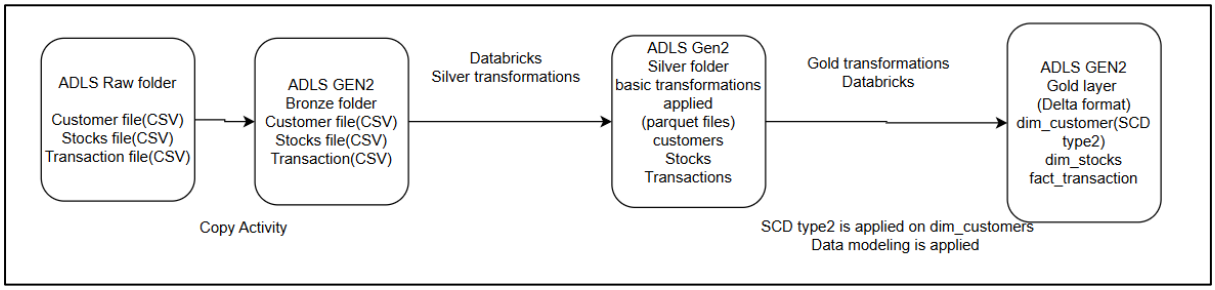
Financial Data Lakehouse on Azure with Medallion Architecture

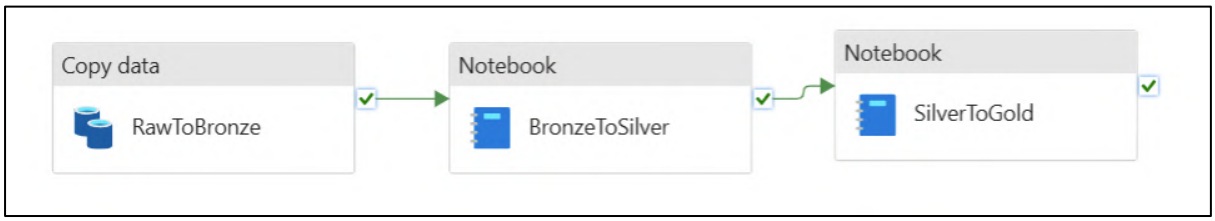
Objective:

To build a scalable, maintainable, and analytics-ready financial data platform using a Medallion architecture (Bronze → Silver → Gold) on Azure Data Lake Storage Gen2 using Apache Spark (PySpark), with support for SCD Type 2, quality validation, and dimensional modeling

Architecture



Pipeline



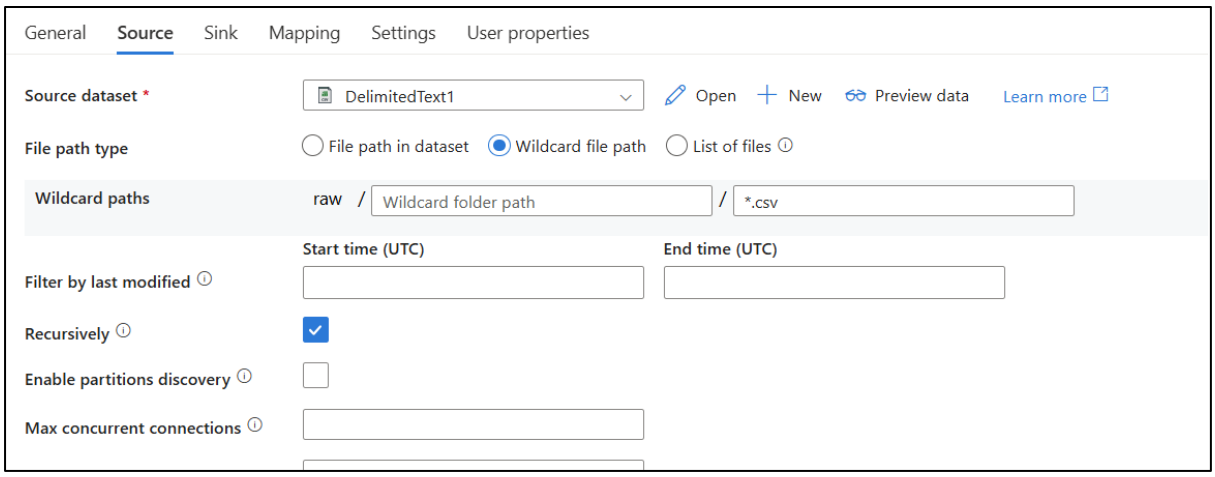
1. Bronze Layer – Raw Ingestion

Purpose: Store raw, unprocessed CSV files from source systems.

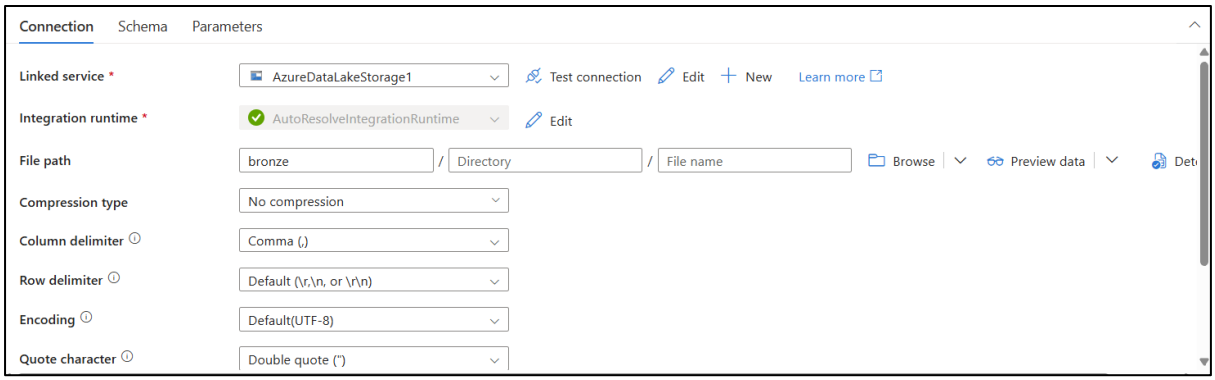
Sources Ingested:

- raw_customer_data_consistent.csv
- raw_stock_data.csv
- raw_transaction_data_consistent.csv

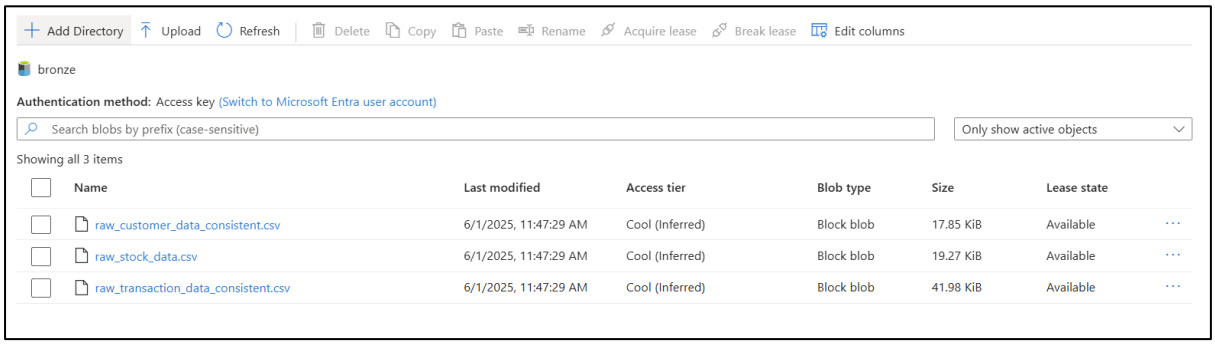
Raw to Bronze source and Sink screenshots.



Sink dataset:



Files in bronze folder -ADLS Gen2



2. Silver Layer – Data Cleaning & Standardization

Purpose: Apply data cleansing, transformation, and standard formatting for analytics readiness.

Transformations Done:

Customer Data:

- Remove duplicates based on Customer_ID.
- Trim and clean Name, Email, and Phone.
- Normalize date format for DOB.
- Lowercase and title-case where appropriate.
- Validate presence of critical fields.

Transaction Data:

- Filter transactions with valid Quantity, Price.
- Parse Transaction_Date.
- Fill missing values and remove nulls in critical fields.
- Deduplicate on Transaction_ID.

Stock Data:

- Deduplicate on Date + Stock_Symbol.
- Normalize Date field format.

Output Format:

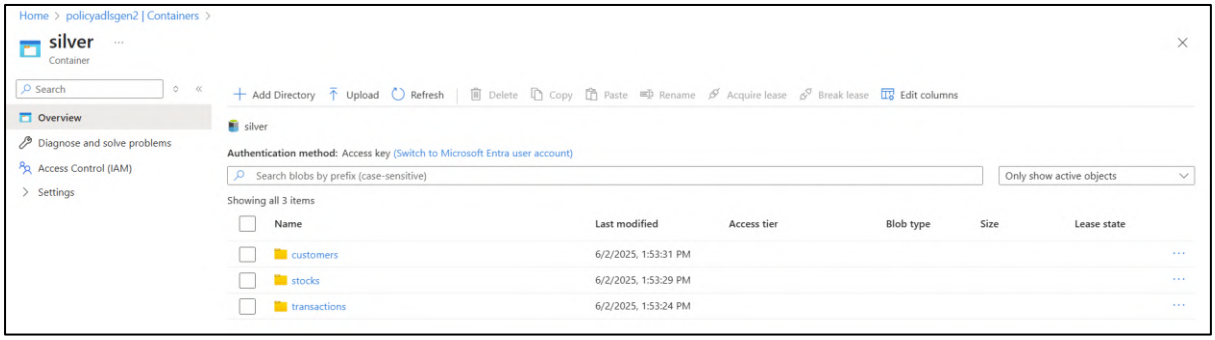
- Saved as Parquet files to ADLS Gen2 silver container.

Bronze to Silver transformation in pyspark

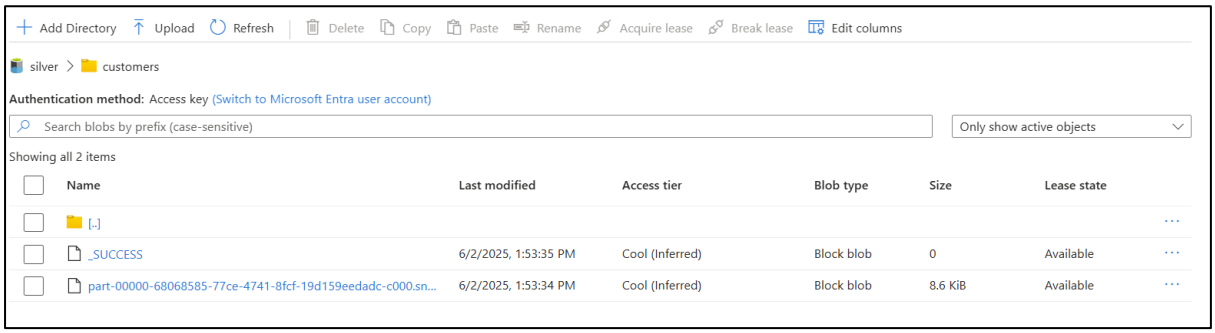
```
#import libraries

from pyspark.sql.functions import *

from pyspark.sql.window import *

from delta.tables import *

# Read bronze_nifty_companies from ADLS or Delta

df_br_cust = spark.read.format("csv") \

.option("header","true") \

.option("inferschema","true") \

.load("abfss://bronze@policyadlsgen2.dfs.core.windows.net/raw_customer_data_consistent.csv")

df_br_stock = spark.read.format("csv") \

.option("header","true") \

.option("inferschema","true") \

.load("abfss://bronze@policyadlsgen2.dfs.core.windows.net/raw_stock_data.csv")

df_br_trans = spark.read.format("csv") \

.option("header","true") \

.option("inferschema","true") \

.load("abfss://bronze@policyadlsgen2.dfs.core.windows.net/raw_transaction_data_consistent.csv")


# transactions clean up

df_tran_clean = df_br_trans \

.withColumn("Transaction_Date", to_date(col("Transaction_Date"), "yyyy-MM-dd")) \

.filter(col("Quantity") > 0) \

.filter(col("Price") > 0) \

.fillna("Unknown",subset=["Customer_ID","Product"]) \

.dropna(subset=["Transaction_Type"]) \

.dropDuplicates(["Transaction_ID"])


#clean up stocks

df_stock_clean = df_br_stock \

.withColumn("Date",to_date(col("Date"),"YYYY-MM-DD")) \

.dropDuplicates(["Date","Stock_symbol"])
```

```
#cleaning customer data

df_cust_clean = df_br_cust.dropDuplicates(["Customer_ID"]) \

.withColumn("Name", trim("Name")) \

.withColumn("Email", lower(trim("Email"))) \

.withColumn("Phone", trim("Phone")) \

.withColumn("DOB", to_date("DOB", "yyyy-MM-dd")) \

.filter(col("DOB").isNotNull()) \

.dropna(subset=["Customer_ID", "Name", "DOB", "Email"]) \

.withColumn("Name", initcap("Name"))


#writing files to ADLS GEN2 silver folder

df_tran_clean.write.mode("overwrite").format("parquet").save("abfss://silver@policyadlsgen2.dfs.core.windows.net/transactions")

df_stock_clean.write.mode("overwrite").format("parquet").save("abfss://silver@policyadlsgen2.dfs.core.windows.net/stocks")

df_cust_clean.write.mode("overwrite").format("parquet").save("abfss://silver@policyadlsgen2.dfs.core.windows.net/customers")
```

Files from Silver layer - ADLS GEN2



Customer file in silver layer:



Stocks file in Silver layer:

Add Directory | Upload | Refresh | Delete | Copy | Paste | Rename | Acquire lease | Break lease | Edit columns

silver > stocks

Authentication method: Access key (Switch to Microsoft Entra user account)

Search blobs by prefix (case-sensitive) | Only show active objects

Showing all 2 items

Name	Last modified	Access tier	Blob type	Size	Lease state
[..]					
_SUCCESS	6/2/2025, 1:53:30 PM	Cool (Inferred)	Block blob	0	Available
part-00000-bf4b5e19-8328-4e1d-b8e7-bf9dabad9459-c000.s...	6/2/2025, 1:53:30 PM	Cool (Inferred)	Block blob	9.71 KiB	Available

Transaction file in Silver layer

Add Directory | Upload | Refresh | Delete | Copy | Paste | Rename | Acquire lease | Break lease | Edit columns

silver > transactions

Authentication method: Access key (Switch to Microsoft Entra user account)

Search blobs by prefix (case-sensitive) | Only show active objects

Showing all 2 items

Name	Last modified	Access tier	Blob type	Size	Lease state
[..]					
_SUCCESS	6/2/2025, 1:53:28 PM	Cool (Inferred)	Block blob	0	Available
part-00000-bf4f3e8f-7a04-459c-9726-dbb47dcb03f2-c000.sn...	6/2/2025, 1:53:27 PM	Cool (Inferred)	Block blob	27.2 KiB	Available

3. Gold Layer – Dimensional Modeling & Business Logic

Purpose: Create analytics-ready dimensional models with Slowly Changing Dimensions (SCD) support.

Dimensions:

Customer Dimension (dim_customer):

- Implements **SCD Type 2** using Delta Lake.
- Uses SHA-256 record_hash to detect changes.
- Tracks is_current, start_date, end_date.

Stock Dimension (dim_stock):

- Tracks only the latest available metadata (Latest_Metadata_Date) using a window functions

Fact Table:

- **Transaction Fact (fact_transaction)**:
- Enriched with customer and stock dimension lookups.
- Adds derived columns: Year, Month, Day from Transaction_Date.
- Supports time-based analytics and joins with dimensions.

Output Format:

- Saved as Delta Lake format to ADLS Gen2 gold container.

Silver to Gold transformations:

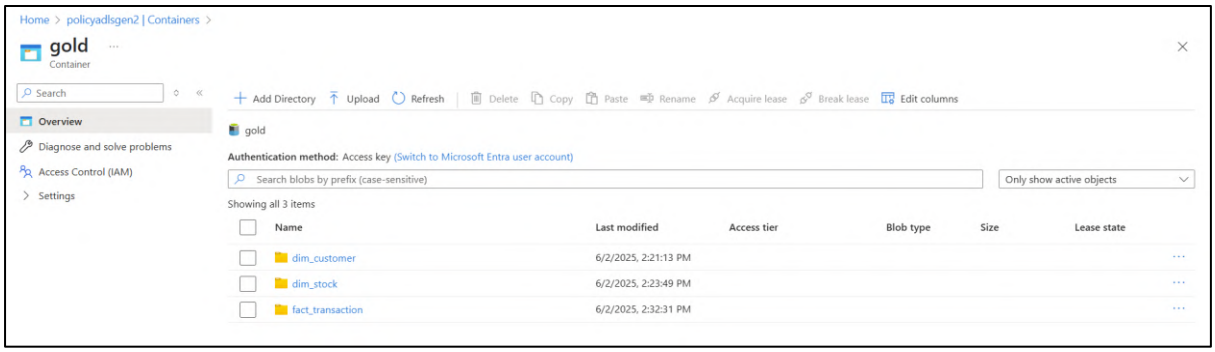
```
#Read files from silver files

silver_path = "abfss://silver@policyadlsgen2.dfs.core.windows.net/"

df_g_cust = spark.read.format("parquet").load(f"{silver_path}customers")

df_g_transactions = spark.read.format("parquet").load(f"{silver_path}transactions")

df_g_stocks = spark.read.format("parquet").load(f"{silver_path}/stocks")


# Add SCD Type 2 metadata columns to incoming data

df_cust_transformed = df_g_cust.withColumn("record_hash", sha2(concat_ws("||",
*df_g_cust.columns), 256)) \

.withColumn("is_current", lit(True)) \

.withColumn("start_date", current_timestamp()) \

.withColumn("end_date", lit(None).cast("timestamp"))
```

```
# Define target table path for Gold dim_customer
gold_cust_path = "abfss://gold@policyadlsgen2.dfs.core.windows.net/dim_customer/"

# Check if Gold table exists
if DeltaTable.isDeltaTable(spark, gold_cust_path):
delta_gold = DeltaTable.forPath(spark, gold_cust_path)
df_existing = delta_gold.toDF().filter("is_current = True")

# Join on business key (e.g., Customer_ID) and hash comparison
join_cond = [df_existing["Customer_ID"] == df_cust_transformed["Customer_ID"]]
df_changes = df_existing.join(df_cust_transformed, join_cond, "inner") \
.filter(df_existing["record_hash"] = df_cust_transformed["record_hash"]) \
.drop(df_existing["Customer_ID"],
df_existing["Name"],
df_existing["DOB"],
df_existing["Email"],
df_existing["Phone"],
df_existing["record_hash"],
df_existing["is_current"],
df_existing["start_date"],
df_existing["end_date"])
if df_changes.count() > 0:

# Expire old records
delta_gold.alias("tgt").merge(
df_changes.alias("src"),
"tgt.Customer_ID = src.Customer_ID AND tgt.is_current = true"
).whenMatchedUpdate(set={
"is_current": lit(False),
"end_date": current_timestamp()
}).execute()
```

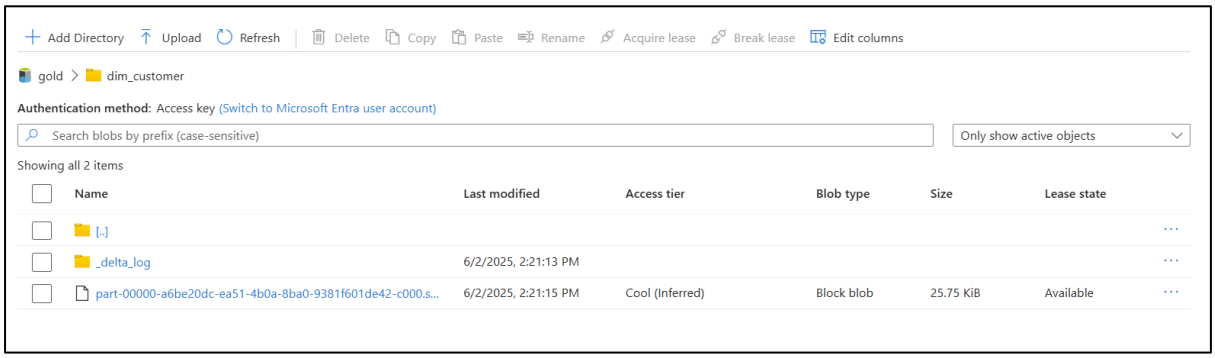
```
# Insert new version
df_cust_transformed.alias("new_data") \
.join(df_existing.select("Customer_ID"), "Customer_ID", "left_anti") \
.unionByName(df_changes) \
.write.format("delta").mode("append").save(gold_cust_path)
else:

# First time load
df_cust_transformed.write.format("delta").mode("overwrite").save(gold_cust_path)

# 1. Get latest metadata per Stock_Symbol based on Date
window_spec = Window.partitionBy("Stock_Symbol").orderBy(col("Date").desc())
df_dim_stock = df_g_stocks.withColumn("row_num", row_number().over(window_spec)) \
.filter("row_num = 1") \
.select("Stock_Symbol", "Date") \
.withColumnRenamed("Date", "Latest_Metadata_Date")

# 2. Save to Gold layer
df_dim_stock.write.format("delta").mode("overwrite") \
.save("abfss://gold@policyadlsgen2.dfs.core.windows.net/dim_stock/")

# Read Gold-layer customer and stock dimensions (latest SCD state)
df_dim_customer =
spark.read.format("delta").load("abfss://gold@policyadlsgen2.dfs.core.windows.net/dim_cust
omer/") \
.filter("is_current = true")
df_dim_stock =
spark.read.format("delta").load("abfss://gold@policyadlsgen2.dfs.core.windows.net/dim_stoc
k/")
```

```
# Join with dimension tables using business keys
df_fact = df_g_transactions \
.join(df_dim_customer.select("Customer_ID"), on="Customer_ID", how="inner") \
.join(df_dim_stock.select("Stock_Symbol"), on="Stock_Symbol", how="inner") \
.withColumn("Year", year("Transaction_Date")) \
.withColumn("Month", month("Transaction_Date")) \
.withColumn("Day", dayofmonth("Transaction_Date"))

# Select fact table schema
df_fact_selected = df_fact.select(
"Transaction_ID",
"Customer_ID",
"Stock_Symbol",
"Transaction_Date",
"Transaction_Type",
"Quantity",
"Price",
"Product",
"Year",
"Month",
"Day"
)

# Write fact table to Gold layer in Delta format
df_fact_selected.write.format("delta").mode("overwrite") \
.save("abfss://gold@policyadlsgen2.dfs.core.windows.net/fact_transaction/")
```

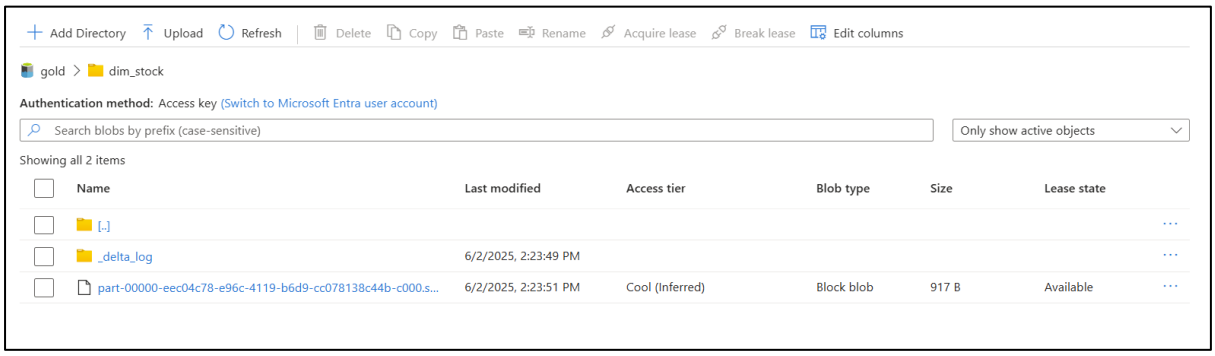
Files in Gold Layer:



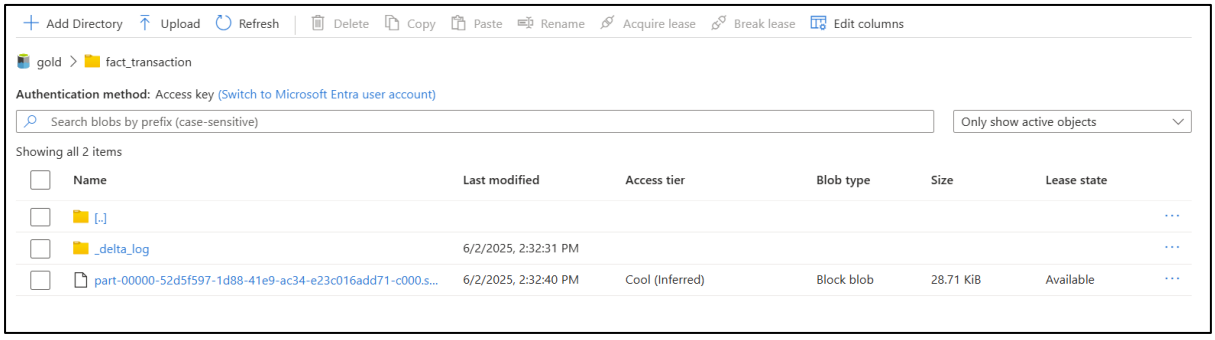
Customer file in Gold layer:



Stock file in Gold layer:



Transactions file in Gold layer



Next day- New Customer

- Some existing records were updated

- new records were added

df_gold: pyspark.sql.dataframe.DataFrame = [Customer_ID: string, Name: string ... 7 more fields]

Table

	Customer_ID	Name	DOB	Email	Phone	record_hash
1	CUS2007	Customer_cust20...	1970-06-18	cust1013@example.co...	919361914869	ebbb99a0792cb795809fa2073ca6787109a56a65a89ef0375f8cc52a82a94af5
2	CUST1000	Customer_cust10...	2000-05-28	cust1000@example.co...	918416115050	bdb24887b9c25ef8d392818bc61817bfeb4491d20de08dda34b8fe2531400
3	CUST1000	Customer_cust10...	2000-05-28	divya@example.com	918416115050	4e119985416dea4106e56d2611bc4446bf0aa03c11cb068774e5ece1111c95
4	CUST1001	Customer_cust10...	1979-08-28	cust1001@example.co...	919534702881	4e8b4064ed93608176511b9caad7b12ba46e72266b014ab67da513bdc1aba
5	CUST1001	Customer_cust10...	1979-08-28	megha@example.com	919534702881	774a7ad1d8c71e8cb369888658a49a12a981ae76171630ee2e309d7839f7cf8
6	CUST1002	Customer_cust10...	1968-02-17	cust1002@example.co...	917725188947	b4ca9c821e50d8ffc3f09eb87a05946d3c5392d9801722e3679def838b9553f8
7	CUST1002	Customer_cust10...	1968-02-17	srihitha@example.com	917725188947	fbb3244be8c6cd73e472d9f3a35d91708b9a58c8d6bf5e583d379cd5e439c9
8	CUST1003	Customer_cust10...	1968-10-25	cust1003@example.co...	917193540775	d67a216c9c6459569d33c27a7de8e5ee399146252ca874632f5f17c74d9461e
9	CUST1003	Customer_cust10...	1968-10-25	test@example.com	917193540775	c665c329d6943d6ed4904cd0d1ca88c05526ba3f6f0062dd9f1a40ae1fe5c5e
10	CUST1004	Customer_cust10...	1967-12-13	cust1004@example.co...	919716735458	6f04996feb59b01aea5e1818b026d19b7d162c5fe4e26a41918cfd509827749
11	CUST1004	Customer_cust10...	1967-12-13	test1@example.com	919716735458	630588241db1049e48f1a2a758291164ac18060e729e85a6ad1ade689d4763
12	CUST1005	Customer_cust10...	1982-04-21	cust1005@example.co...	917650147854	d012fc7129ae21e40b2358f087a4e01b02e0143dc854088111adbe71f6dca1f
13	CUST1005	Customer_cust10...	1982-04-21	test2@example.com	917650147854	5477e2d848bd5b0e71673bf850c343128c9ffd0ca152b8dabefad62fda5dcb1
14	CUST1006	Customer_cust10...	1984-04-01	cust1006@example.co...	919961593877	e838309000e7d9a0763d4163033fca52f0061daa415f1ed0632fa40db330c39

The existing records had some update, so the existing record was marked "is_current" False and end-date as processed date.

Table

	Phone	record_hash	is_current	start_date	end_date
1	361914869	ebbb99a0792cb795809fa2073ca6787109a56a65a89ef0375f8cc52a82a94af5	true	2025-06-02T21:07:20.711+00:...	null
2	416115050	bdb24887b9c25ef8d392818bc61817bfeb4491d20de08dda34b8fe2531400908	false	2025-06-02T18:21:13.977+00:...	2025-06-02T21:06:50.647+00:...
3	416115050	4e119985416dea4106e56d2611bc4446bf0aa03c11cb068774e5ece1111c9519	true	2025-06-02T21:07:20.711+00:...	null
4	534702881	4e8b4064ed93608176511b9caad7b12ba46e72266b014ab67da513bdc1ababc2	false	2025-06-02T18:21:13.977+00:...	2025-06-02T21:06:50.647+00:...
5	534702881	774a7ad1d8c71e8cb369888658a49a12a981ae76171630ee2e309d7839f7cf8d	true	2025-06-02T21:07:20.711+00:...	null
6	725188947	b4ca9c821e50d8ffc3f09eb87a05946d3c5392d9801722e3679def838b9553f8	false	2025-06-02T18:21:13.977+00:...	2025-06-02T21:06:50.647+00:...
7	725188947	fbb3244be8c6cd73e472d9f3a35d91708b9a58c8d6bf5e583d379cd5e439c927	true	2025-06-02T21:07:20.711+00:...	null
8	193540775	d67a216c9c6459569d33c27a7de8e5ee399146252ca874632f5f17c74d9461e8	false	2025-06-02T18:21:13.977+00:...	2025-06-02T21:06:50.647+00:...
9	193540775	c665c329d6943d6ed4904cd0d1ca88c05526ba3f6f0062dd9f1a40ae1fe5c5e8	true	2025-06-02T21:07:20.711+00:...	null
10	716735458	6f04996feb59b01aea5e1818b026d19b7d162c5fe4e26a41918cfd5098277494	false	2025-06-02T18:21:13.977+00:...	2025-06-02T21:06:50.647+00:...
11	716735458	630588241db1049e48f1a2a758291164ac18060e729e85a6ad1ade689d4763bd	true	2025-06-02T21:07:20.711+00:...	null
12	650147854	d012fc7129ae21e40b2358f087a4e01b02e0143dc854088111adbe71f6dca1ff	false	2025-06-02T18:21:13.977+00:...	2025-06-02T21:06:50.647+00:...
13	650147854	5477e2d848bd5b0e71673bf850c343128c9ffd0ca152b8dabefad62fda5dcb1a	true	2025-06-02T21:07:20.711+00:...	null
14	961593877	e838309000e7d9a0763d4163033fca52f0061daa415f1ed0632fa40db330c39f	true	2025-06-02T18:21:13.977+00:...	null
15					

264 rows | 2.77s runtime

Refreshed 1 minute ago

	Customer_ID	Name	DOB	Email	Phone	record_hash
251	CUST1243	Customer_cust12...	1985-03-26	cust1243@example.co...	917302114479	f4a9e159aed6468c96a8a508ba4329efb46dfded24c7544456a56a5e169aae1
252	CUST1244	Customer_cust12...	1961-06-26	cust1244@example.co...	918826546195	93ee30576277546bcaad547956dcf5c9886dedc0590cd56ab5c84cc9bce72e
253	CUST1245	Customer_cust12...	1973-03-16	cust1245@example.co...	918992454569	7a6487a2cf3c15707cf5b3bb1e05ad629ffdd54bd6cebe2f4ef934d6335c4bf2
254	CUST1246	Customer_cust12...	1981-07-19	cust1246@example.co...	919430307649	0b6f9fbba0b12d46802461d0e74a4d8b9b8df1f05100c04f67382069bd1f621
255	CUST1247	Customer_cust12...	1967-06-15	cust1247@example.co...	917021183910	995ca302f315680ab9bfd1ec641c2c80ac749c5274d29d35dfd5ab52873184e
256	CUST1248	Customer_cust12...	1980-01-07	cust1248@example.co...	919131303358	3587556cb7b706dafba144ec33bb7bcff4529d6473d030e0c03f0b67f55f712
257	CUST1249	Customer_cust12...	1965-04-27	cust1249@example.co...	917564138621	6f91b3e355acaa8ec6ad8e31ddec04424c0d0ef47dfc92d541711bf1839b2cb
258	CUST2000	Customer_cust20...	1984-04-01	cust1006@example.co...	919961593877	078ccfe039381d96e0332ccf49f0045967a06a910ec0a1947fd57f4f4bddd939
259	CUST2001	Customer_cust20...	1970-12-29	cust1007@example.co...	919918326269	c5703713577c074542d627f6ff69a97ac2ebf349cd3b13efb40fef692029d52d
260	CUST2002	Customer_cust20...	1996-09-04	cust1008@example.co...	919400203003	5428b4c2afbea171c5201a77f65f0e19b90d3de98ec20a1eb25dbf4a253eb13
261	CUST2003	Customer_cust20...	1961-03-03	cust1009@example.co...	918688263064	c45711e811e329753cee65f8272a333c5823364d756dfbff1dbc30b2c6e889c
262	CUST2004	Customer_cust20...	2000-11-16	cust1010@example.co...	918529933728	ff5795e7e7e4c7485ffdd2f1f0fa60b3c1e85ea09fd05e658f3e14ce708c13ea
263	CUST2005	Customer_cust20...	1988-09-21	cust1011@example.co...	918459763803	e2db6506d1d1c4eabb546144bf0c5d6bbf185ca5d4fa71f0bc6538b7ab68f27
264	CUST2006	Customer_cust20...	1973-10-14	cust1012@example.co...	917735480322	2a1657a0e2700ae10f0dc310546bb54a58b528129a2d96d266e6218d101db

264 rows | 2.77s runtime

Refreshed 2 minutes ago

The brand new records were added with start date as ingestion date and Is_current as ”True” and end-date is null.

Table

	Phone	record_hash	is_current	start_date	end_date
251	302114479	f4a9e159aed6468c96a8a508ba4329efb46dfded24c7544456a56a5e169aae1a	true	2025-06-02T18:21:13.977+00:...	null
252	826546195	93ee30576277546bcaad547956dcf5c9886dedc0590cd56ab5c84cc9bce72ea3	true	2025-06-02T18:21:13.977+00:...	null
253	992454569	7a6487a2cf3c15707cf5b3bb1e05ad629ffdd54bd6cebe2f4ef934d6335c4bf2	true	2025-06-02T18:21:13.977+00:...	null
254	430307649	0b6f9fbba0b12d46802461d0e74a4d8b9b8df1f05100c04f67382069bd1f6219	true	2025-06-02T18:21:13.977+00:...	null
255	021183910	995ca302f315680ab9bfd1ec641c2c80ac749c5274d29d35dfd5ab52873184ee	true	2025-06-02T18:21:13.977+00:...	null
256	131303358	3587556cb7b706dafba144ec33bb7bcff4529d6473d030e0c03f0b67f55f712a	true	2025-06-02T18:21:13.977+00:...	null
257	564138621	6f91b3e355acaa8ec6ad8e31ddec04424c0d0ef47dfc92d541711bf1839b2cbb	true	2025-06-02T18:21:13.977+00:...	null
258	961593877	078ccfe039381d96e0332ccf49f0045967a06a910ec0a1947fd57f4f4bddd939	true	2025-06-02T21:07:20.711+00:...	null
259	918326269	c5703713577c074542d627f6ff69a97ac2ebf349cd3b13efb40fef692029d52d	true	2025-06-02T21:07:20.711+00:...	null
260	400203003	5428b4c2afbea171c5201a77f65f0e19b90d3de98ec20a1eb25dbf4a253eb13d	true	2025-06-02T21:07:20.711+00:...	null
261	688263064	c45711e811e329753cee65f8272a333c5823364d756dfbff1dbc30b2c6e889ca	true	2025-06-02T21:07:20.711+00:...	null
262	529933728	ff5795e7e7e4c7485ffdd2f1f0fa60b3c1e85ea09fd05e658f3e14ce708c13ea	true	2025-06-02T21:07:20.711+00:...	null
263	459763803	e2db6506d1d1c4eabb546144bf0c5d6bbf185ca5d4fa71f0bc6538b7ab68f27b	true	2025-06-02T21:07:20.711+00:...	null
264	735480322	2a1657a0e2700ae10f0dc310546bb54a58b528129a2d96d266e6218d101db070	true	2025-06-02T21:07:20.711+00:...	null

264 rows | 2.77s runtime

Refreshed 3 minutes ago

Key Features

- Clean separation of layers using Medallion Architecture.
- Robust SCD Type 2 handling in Customer dimension.
- Daily incremental ready transformations (supports idempotency and hash diffing).
- Modular and extensible ETL logic.
- Stored in Delta & Parquet formats optimized for analytical workloads.