accounts_scdtype1 (

    customer_id INT,
    account_id INT,
    account_type STRING,
    balance FLOAT,
    hashkey BIGINT,
    createdby STRING,
    createddate TIMESTAMP,
    updatedby STRING,
    updateddate TIMESTAMP


)
USING DELTA
LOCATION '/mnt/container1/gold_folder/accounts_scdtype1';



SELECT * FROM hive_metastore.default.accounts_scdtype1;
```

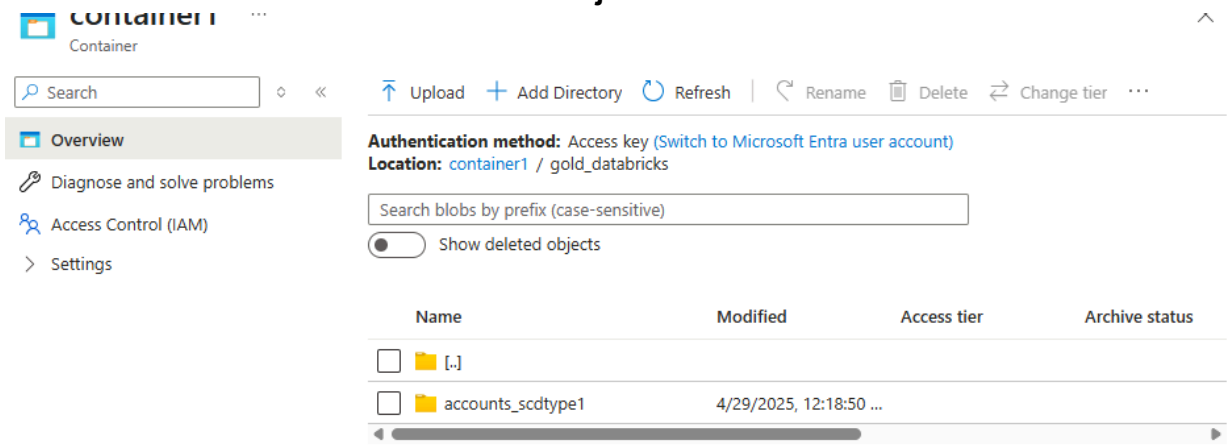
_sqldf: pyspark.sql.dataframe.DataFrame = [customer_id: integer, account_id: integer ... 7 more fields]

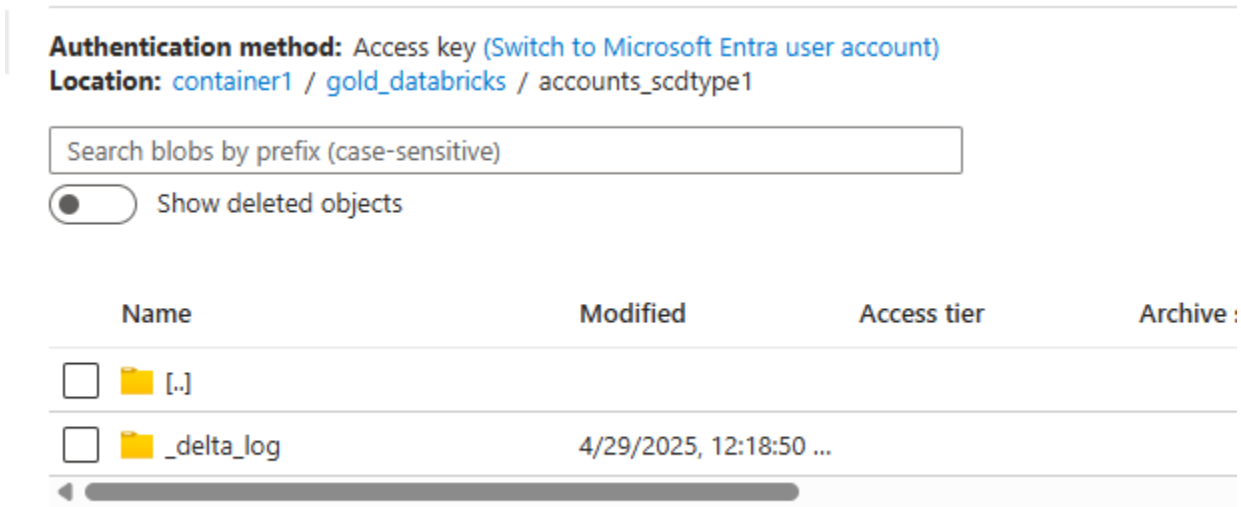
Table +

customer_id	account_id	account_type	balance	hashke

No rows returned

2. **Here's the delta table for accounts that we just created:**





3. **Adding Hashkey:**

2 minutes ago (1s) 16 Python

```python
# adding hash key
src_accounts = src_accounts.withColumn("hashkey", crc32(concat(*src_accounts.
columns)))
display(src_accounts)
```

(1) Spark Jobs

src_accounts: pyspark.sql.dataframe.DataFrame = [account_id: integer, customer_id: integer ... 3 more fields]

Table +

	t_id	customer_id	account_type	balance	hashkey
3	82	2	Checking	8300.5	3247491694
4	52	10	Checking	5300	2280475706
5	65	69	Savings	550.25	3693472785
6	76	22	Checking	7700	3359048414
7	62	35	Checking	6300.5	800230545
8	98	49	Checking	9900.5	3893038149
9	53	86	Savings	400.25	1685401127
10	66	26	Checking	6700.5	372242038
11	72	17	Checking	7300	1425050024
12	86	21	Checking	8700.5	2919860885

4. Defining source and the target path and reading the parquet file from the silver layer:

12:46 AM (1s) 14

```python
src_path = "/mnt/container1/silver_databricks/accounts"
tgt_path ="/mnt/container1/gold_databricks/accounts_scdtype1"
src_accounts = spark.read.format("parquet").load(src_path)
```

(1) Spark Jobs

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

5. After importing pyspark.sql functions, I added another column, hashkey:

✓ 01:02 AM (<1s) 15

```
from pyspark.sql.functions import *
```

✓ 01:02 AM (3s) 16

```
# adding hash key
src_accounts = src_accounts.withColumn("hashkey", crc32(concat(*src_accounts.
columns)))
display(src_accounts)
```

(1) Spark Jobs

src_accounts: pyspark.sql.dataframe.DataFrame = [account_id: integer, customer_id: integer ... 3 more fields]

Table +

	t_id	customer_id	account_type	balance	hashkey
52	20	21	Checking	2000	2020209643
53	71	73	Savings	625.75	2590228591
54	13	29	Savings	1300.25	3025925162
55	64	12	Checking	6500	1796018383
56	61	52	Savings	500.25	2662432370
57	32	9	Checking	3300	2957212870
58	89	54	Savings	850.25	540724915
59	31	71	Savings	125.75	352326581
60	90	38	Checking	9100.5	2053712775
61	10	92	Checking	1800.5	533177005

6. IMport the Delta Table class:

12:46 AM (2s) 17

```
from delta.tables import DeltaTable
deltatable = DeltaTable.forPath(spark, tgt_path)
deltatable.toDF().show
```

```
<bound method DataFrame.show of DataFrame[customer_id: int, account_id: int, account_typ
e: string, balance: float, hashkey: bigint, createdby: string, createddate: timestamp, u
pdatedby: string, updateddate: timestamp]>
```

7. Merging the delta table with dataframe and map the columns

01:06 AM (<1s) 18

```
from pyspark.sql.functions import col
src_accounts = src_accounts.alias("account").join(deltatable.toDF().alias("tgt"),
((col("account.account_id")==col("tgt.account_id")) & (col("account.hashkey")==col
("tgt.hashkey"))),"anti").select("account.*")
```

src_accounts: pyspark.sql.dataframe.DataFrame = [account_id: integer, customer_id: integer ... 3 mo
fields]

01:14 AM (13s) 19 Python

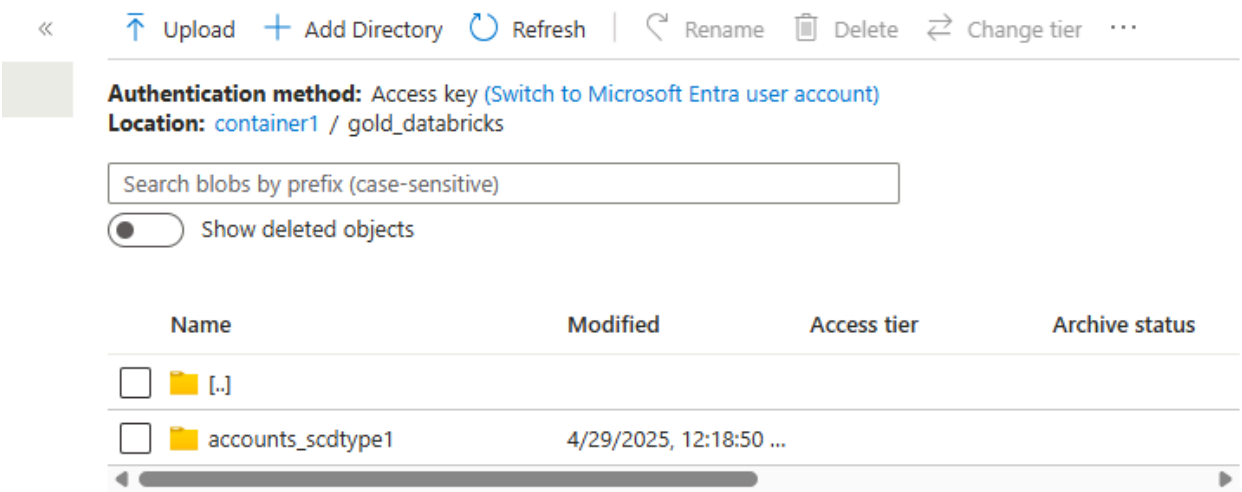
```
deltatable.alias("tgt").merge(
    src_accounts.alias("src"),
    "tgt.account_id = src.account_id")\
  .whenMatchedUpdate(set={"tgt.account_id": "src.account_id ","tgt.customer_id":
  "src.customer_id","tgt.account_type": "src.account_type","tgt.balance": "src.
  balance","tgt.hashkey": "src.hashkey", "tgt.updatedby":lit("databricks"), "tgt.
  updateddate":current_timestamp()})\
  .whenNotMatchedInsert(values={"tgt.account_id": "src.account_id ","tgt.
  customer_id": "src.customer_id","tgt.account_type": "src.account_type","tgt.
  balance": "src.balance","tgt.hashkey": "src.hashkey", "tgt.createdby":lit
  ("databricks"), "tgt.createddate":current_timestamp(), "tgt.updatedby":lit
  ("databricks"), "tgt.updateddate":current_timestamp()})\
      .execute()
display(spark.read.format("delta").load(tgt_path))
```

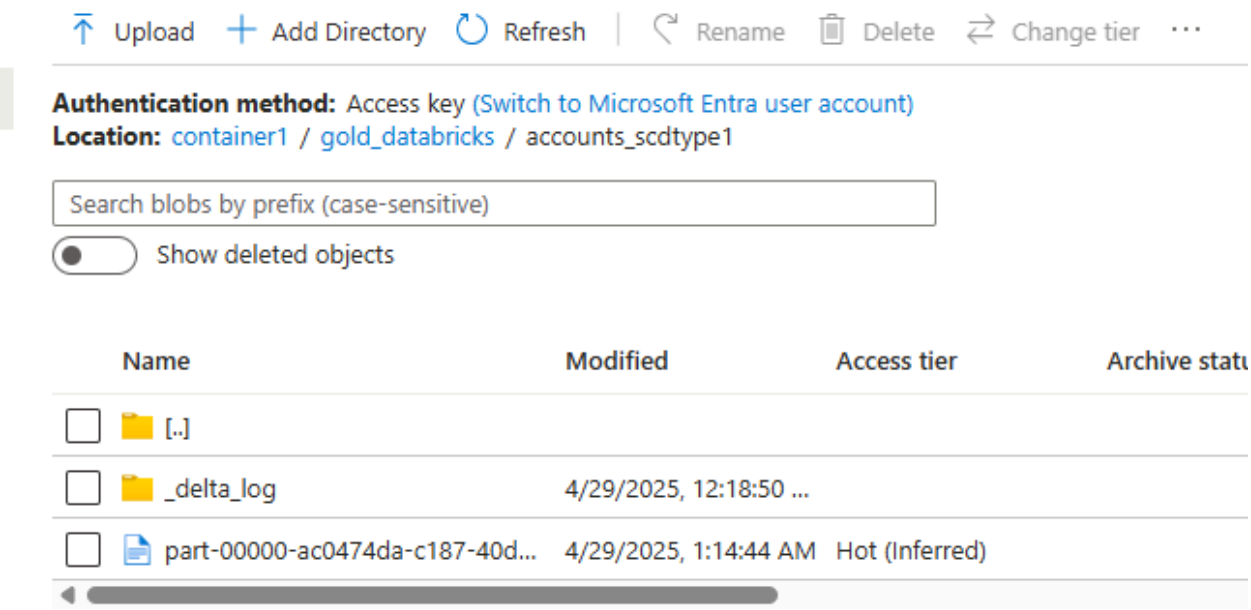
(8) Spark Jobs

Table +

	customer_id	accou...	account_type	balance	has
1	27	36	Checking	3700	1
2	65	85	Savings	800.25	3
3	2	82	Checking	8300.5	3

8. Here's the file has created in the gold_databricks folder:



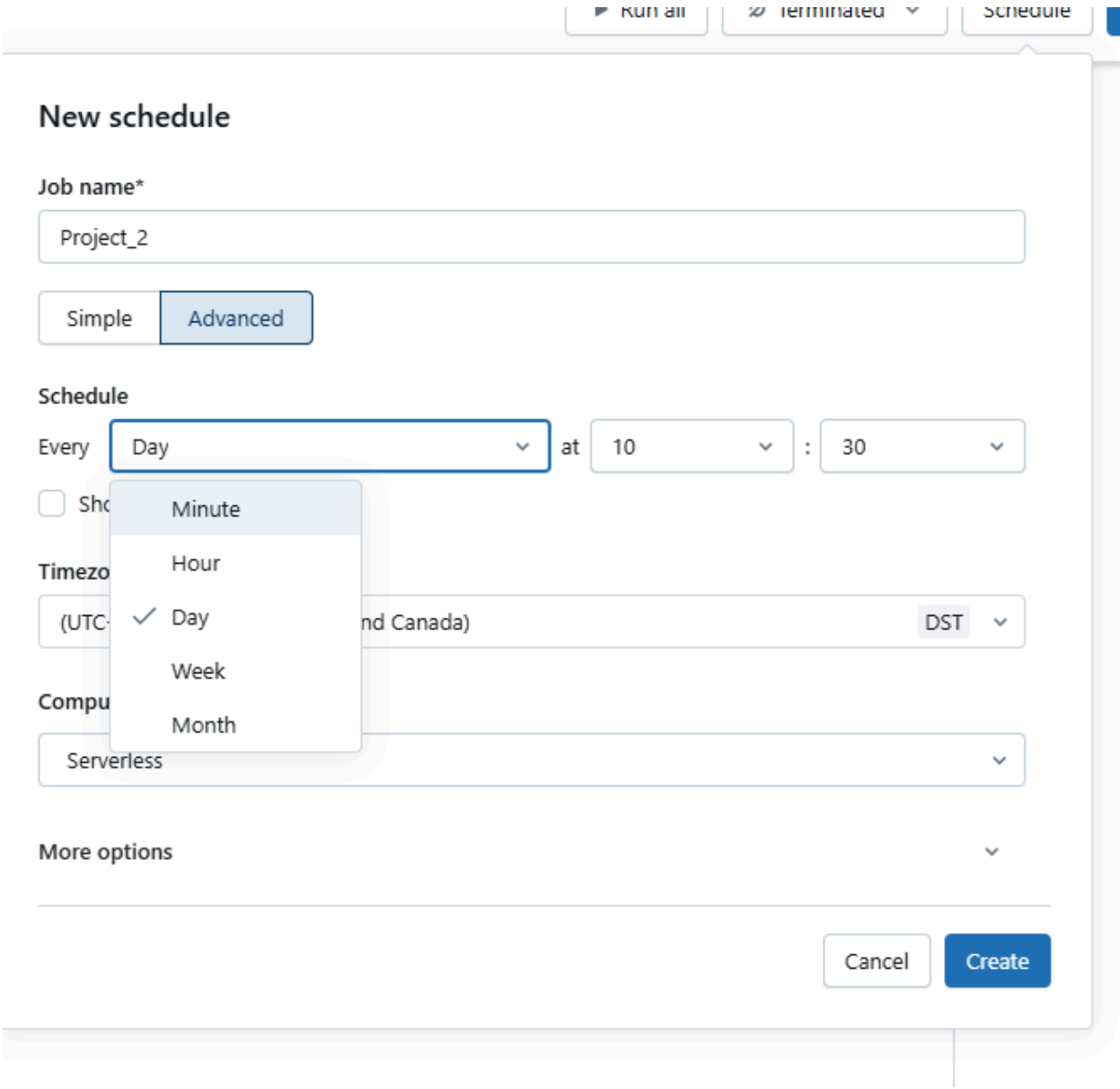


9. The same steps have to repeat for all other 4 files.

Next step is to Schedule the pipeline:

1. **There is an option “Schedule at the top right corner.**
2. **Give the Job Name.**

3. **Select the appropriate time. Even can select how often the pipeline should be executed:**



4. **Select Timezone -> Create**