RETAIL – CASE - STUDY

NAME: Govind Polireddi **Employee Id**: AS1552

Problem Statement

Retail companies struggle with integrating **in-store POS transactions, online orders, and inventory systems** into a single analytics-ready platform.
They need:

- **Batch processing** for daily sales reconciliation.
- **Real-time ingestion** for fraud detection & stock alerts.
- **Cloud-native scalability** to support peak season sales.
- **Unified warehouse (Snowflake / Synapse)** for advanced reporting
- **Monitoring & error handling** for operational resilience.
- **Data security** for customer PII.
- **Cost transparency** for infrastructure & query optimization.

APPROACH:

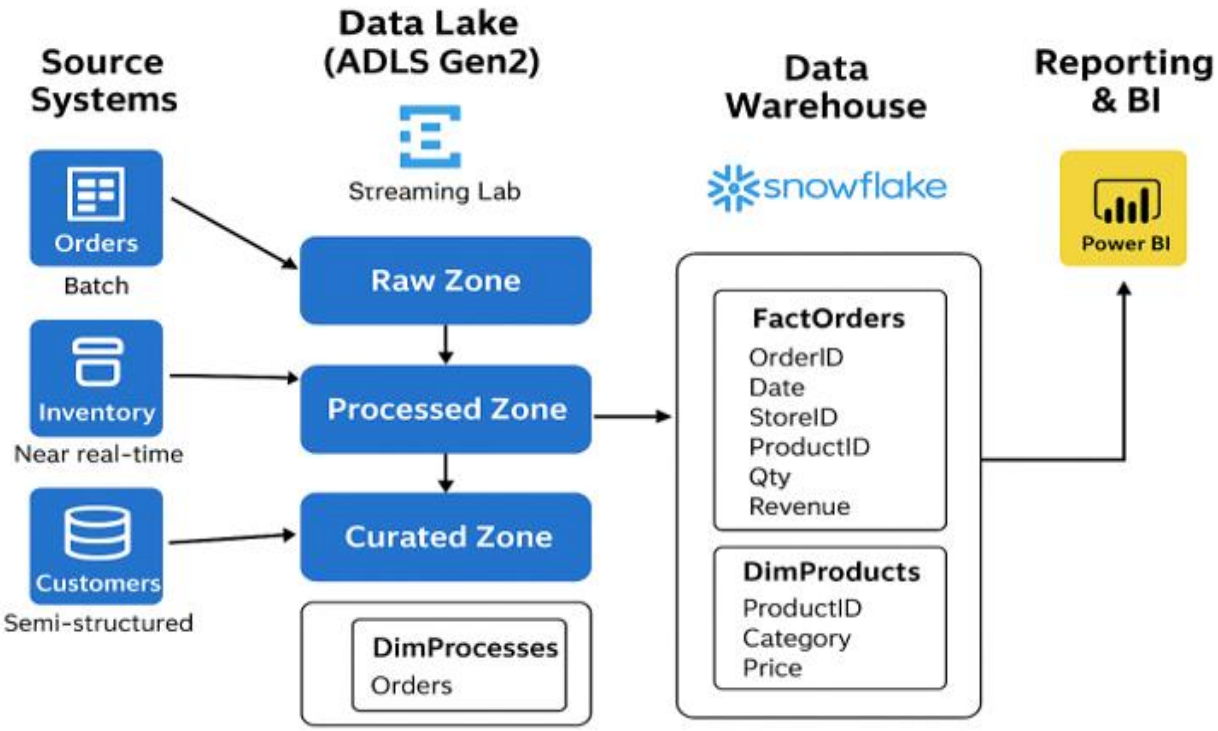
- **Step-1:** Creating a ADLS storage account to store the raw data and also the cleaned and validated data in medallion structure (Bronze, Silver, Gold).
- **Step-2:** For **BATCH-PROCESSING** create 2 ADF(Azure Data Factory) Pipelines one for each of the Batch Orders CSV and Payments CSV files.
- **Step-3:** Use Event- Grid to create trigger the pipelines when ever there is new file uploaded.
- **Step-4:** Copy the data to the bronze by using COPY activity, after clean and validate the data and store it in the Silver and Gold layer and finally load it into the Snowflake table.
- **Step-4:** Create Alerts from ADF monitor and LOG the DATABRICKS logs into **LOG ANALYTICS WORKSPACE.**
- **Step-5:** For **STREAM-PROCESSING** create a Event-hub to collect the events from the Kafka. Create Databricks Notebook and connect to the Event-hub. Collect the events from event hub and process the event and store the data in delta format in bronze, silver and gold. Finally connect the snowflake toload the data into the fact table.
- **Step-6:** Load the customer, product and store data from ADLS to the snowflake by

ADLS SAS token and creating stage in the snowflake to copy the data. Use copy into the Snowflake table.

- **Step-7:** For the customer data do the PII masking for the Email and phone number for security purpose.
- **Step-8:** Connect snowflake with Power BI for the real time analytics.

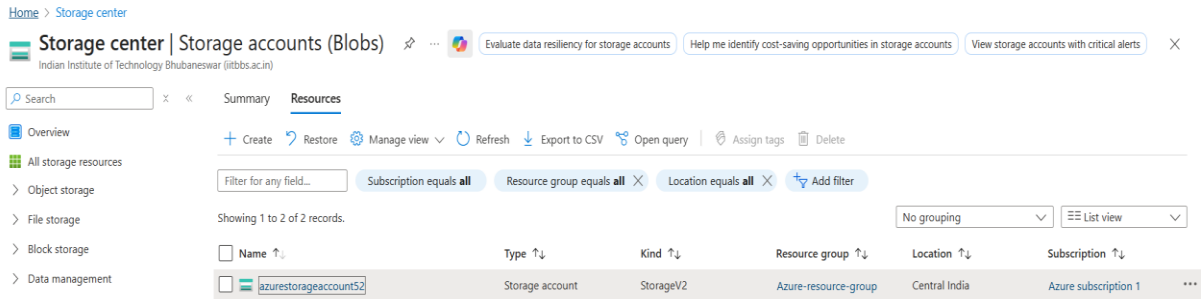
ARCHITECTURE:

I am using the below architecture

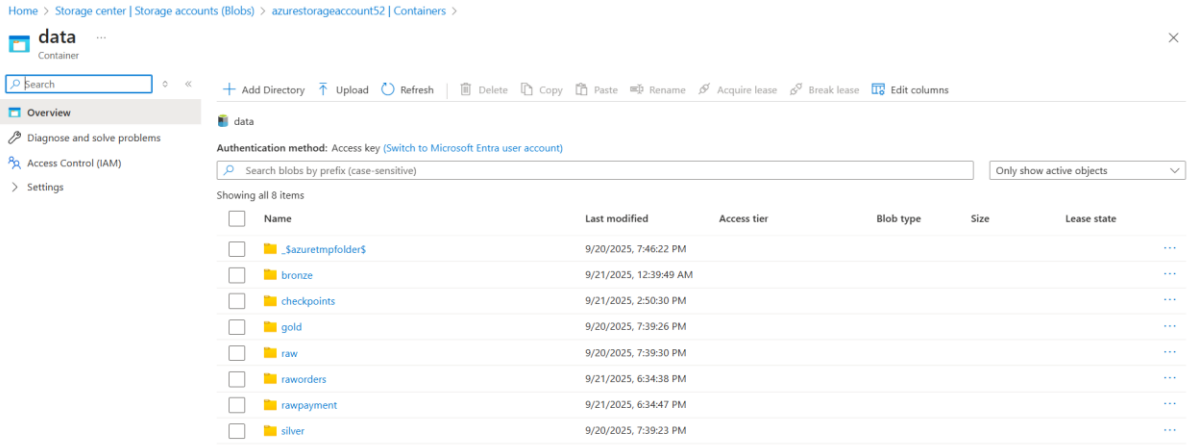


AZURE STORAGE ACCOUNT CREATION:

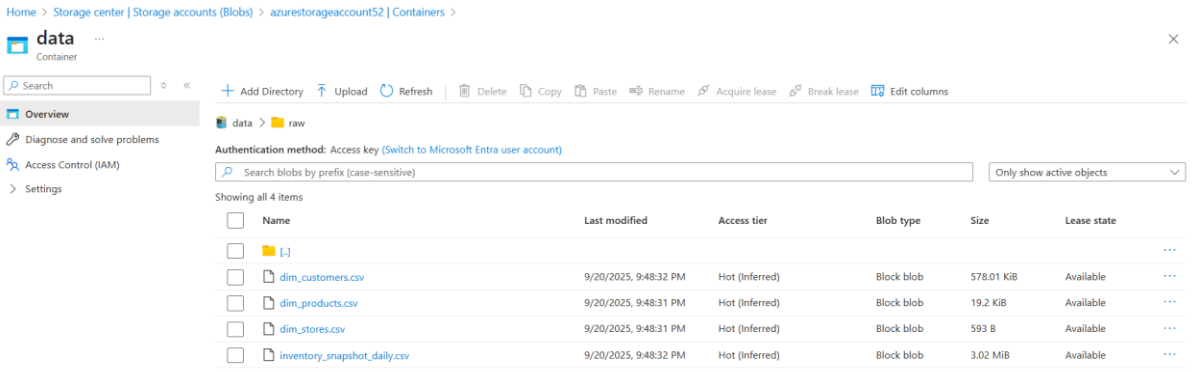
- ✓ Created an storage account “**azurestorageaccount52”.**



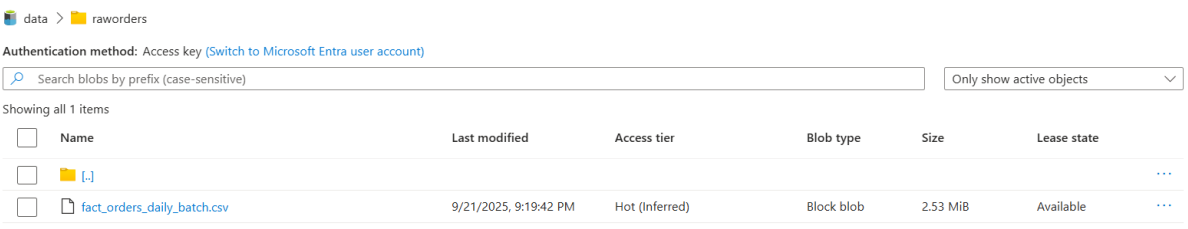
- ✓ Create a container and folders in it to store the data in medallion Architecture.

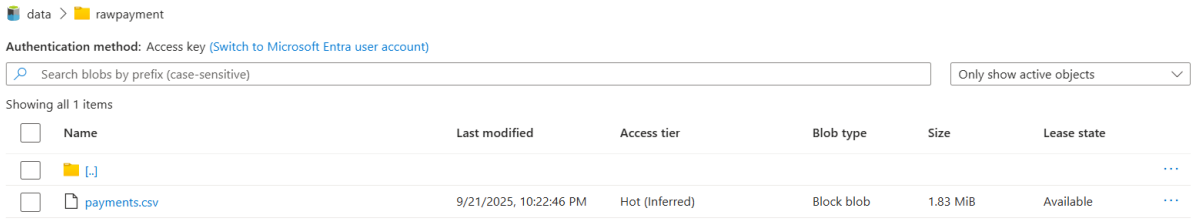


- ✓ Store the customer, products and store data in **raw directory**



- ✓ The raworders and rawpayment folders are for the storage of Batch process files automatically when there is a new file uploaded.

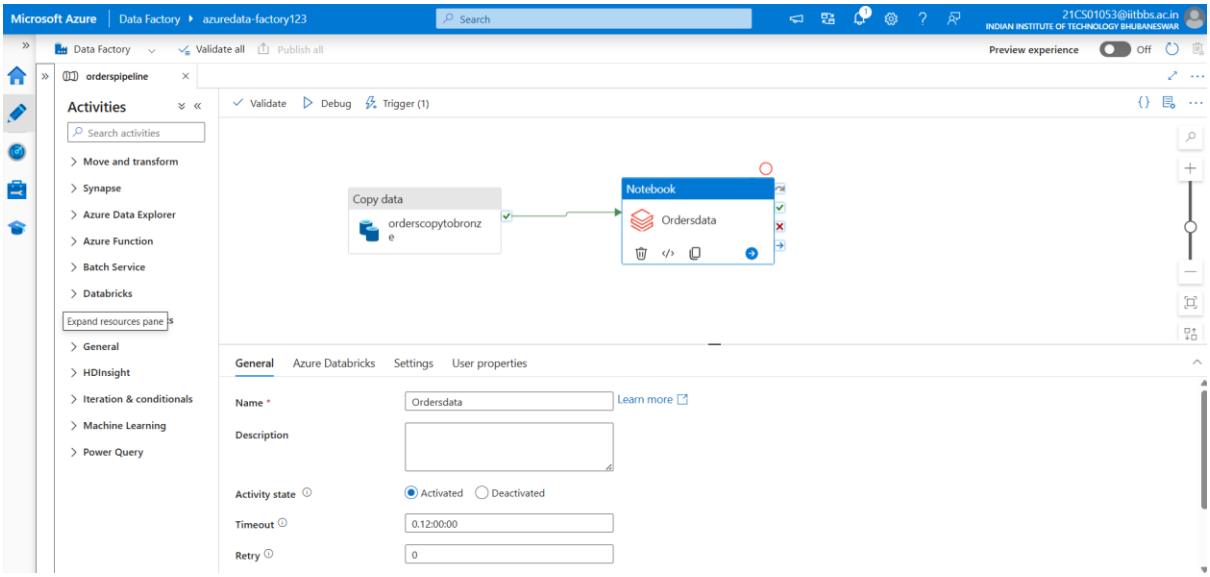




BATCH – PROCESSING:

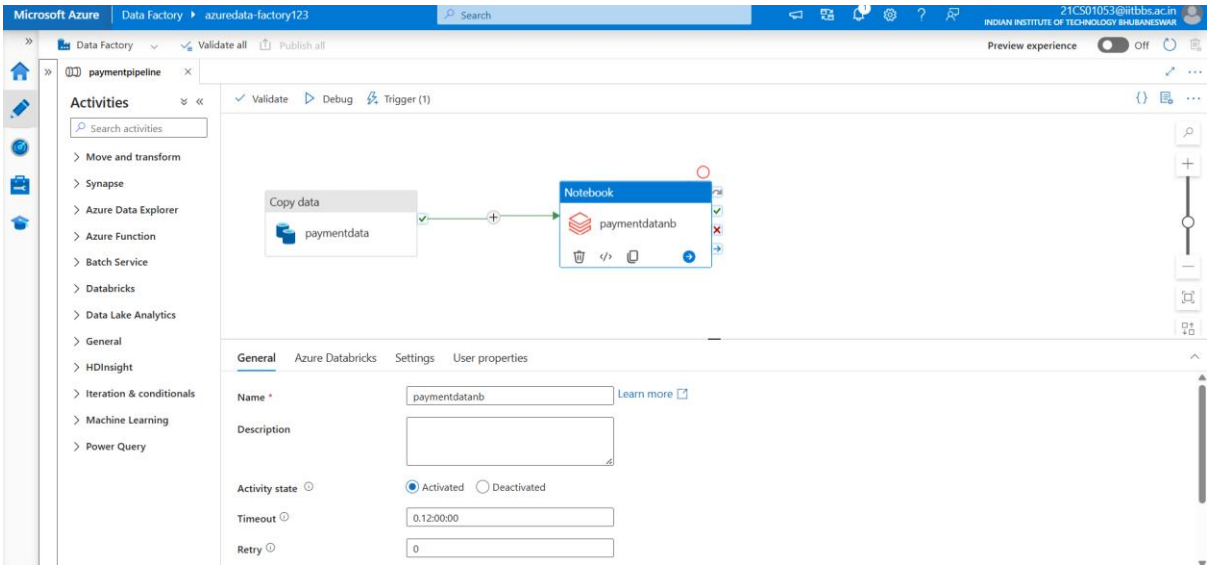
For Batch orders pipeline.

- ✓ Create a pipeline for batch orders to run when a new file is added in raworders.

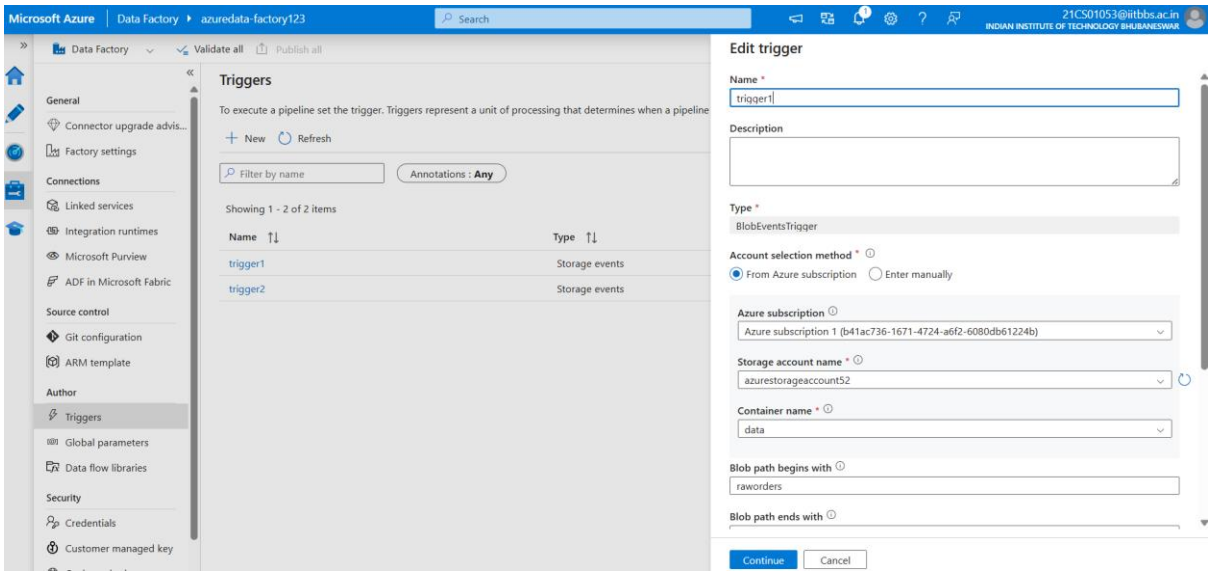


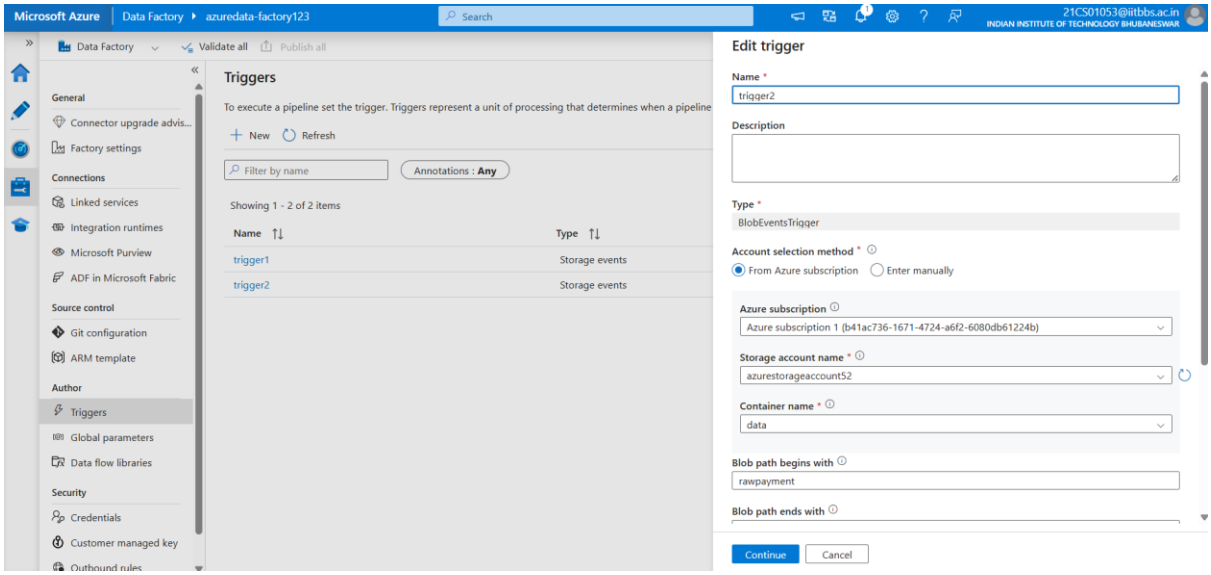
For Batch payment pipeline

- ✓ Create a pipeline for batch payment to run when a new file is added in rawpayments.

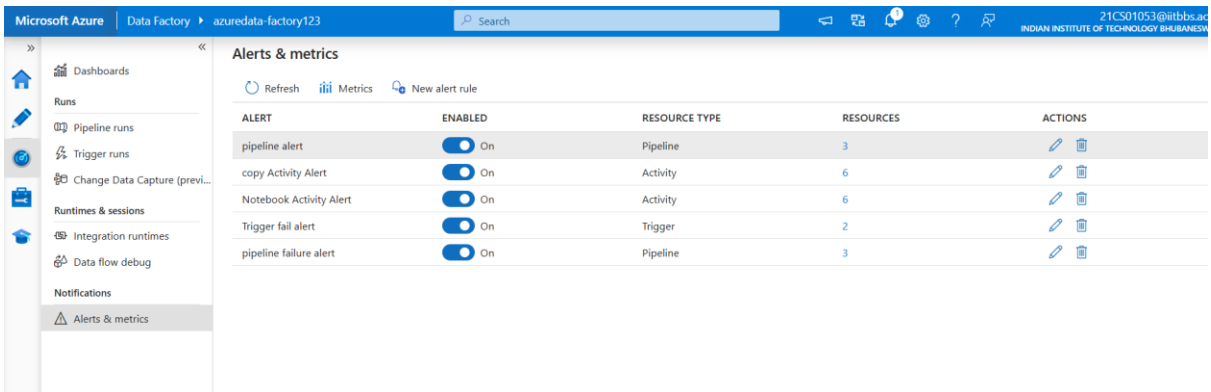


✓ Create 2 triggers to run the pipelines for new file upload.



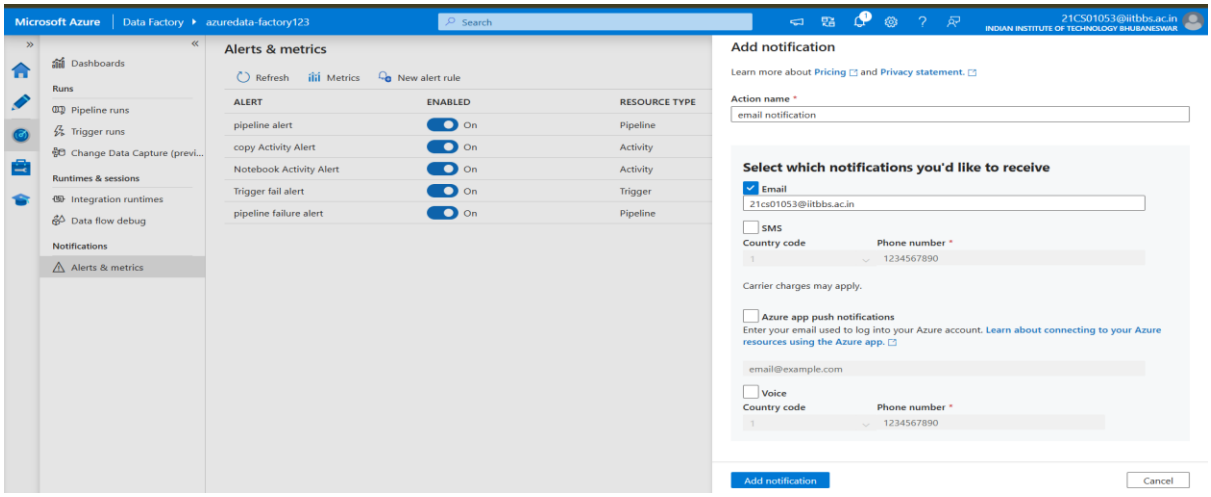


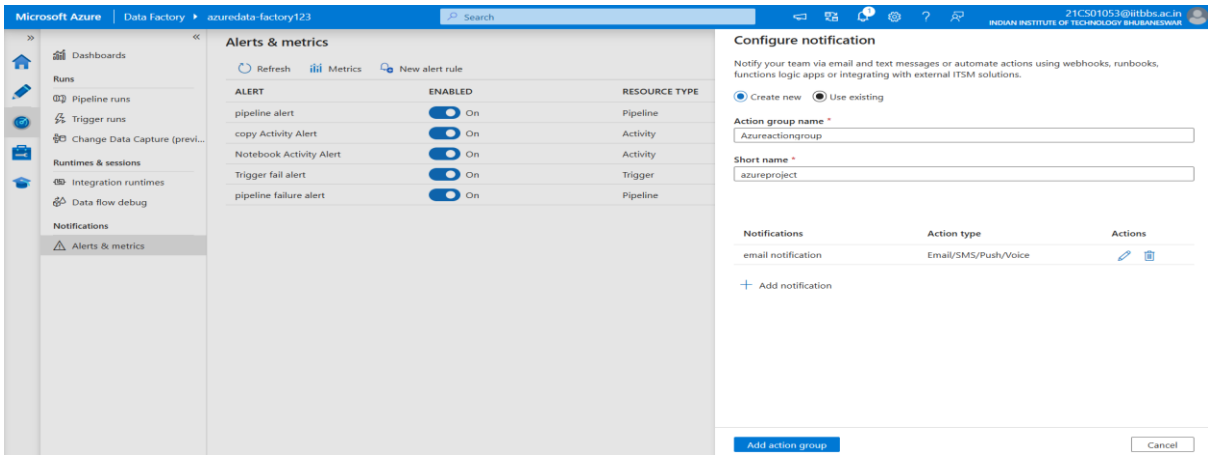
✓ Create Alerts for the pipeline failures and activity failures for both the pipelines.



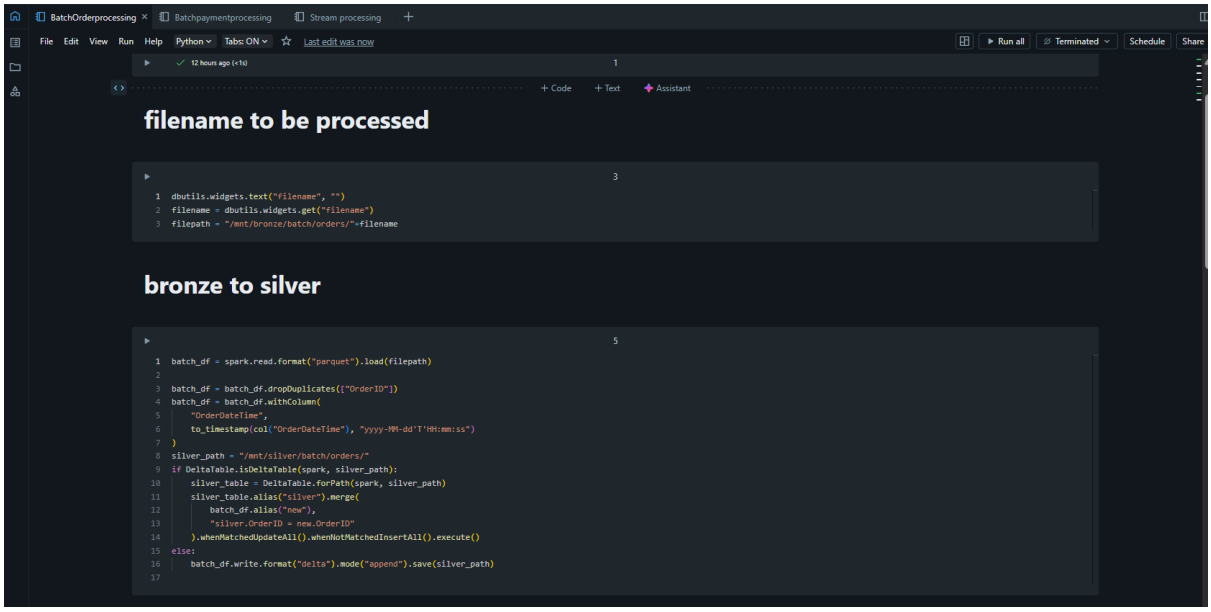
CREATE ACTION GROUP:

- ✓ Created action group for notification alerts through email.





Orders Batch processing Notebook:



silver to gold

```
7
gold_path = "/mnt/gold/batch/orders/"
if DeltaTable.isDeltaTable(spark, gold_path):
    gold_table = DeltaTable.forPath(spark, gold_path)
    gold_table.alias("gold").merge(
        batch_df.alias("new"),
        "gold.StoreID = new.StoreID AND gold.OrderDate = new.OrderDate"
    ).whenMatchedUpdateAll().whenNotMatchedInsertAll().execute()
else:
    batch_df.write.format("delta").mode("append").save(gold_path)
```

Gold to Snowflake table

```
9
pip install snowflake-snowpark-python
```

```
12 hours ago (<1s)
10
from snowflake.snowpark import Session
```

```
11
# Snowflake options
batch_df.write \
    .format("snowflake") \
    .options(
            sfURL="SVXVXKL-FR13613.snowflakecomputing.com",
            sfUser="BHUBANESWAR",
            sfPassword="Dck5n7J3Nw5Dct8",
            sfDatabase="TRAINING",
            sfSchema="PUBLIC",
            sfWarehouse="SIGMOID",
            sfRole="ACCOUNTADMIN"
        ) \
    .option("dbtable", "FACTORDERS") \
    .mode("append") \
    .save()
```

[Shift+Enter] to run and move to next cell
[Ctrl+Shift+P] to open the command palette
[Esc H] to see all keyboard shortcuts

OUTPUT TABLE (FACT ORDER):

Table | Chart — 30,000 rows — 117ms

	ORDERID	ORDERDATETIME	STOREID	CUSTOMERID	PRODUCTID	QUANTITY	UNITPRICE	DISCOUNTPCT	PAYMENTID	PAYMENTMETHOD	CHANNEL	STATUS	ORDERTOTAL	CURRENCY
1	296	2025-05-07 14:27:00.000	9	4612	90	1	59.66	0.00	PMT00000296	CARD	Online	Completed	59.66	INR
2	467	2025-07-18 05:57:00.000	6	3198	179	1	76.01	0.00	PMT00000467	UPI	Online	Completed	76.01	INR
3	675	2025-04-06 06:38:00.000	2	3965	10	1	22.57	5.00	PMT00000675	CASH	POS	Completed	21.44	INR
4	691	2025-08-14 05:17:00.000	8	4384	164	1	14.05	0.00	PMT00000691	UPI	POS	Cancelled	14.05	INR
5	829	2025-04-30 09:48:00.000	2	5610	129	1	18.64	5.00	PMT00000829	WALLET	Online	Completed	17.71	INR
6	1090	2025-05-02 05:08:00.000	9	3574	114	1	57.26	0.00	PMT00001090	CARD	Online	Completed	57.26	INR
7	1159	2025-05-17 00:52:00.000	7	5319	129	1	18.64	5.00	PMT00001159	WALLET	Online	Completed	17.71	INR
8	1436	2025-05-23 09:32:00.000	8	50	74	1	58.51	0.00	PMT00001436	CARD	POS	Completed	58.51	INR
9	1512	2025-07-14 11:27:00.000	12	2359	199	2	36.63	0.00	PMT00001512	CARD	POS	Pending	73.26	INR
10	1572	2025-07-11 17:28:00.000	1	5924	161	1	77.31	10.00	PMT00001572	WALLET	Online	Completed	69.58	INR
11	2069	2025-06-23 02:01:00.000	3	40	172	1	28.15	10.00	PMT00002069	CARD	Online	Completed	25.34	INR
12	2088	2025-07-05 14:43:00.000	5	2849	145	1	65.29	20.00	PMT00002088	UPI	Online	Completed	52.23	INR
13	2136	2025-04-12 07:37:00.000	1	126	225	1	39.69	0.00	PMT00002136	UPI	POS	Completed	39.69	INR
14	2162	2025-06-27 08:38:00.000	8	4156	161	1	77.31	10.00	PMT00002162	CARD	POS	Completed	69.58	INR
15	2294	2025-05-18 22:40:00.000	5	1355	233	3	13.78	5.00	PMT00002294	UPI	POS	Cancelled	39.27	INR
16	2904	2025-04-11 20:15:00.000	7	4403	80	1	99.68	0.00	PMT00002904	CASH	POS	Completed	99.68	INR
17	3210	2025-06-18 09:13:00.000	1	4239	249	1	72.05	0.00	PMT00003210	UPI	POS	Completed	72.05	INR
18	3414	2025-05-01 08:02:00.000	6	697	192	1	47.36	0.00	PMT00003414	CARD	POS	Pending	47.36	INR
19	3606	2025-06-09 18:48:00.000	2	4058	123	1	52.93	10.00	PMT00003606	WALLET	POS	Completed	47.64	INR
20	3959	2025-08-09 01:43:00.000	10	1769	87	3	54.84	0.00	PMT00003959	CARD	POS	Completed	164.52	INR
21	4032	2025-04-01 13:29:00.000	2	4671	176	1	26.16	0.00	PMT00004032	CARD	Online	Completed	26.16	INR
22	4821	2025-07-06 12:31:00.000	11	389	163	1	95.91	5.00	PMT00004821	UPI	POS	Pending	91.11	INR
23	4937	2025-06-04 16:31:00.000	8	5201	56	1	65.47	0.00	PMT00004937	CASH	POS	Completed	65.47	INR
24	5325	2025-07-20 12:41:00.000	11	5431	4	1	19.23	10.00	PMT00005325	CARD	POS	Completed	17.31	INR
25	5645	2025-06-02 04:13:00.000	8	2074	31	1	33.53	0.00	PMT00005645	CARD	POS	Completed	33.53	INR
26	5925	2025-08-02 03:49:00.000	7	1767	122	1	22.69	15.00	PMT00005925	CARD	POS	Completed	19.29	INR
27	6194	2025-08-07 10:18:00.000	11	2373	52	1	108.43	0.00	PMT00006194	UPI	POS	Completed	108.43	INR

Payments Batch processing Notebook:

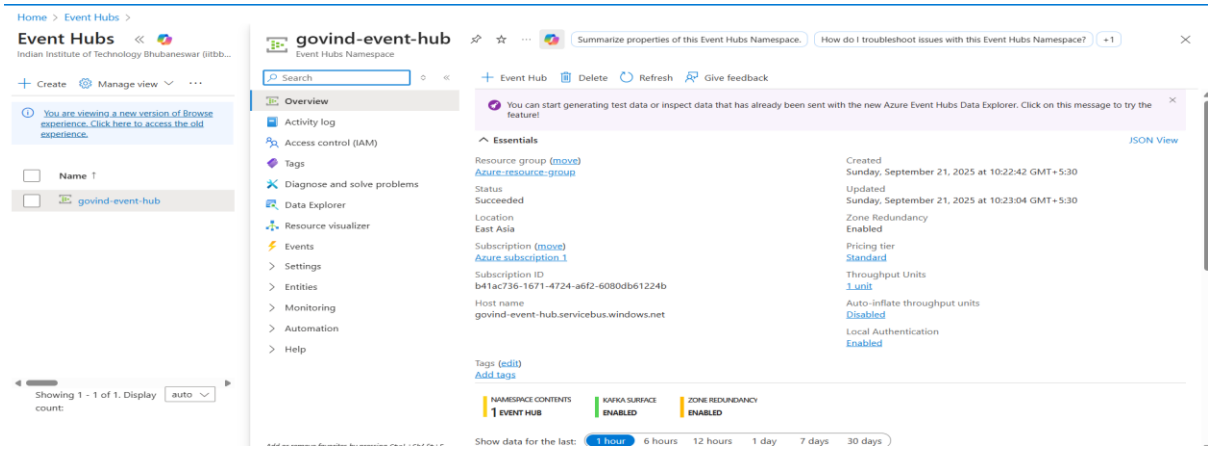
OUTPUT TABLE (PAYMENTS):

Table | Chart — 30,000 rows — 89ms

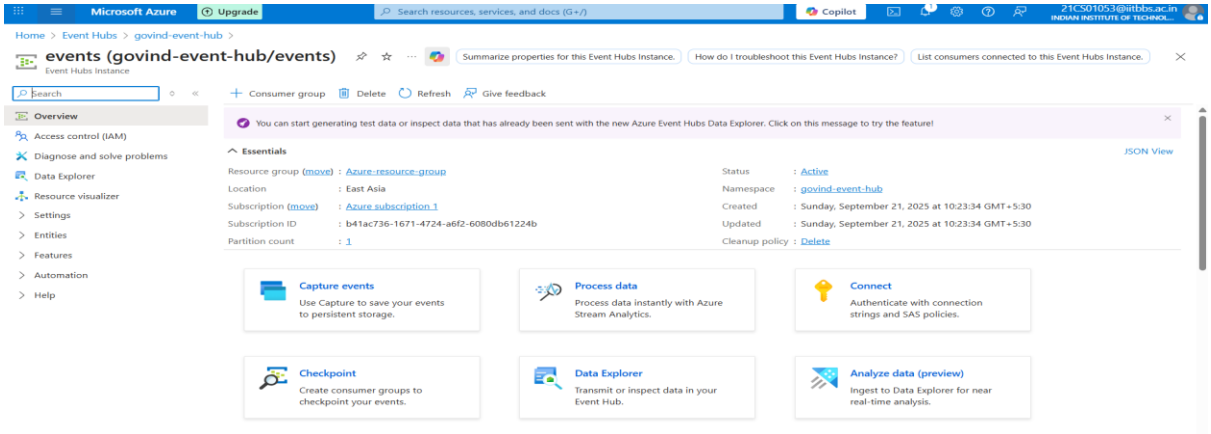
	PAYMENTID	ORDERID	AMOUNT	CURRENCY	METHOD	GATEWAY	STATUS	AUTHCODE	FRAUDSCORE
1	PMT00000122	122	76.65	INR	UPI	Razorpay	Pending	F8BV5ZHP	0.041
2	PMT00000638	638	344.82	INR	UPI	PayU	Captured	MRVNZQER	0.032
3	PMT00000960	960	89.16	INR	UPI	Stripe	Captured	MPEOSO1F	0.167
4	PMT00001057	1057	94.66	INR	CARD	Cashdesk	Captured	YNLAIMA5	0.105
5	PMT00001141	1141	61.40	INR	CARD	Razorpay	Captured	19FG4UBV	0.100
6	PMT00001648	1648	63.62	INR	CASH	PayU	Captured	E2K3RUUW	0.051
7	PMT00001911	1911	312.84	INR	CASH	PayU	Captured	TXYEP6C1	0.352
8	PMT00002258	2258	130.96	INR	CARD	Stripe	Captured	J9PJPICR	0.033
9	PMT00002695	2695	25.95	INR	CARD	Cashdesk	Captured	ZEX85CDI	0.152
10	PMT00002949	2949	41.89	INR	UPI	PayU	Captured	YVDPT6PK	0.024
11	PMT00002980	2980	77.78	INR	CARD	Cashdesk	Pending	2W7YRB21	0.167
12	PMT00003908	3908	52.41	INR	UPI	Cashdesk	Captured	X7WMQAY4	0.104
13	PMT00003912	3912	13.32	INR	CARD	Cashdesk	Captured	8A90UUFX	0.171
14	PMT00004493	4493	25.42	INR	UPI	Stripe	Captured	FNEJWGYE	0.132
15	PMT00004494	4494	58.51	INR	CARD	Razorpay	Captured	3W5QUJQ9	0.180
16	PMT00004928	4928	823.41	INR	WALLET	Cashdesk	Captured	FF2XZDT9	0.097

STREAM– PROCESSING:

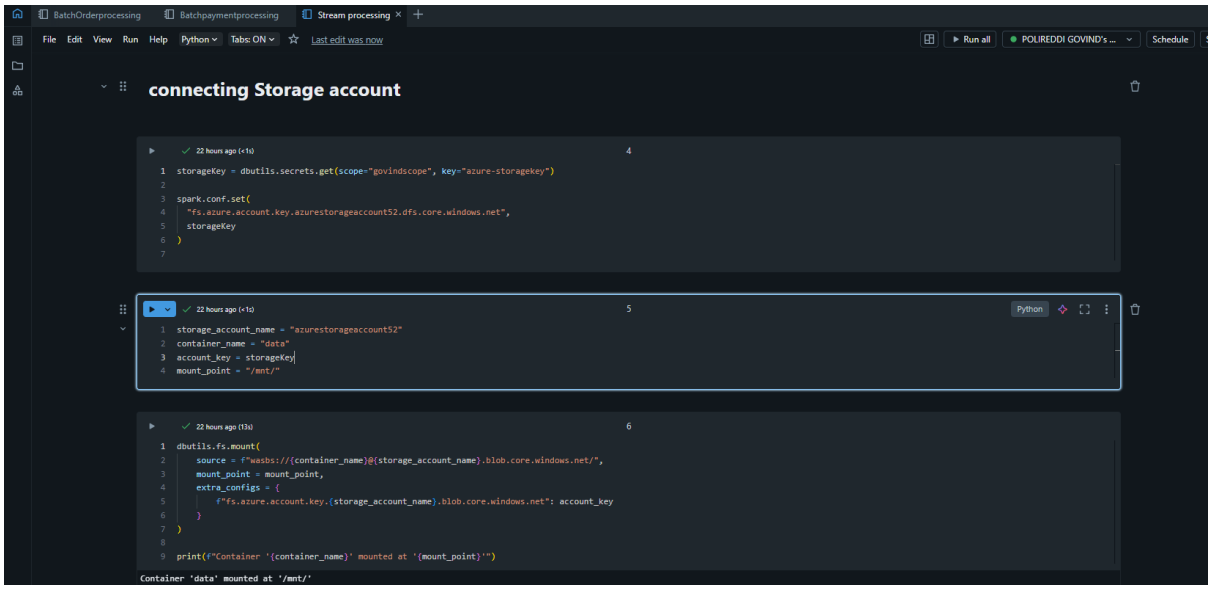
- ✓ Create an Event hub to collect events and send to the databricks notebook



- ✓ Create a new event in the event hub namespace “events”



- ✓ Create a notebook and make connection with storage account.
- ✓ **USE THE AZURE KEY – VAULT to STORE THE SECRETS.**



✓ Connect to the event hub

connecting to Azure event-hub

```
connectionstring = dbutils.secrets.get(scope="govindscope", key="azure-eventhub-connectionstring")
event_hub_config = {
    "kafka.bootstrap.servers": "govind-event-hub.servicebus.windows.net:9093",
    "subscribe": "events",
    "startingOffsets": "latest",
    "kafka.security.protocol": "SASL_SSL",
    "kafka.sasl.mechanism": "PLAIN",
    "kafka.sasl.jaas.config": f'{connectionstring};'
}

df_kafka = spark.readStream.format("kafka").options(**event_hub_config).load()
```

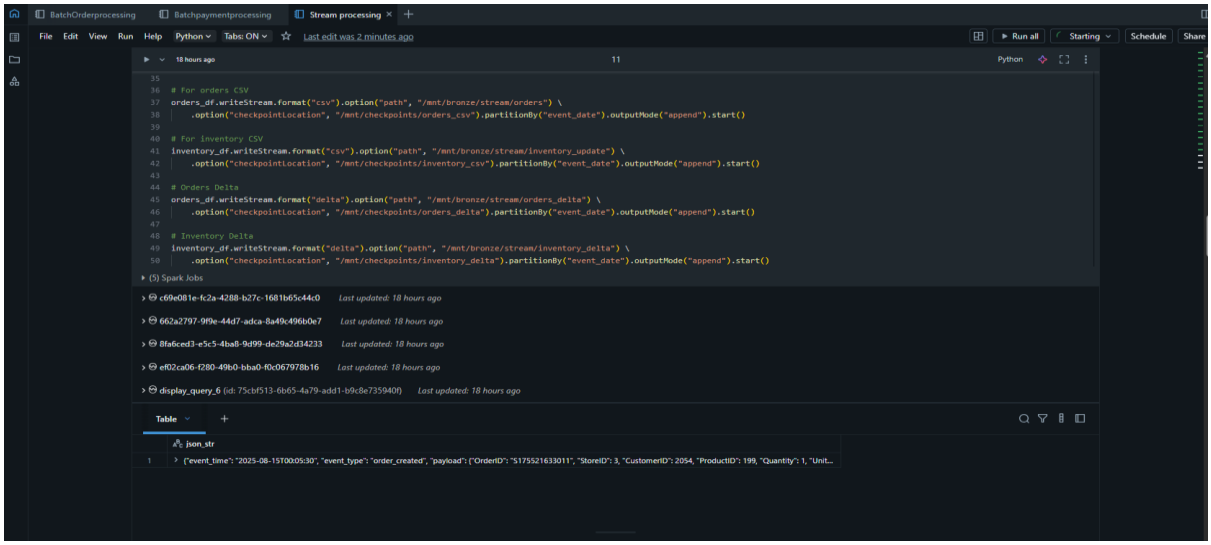
✓ Schema define for the data

```
order_schema = StructType([
    StructField("event_time", TimestampType()),
    StructField("event_type", StringType()),
    StructField("payload", StructType([
        StructField("OrderID", StringType()),
        StructField("StoreID", IntegerType()),
        StructField("CustomerID", IntegerType()),
        StructField("ProductID", IntegerType()),
        StructField("Quantity", IntegerType()),
        StructField("UnitPrice", DoubleType()),
        StructField("DiscountPct", DoubleType()),
        StructField("Channel", StringType())
    ]))
])

inventory_schema = StructType([
    StructField("event_time", TimestampType()),
    StructField("event_type", StringType()),
    StructField("payload", StructType([
        StructField("StoreID", IntegerType()),
        StructField("ProductID", IntegerType()),
        StructField("DeltaQty", IntegerType())
    ]))
])
```

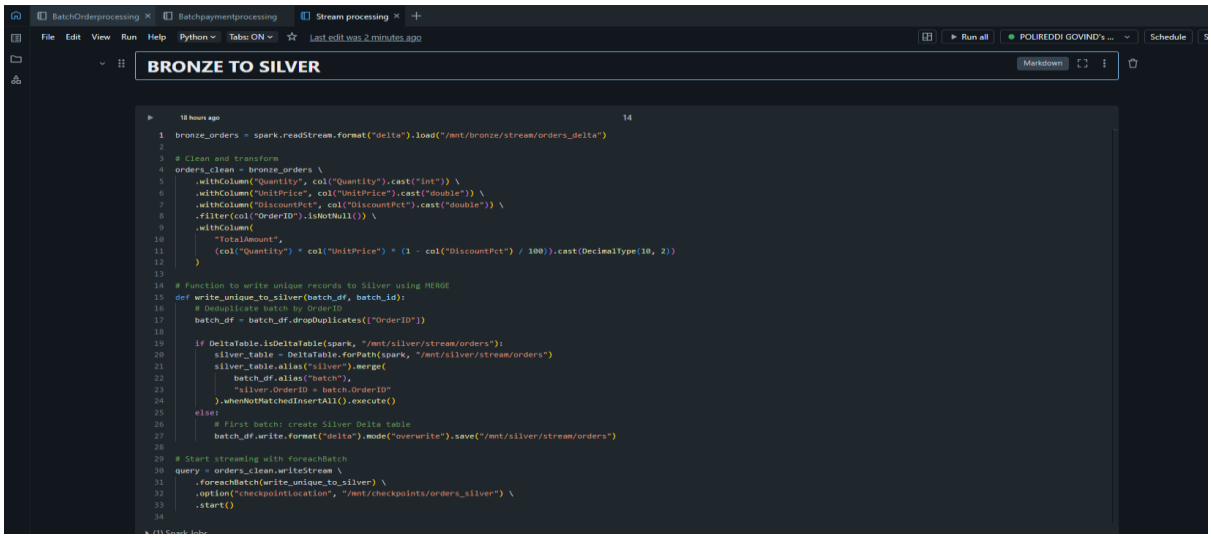
✓ Loading data to the bronze layer which are received from the event hub events.
✓ Separate the events as per the orders and Inventory data

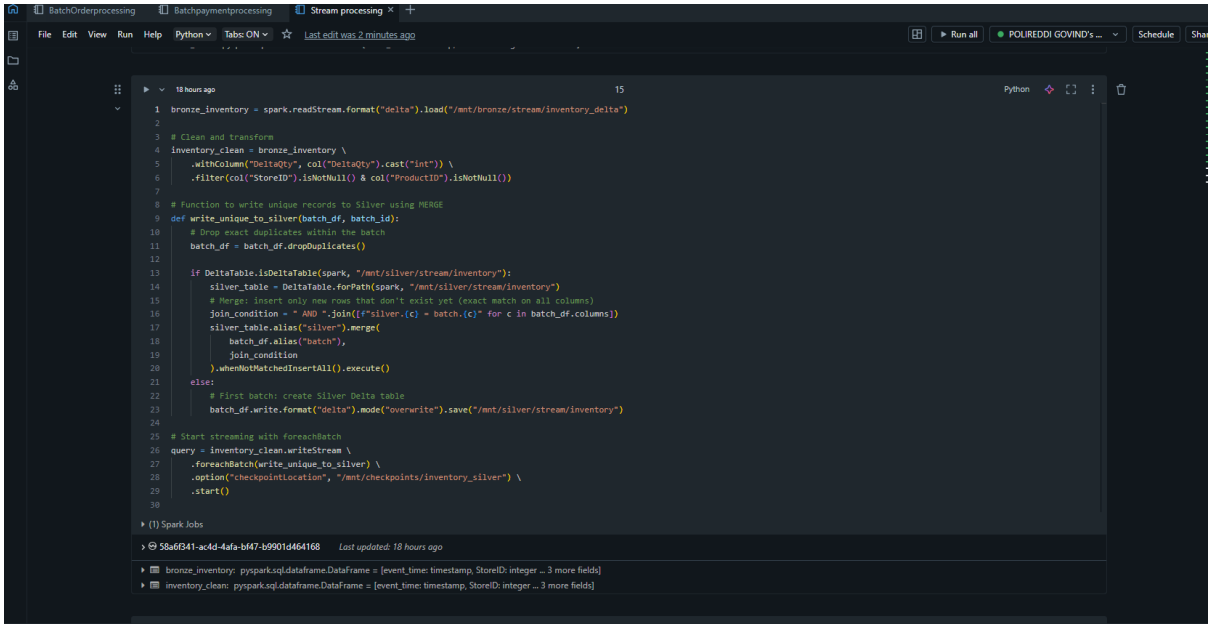
loading events from event-hub to bronze layer

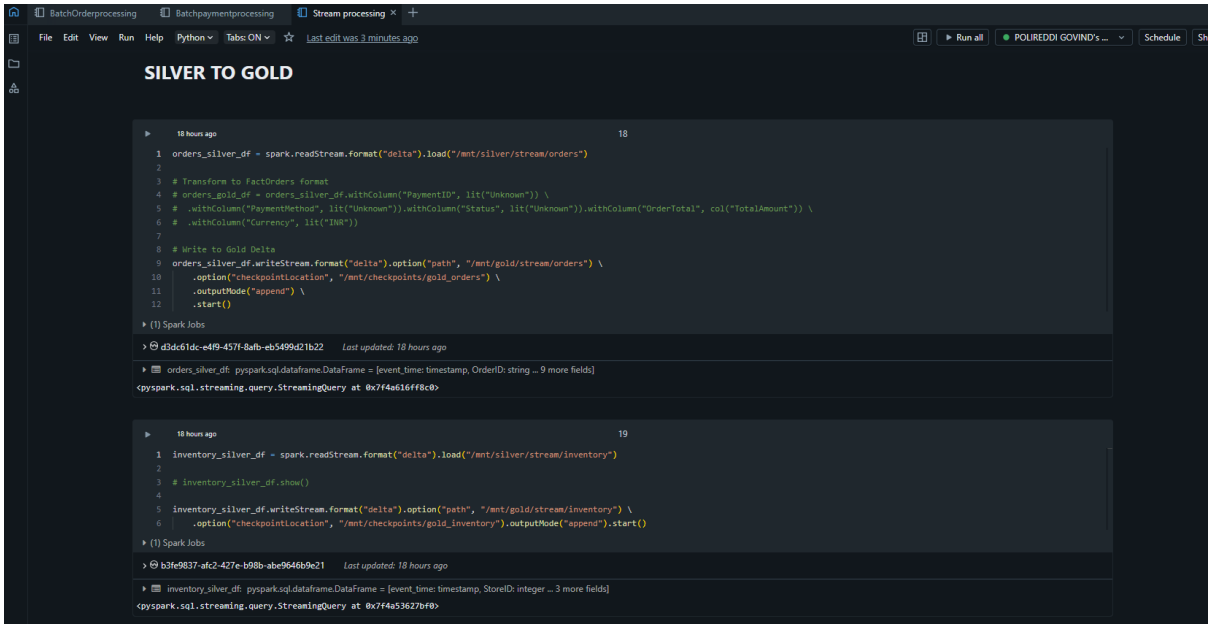
```
df_kafka = spark.readStream.format("kafka").options(**event_hub_config).load()

df_kafka_str = df_kafka.selectExpr("CAST(value AS STRING) as json_str")

df_kafka_str.display()

# Orders stream
orders_df = df_kafka_str.select(from_json(col("json_str"), order_schema).alias("data")) \
    .filter(col("data.event_type") == "order_created").select(
        col("data.event_time").alias("event_time"),
        col("data.payload.OrderID").alias("OrderID"),
        col("data.payload.StoreID").alias("StoreID"),
        col("data.payload.CustomerID").alias("CustomerID"),
        col("data.payload.ProductID").alias("ProductID"),
        col("data.payload.Quantity").alias("Quantity"),
        col("data.payload.UnitPrice").alias("UnitPrice"),
        col("data.payload.DiscountPct").alias("DiscountPct"),
        col("data.payload.Channel").alias("Channel"),
        to_date(col("data.event_time")).alias("event_date")
    )

# Inventory stream
inventory_df = df_kafka_str \
    .select(from_json(col("json_str"), inventory_schema).alias("data")) \
    .filter(col("data.event_type") == "inventory_update") \
    .select(
        col("data.event_time").alias("event_time"),
        col("data.payload.StoreID").alias("StoreID"),
        col("data.payload.ProductID").alias("ProductID"),
        col("data.payload.DeltaQty").alias("DeltaQty"),
        to_date(col("data.event_time")).alias("event_date")
    )
```



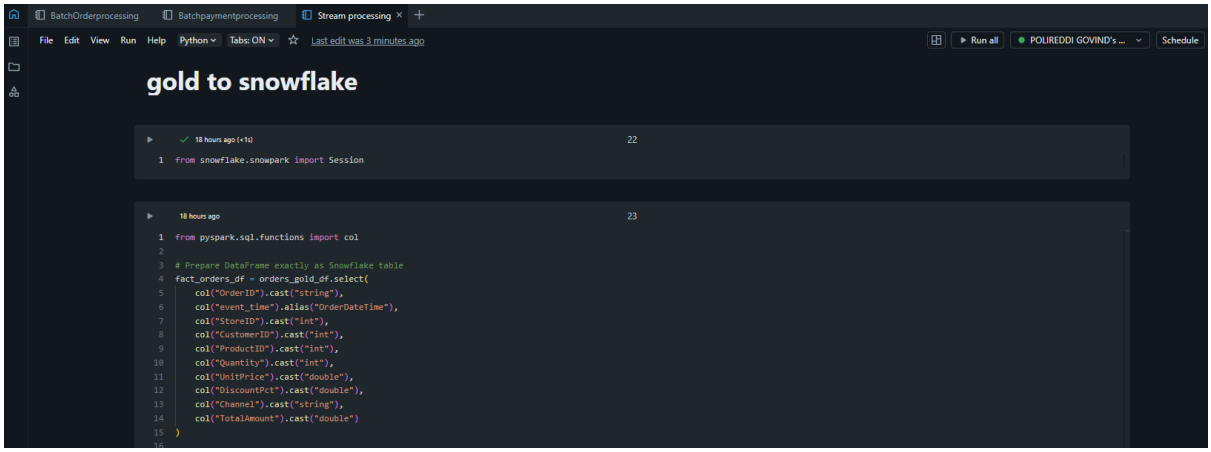
✓ Broze to silver after cleaning and validating the data

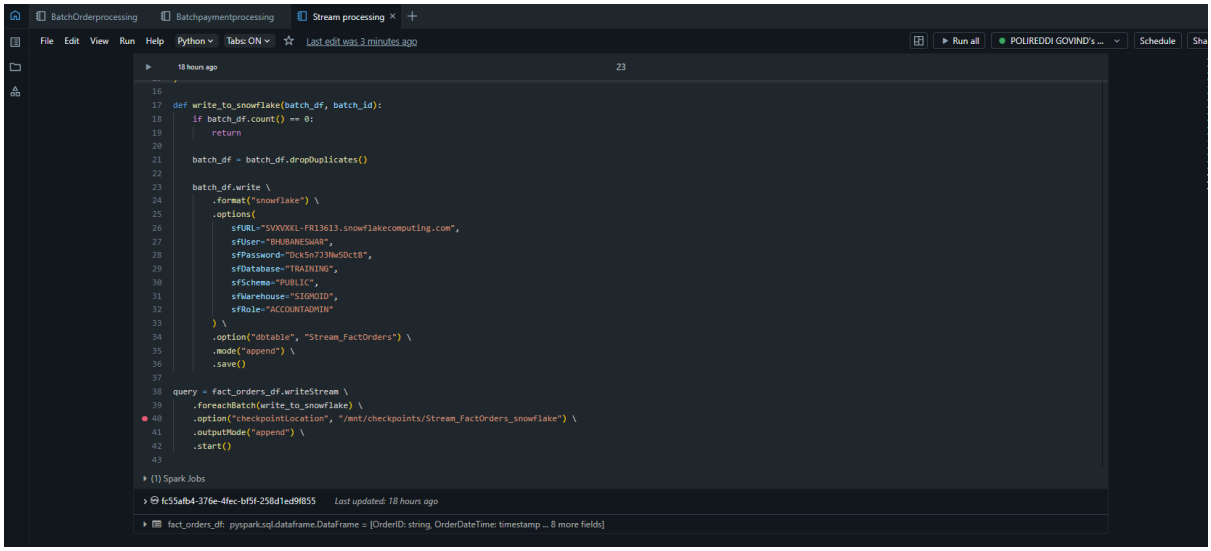






✓ Gold to snowflake table



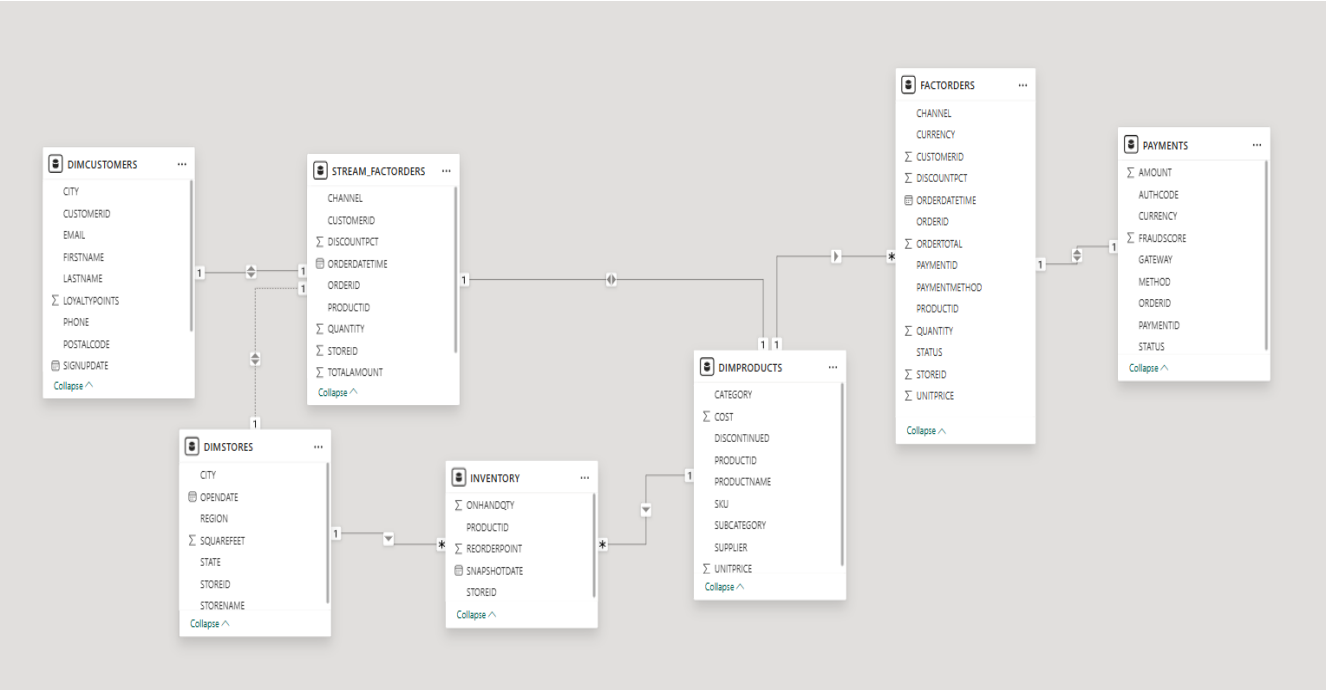


✓ **INVENTORY UPDATE TO THE SNOWFLAKE INVENTORY TABLE**

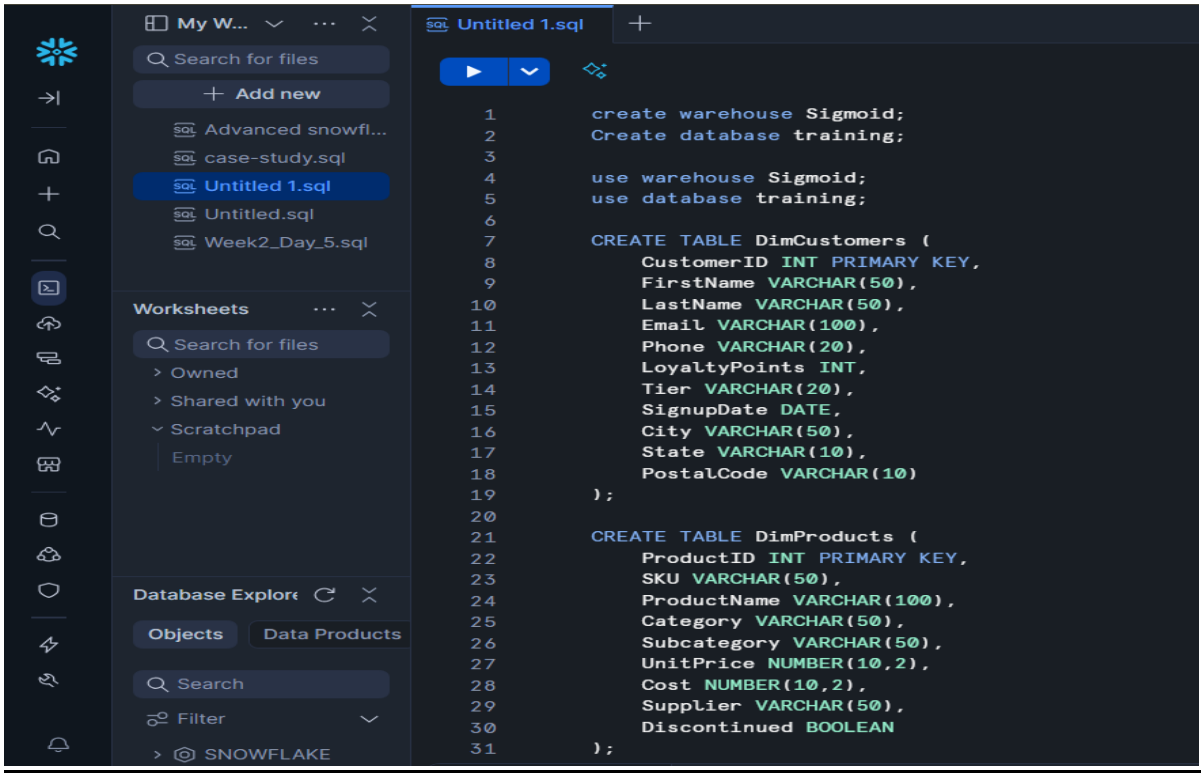
```python
from pyspark.sql.functions import col
import snowflake.connector

inventory_gold = spark.readStream.format("delta").load("/mnt/gold/stream/inventory")

# Upsert function
def upsert_inventory_snowflake(batch_df, batch_id):
    if batch_df.count() == 0:
        return
    batch_df = batch_df.dropDuplicates()
    conn_params = {
        "user": "BHUBANESWAR",
        "password": "Dck5n7J3Nw5Dct8",
        "account": "SVXVXKL-FR13613",
        "warehouse": "SIGMOID",
        "database": "TRAINING",
        "schema": "PUBLIC",
        "role": "ACCOUNTADMIN"
    }
    conn = snowflake.connector.connect(**conn_params)
    cs = conn.cursor()
    # Generate and execute MERGE for each row
    for row in batch_df.collect():
        merge_sql = f"""
        MERGE INTO INVENTORY target
        USING (SELECT {row.StoreID} AS StoreID, {row.ProductID} AS ProductID, {row.DeltaQty} AS OnHandQty) source
        ON target.StoreID = source.StoreID AND target.ProductID = source.ProductID
        WHEN MATCHED THEN UPDATE SET target.OnHandQty = target.OnHandQty + source.OnHandQty
        WHEN NOT MATCHED THEN INSERT (StoreID, ProductID, OnHandQty)
        VALUES (source.StoreID, source.ProductID, source.OnHandQty)
        """
        cs.execute(merge_sql)

    cs.close()
    conn.close()

# Start the streaming job
query = inventory_gold.writeStream \
    .foreachBatch(upsert_inventory_snowflake) \
    .option("checkpointLocation", "/mnt/checkpoints/inventory_snowflake") \
    .start()
```

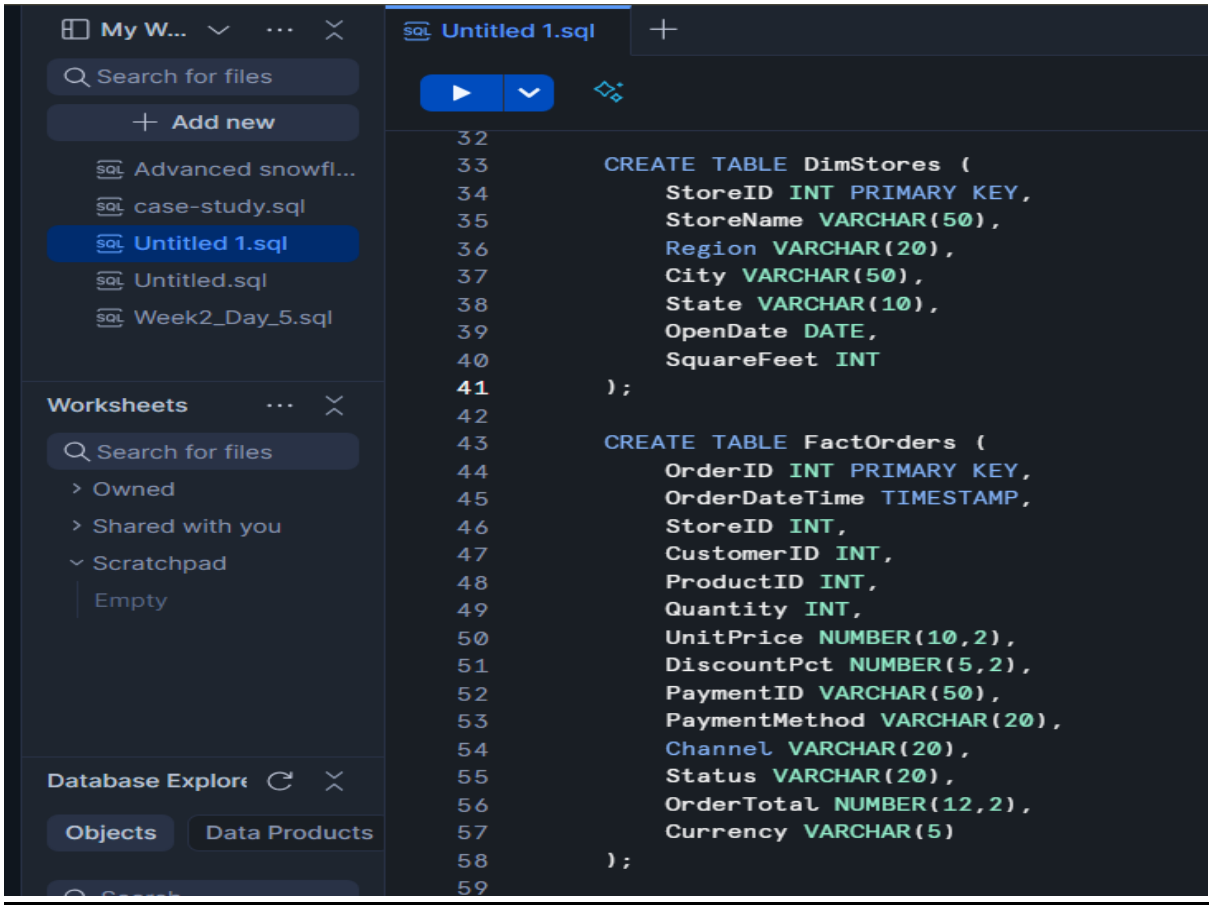
SNOWFLAKE WAREHOUSE DATA TABLE:

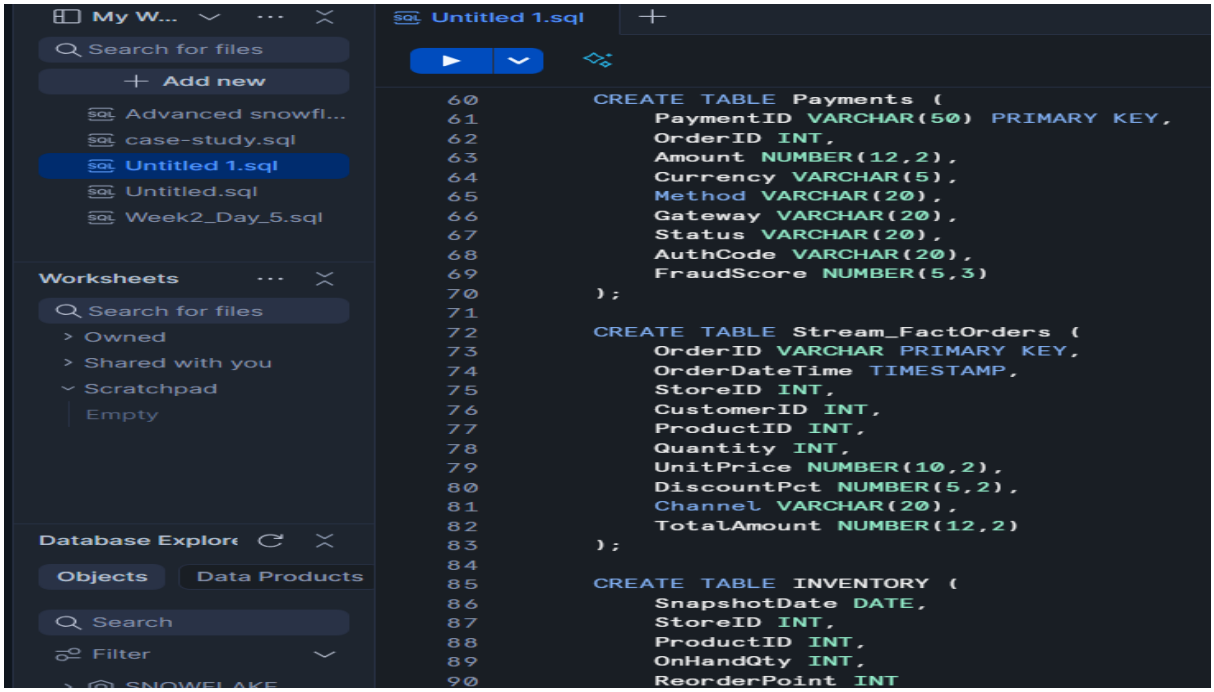
✓ E - R DIAGRAM:



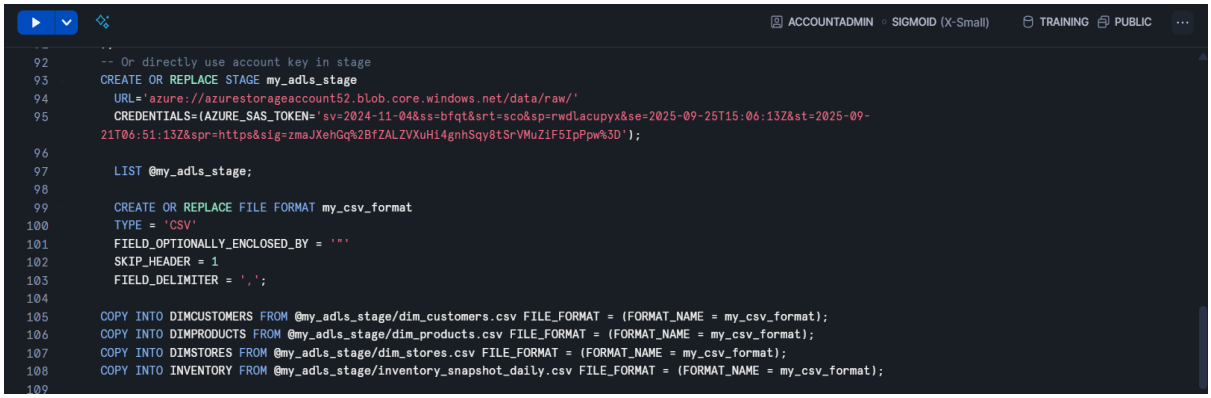
✓ Snowflake tables







- ✓ Loading data from ADLS to a stage in Snowflake and copy into the tables

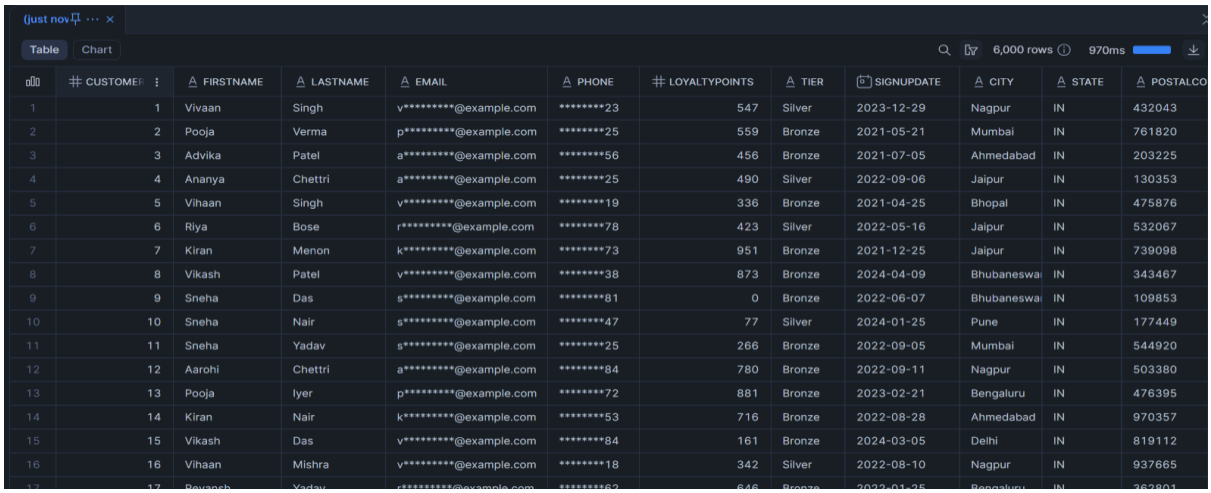


PII MASKING:

- ✓ For the customer table mask the email and phone number

```
143
144
145     CREATE OR REPLACE MASKING POLICY phone_partial AS (val STRING)
146     RETURNS STRING ->
147     CASE
148         WHEN CURRENT_ROLE() IN ('FULL_ACCESS_ROLE') THEN val
149         ELSE CONCAT('********', RIGHT(val,2))
150     END;
151
152
153     ALTER TABLE DimCustomers
154     MODIFY COLUMN Email
155     SET MASKING POLICY pii_mask;
156
157     ALTER TABLE DimCustomers
158     MODIFY COLUMN Phone
159     SET MASKING POLICY phone_partial;
160
```

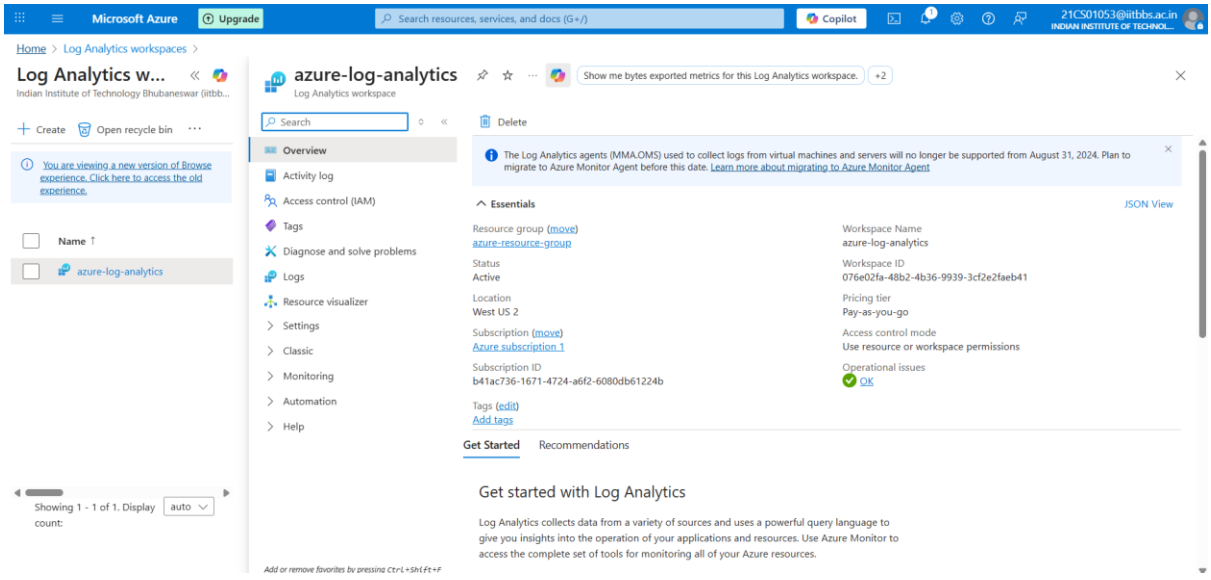
Customer Table:



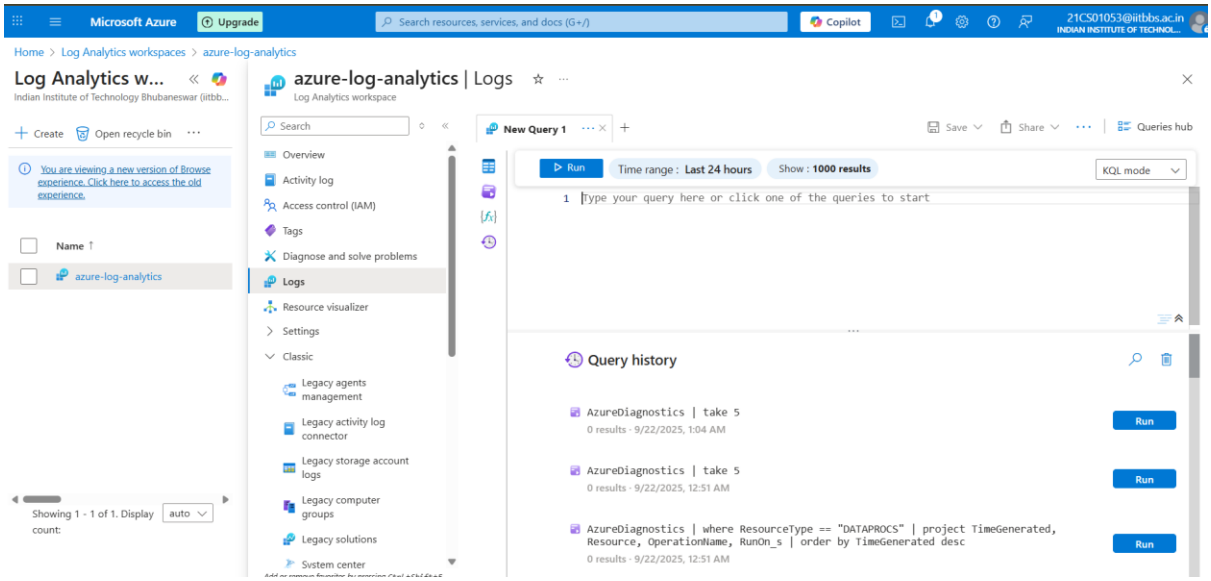
	CUSTOMER	FIRSTNAME	LASTNAME	EMAIL	PHONE	LOYALTYPOINTS	TIER	SIGNUPDATE	CITY	STATE	POSTALCO
1	1	Vivaan	Singh	v*********@example.com	********23	547	Silver	2023-12-29	Nagpur	IN	432043
2	2	Pooja	Verma	p*********@example.com	********25	559	Bronze	2021-05-21	Mumbai	IN	761820
3	3	Advika	Patel	a*********@example.com	********56	456	Bronze	2021-07-05	Ahmedabad	IN	203225
4	4	Ananya	Chettri	a*********@example.com	********25	490	Silver	2022-09-06	Jaipur	IN	130353
5	5	Vihaan	Singh	v*********@example.com	********19	336	Bronze	2021-04-25	Bhopal	IN	475876
6	6	Riya	Bose	r*********@example.com	********78	423	Silver	2022-05-16	Jaipur	IN	532067
7	7	Kiran	Menon	k*********@example.com	********73	951	Bronze	2021-12-25	Jaipur	IN	739098
8	8	Vikash	Patel	v*********@example.com	********38	873	Bronze	2024-04-09	Bhubaneswa	IN	343467
9	9	Sneha	Das	s*********@example.com	********81	0	Bronze	2022-06-07	Bhubaneswa	IN	109853
10	10	Sneha	Nair	s*********@example.com	********47	77	Silver	2024-01-25	Pune	IN	177449
11	11	Sneha	Yadav	s*********@example.com	********25	266	Bronze	2022-09-05	Mumbai	IN	544920
12	12	Aarohi	Chettri	a*********@example.com	********84	780	Bronze	2022-09-11	Nagpur	IN	503380
13	13	Pooja	Iyer	p*********@example.com	********72	881	Bronze	2023-02-21	Bengaluru	IN	476395
14	14	Kiran	Nair	k*********@example.com	********53	716	Bronze	2022-08-28	Ahmedabad	IN	970357
15	15	Vikash	Das	v*********@example.com	********84	161	Bronze	2024-03-05	Delhi	IN	819112
16	16	Vihaan	Mishra	v*********@example.com	********18	342	Silver	2022-08-10	Nagpur	IN	937665

MONITORING :

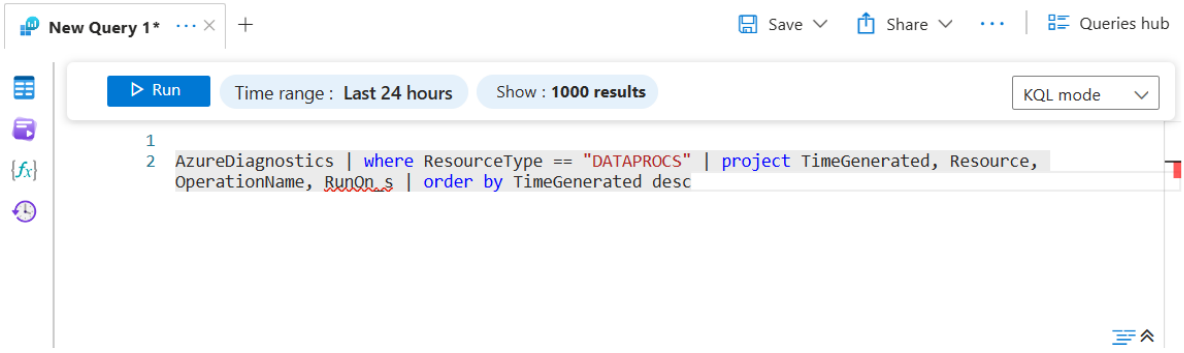
- ✓ Create a log analytics workspace



- ✓ Open LOGS and then use KQL mode

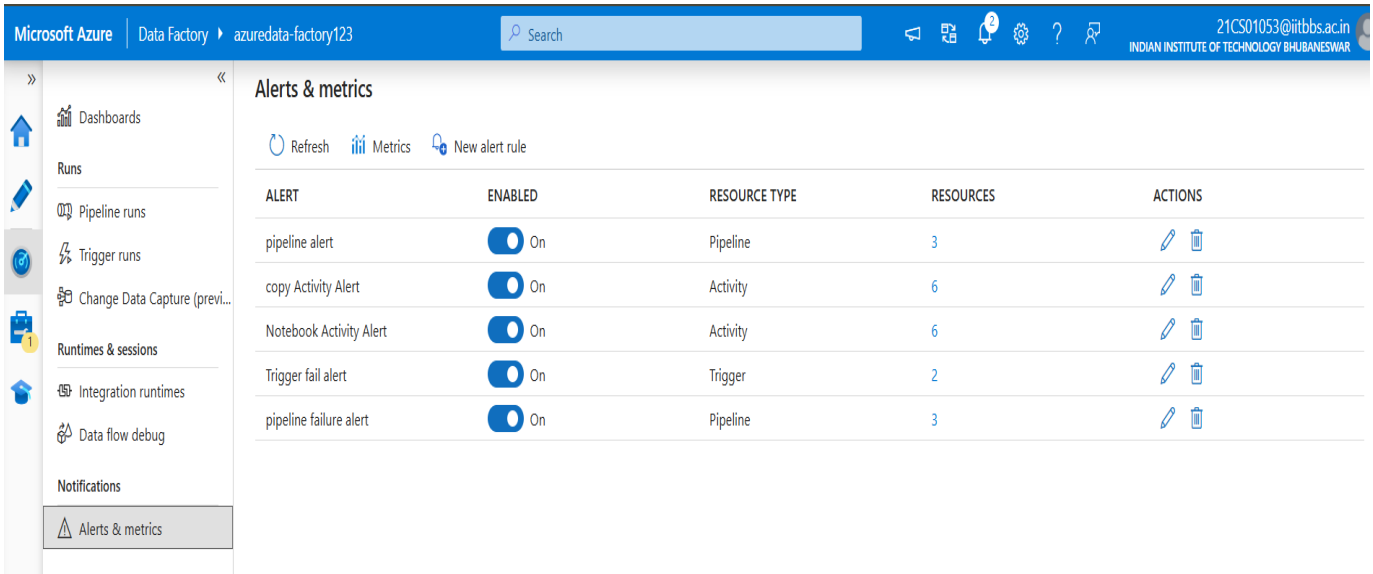


✓ Query the logs



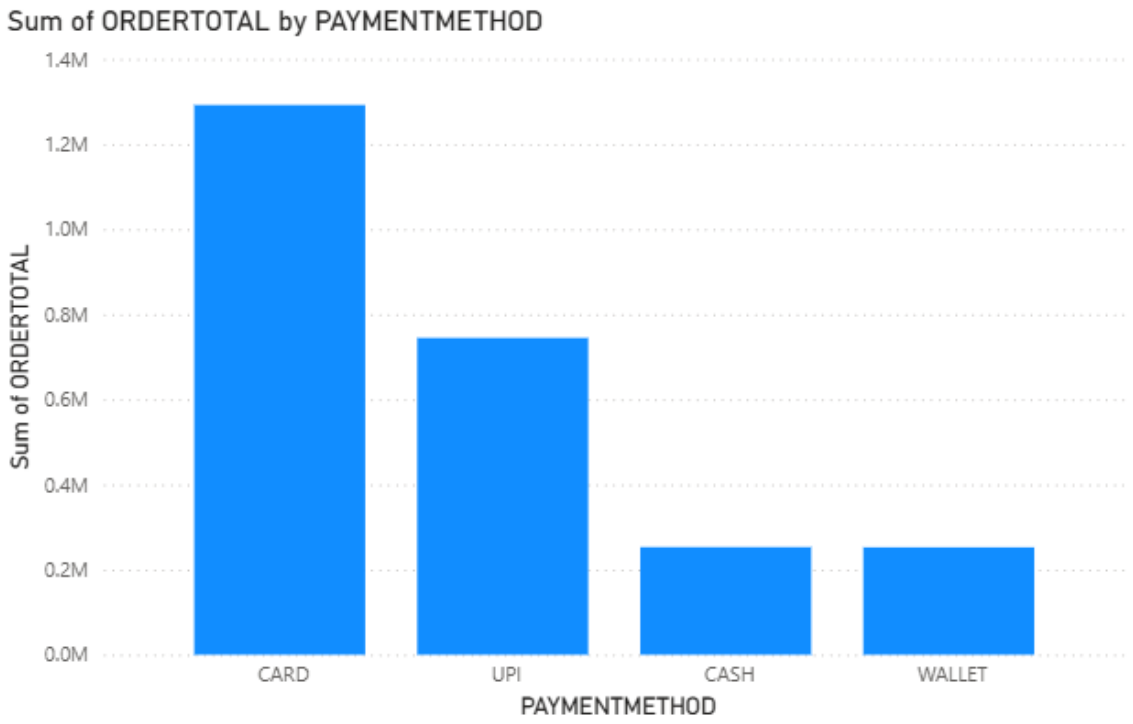
ALERTS:

✓ Create alerts for pipeline failures and activity failures



POWER BI:

✓ Received more payments through CARD



Sum of ORDERTOTAL	Sum of QUANTITY	PAYMENTID
4,134.00	78.00	PMT00005815
2,183.04	48.00	PMT00008562
6,695.04	88.00	PMT00009265
4,978.05	105.00	PMT00010147
2,329.32	42.00	PMT00020180
3,092.40	40.00	PMT00022914
2,511.88	28.00	PMT00024537
4,909.96	44.00	PMT00025521
5,337.16	43.00	PMT00027202
2,471.04	99.00	PMT00028704
38,641.89	**615.00**	