Bootcamp Project 2 - Transactions and Loan Data for a Customer

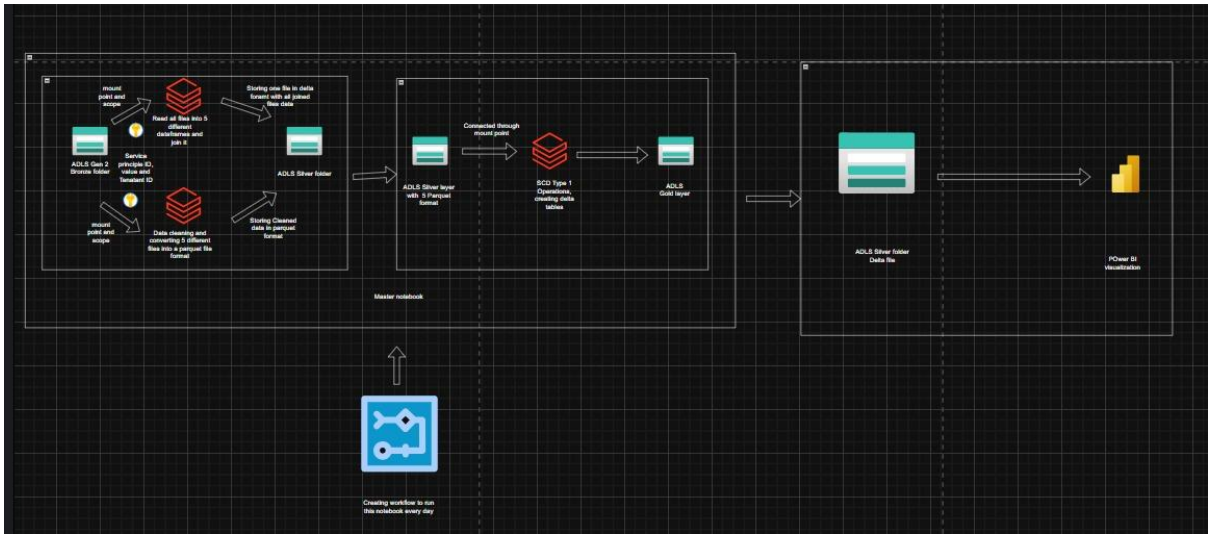
Objective:

The 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.

Tools Required:

- Azure Data Lake Gen 2 Storage
- Azure Data Bricks
- Power BI Desktop
- Microsoft Fabric
- Draw.io

Architecture Diagram:



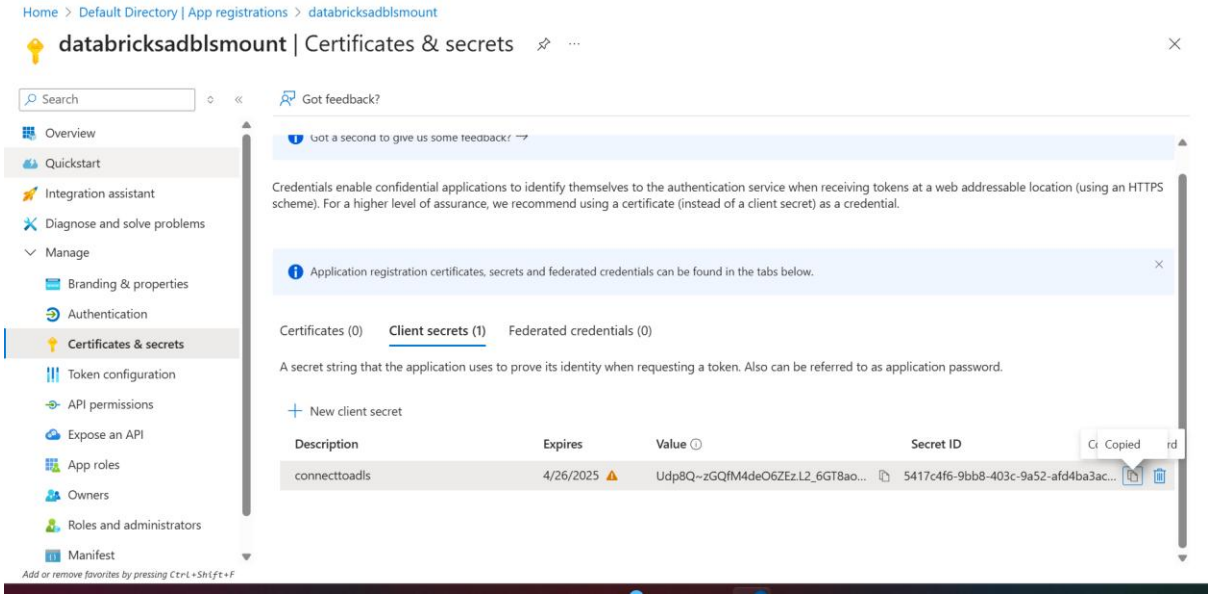
Bronze layer to Silver Layer:

Our goal is to get data from Bronzer layer to Silver Layer which is cleaned.

We are using Azure Databricks to perform these operations, to connect bronze layer to Databricks which is in Azure Data Lake Gen 2 we have to create a mount point between them, for that we need to use service principle.

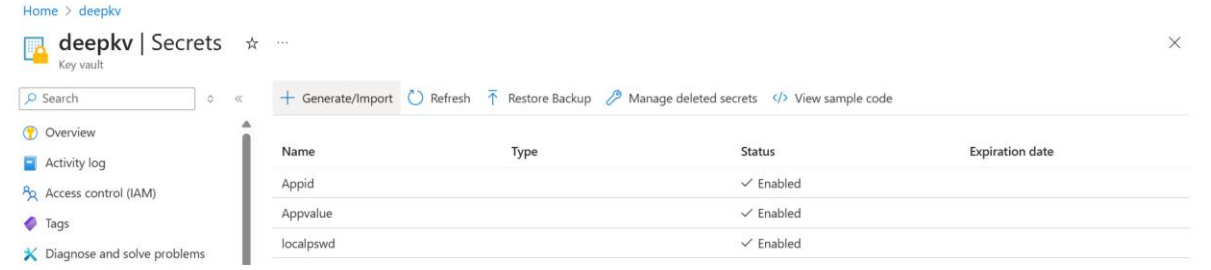
Create a service principle:

Go to Microsoft Entra ID -> Manage -> App registration -> create a service principle -> copy Application ID and Tenant ID -> go to Client services and create new client secret

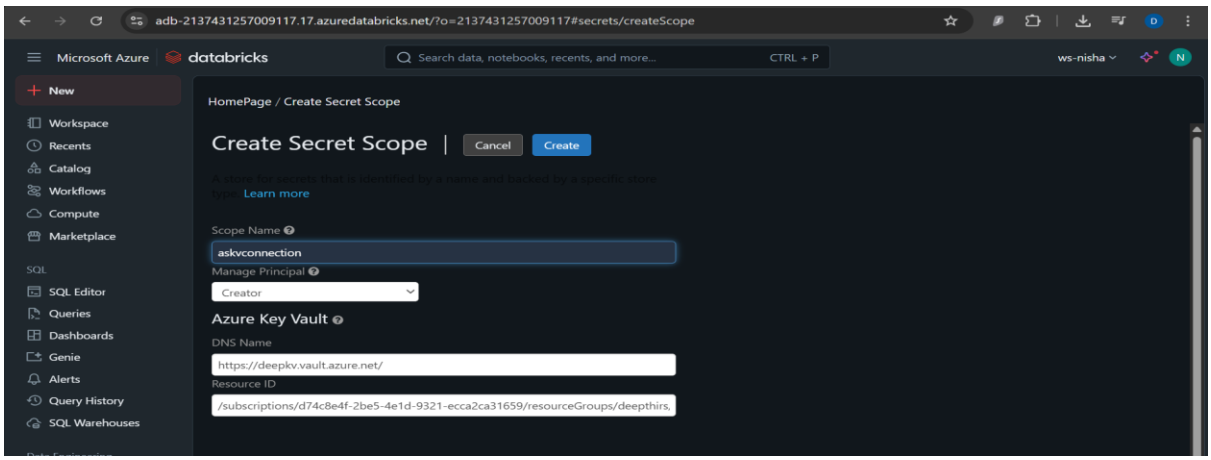


Copy all required details.

Now we have to create secrets in Key vault.



Create a scope in Databricks for this key vault.



Now we can create a mount point to Azure Data Lake gen 2 using below code

```
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":
dbutils.secrets.get(scope="askvconnection",key="Appid"),
       "fs.azure.account.oauth2.client.secret":
dbutils.secrets.get(scope="askvconnection",key="Appvalue"),
       "fs.azure.account.oauth2.client.endpoint": "https://login.microsoftonline.com/Tenant
ID/oauth2/token"}
dbutils.fs.mount(
  source = "abfss://bc-project002@nishastorgae.dfs.core.windows.net/",
  mount_point = "/mnt/bc-project002",
  extra_configs = configs)
```

Then we use pyspark code to read data from bronze layer, remove duplicates and null values and store the cleaned data into Silver layer in parquet format for all the files in Bronze layer.

```
df_account =spark.read.format("csv").option("header", "true").load("/mnt/bc-project002/bronze/accounts.csv")
df_account=df_account.na.drop()
df_account = df_account.dropDuplicates()
df_account.write.mode("overwrite").format("parquet").save("/mnt/bc-project002/silver/Accounts")
```

```
df_customer =spark.read.format("csv").option("header", "true").load("/mnt/bc-project002/bronze/customers.csv")
df_customer=df_customer.na.drop()
df_customer = df_customer.dropDuplicates()
df_customer.write.mode("overwrite").format("parquet").save("/mnt/bc-project002/silver/Customers")
```

```
df_transactions =spark.read.format("csv").option("header", "true").load("/mnt/bc-project002/bronze/transactions.csv")
df_transactions=df_transactions.na.drop()
df_transactions = df_transactions.dropDuplicates()
df_transactions.write.mode("overwrite").format("parquet").save("/mnt/bc-project002/silver/Transactions")
```

```
df_loans =spark.read.format("csv").option("header", "true").load("/mnt/bc-project002/bronze/loans.csv")
df_loans=df_loans.na.drop()
df_loans = df_loans.dropDuplicates()
df_loans.write.mode("overwrite").format("parquet").save("/mnt/bc-project002/silver/Loans")
```

```
df_loan_payments =spark.read.format("csv").option("header", "true").load("/mnt/bc-project002/bronze/loan_payments.
csv")
df_loan_payments=df_loan_payments.na.drop()
df_loan_payments = df_loan_payments.dropDuplicates()
df_loan_payments.write.mode("overwrite").format("parquet").save("/mnt/bc-project002/silver/Loans_payment")
```

We have another task to join all these files into one file and store in Silver later in parquet format.

```
df_join1 = df_account.join(df_customer, on="customer_id", how="left")
df_join2 = df_join1.join(df_transactions, on="account_id", how="left")
df_join3 = df_join2.join(df_loans, on="customer_id", how="left")
final_df = df_join3.join(df_loan_payments, on="loan_id", how="left")
```

```
from pyspark.sql.functions import col

final_join = final_df.select(
    col("account_id").cast("int"),
    col("transaction_id").cast("int"),
    col("customer_id").cast("int"),
    col("loan_id").cast("int"),
    col("payment_id").cast("int"),
    col("balance").cast("float"),
    col("transaction_date").cast("timestamp"),
    col("transaction_amount").cast("float"),
    col("loan_amount").cast("float"),
    col("payment_amount").cast("float"),
    col("payment_date").cast("timestamp")
)
```

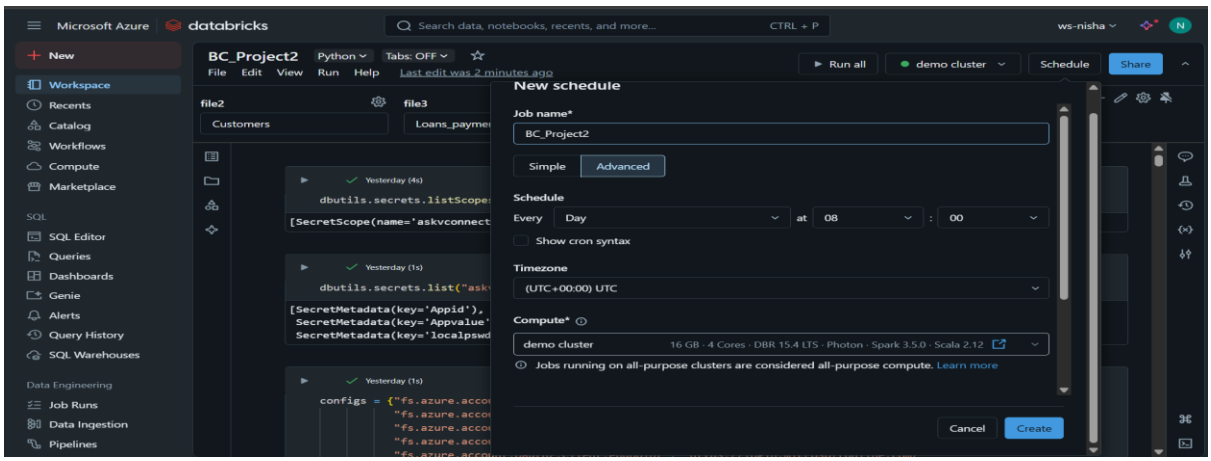
```
final_join.write.format("parquet").option("header", True).mode("overwrite").save("/mnt/bc-project002/silver/
Final_Join")
```

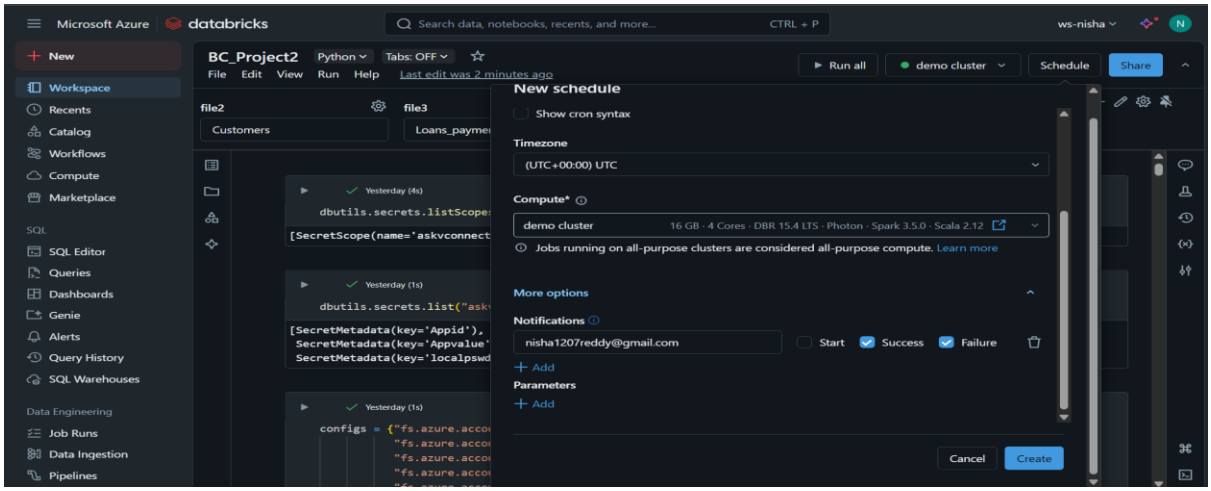
Once the data is copied to Silver layer, we have to perform SCD Type 1 on all the files data. Used below code for all 5 files to perform SCD Type 1

```
%sql

create table if not exists delta.`/mnt/bc-project002/gold/Accounts/`(
  account_id int,
  customer_id int,
  account_type string,
  balance float,
  hash_key bigint,
  created_by string,
  created_date timestamp,
  updated_by string,
  updated_date timestamp
)
src_path="/mnt/bc-project002/silver/"+filedate
print(src_path)
tgt_path="/mnt/bc-project002/gold/Accounts/"
print(tgt_path)
df_src=spark.read.format("parquet").option("header", "true").option("inferSchema", "true").load(src_path)
display(df_src)
df_src1=df_src.withColumn("hash_key",crc32(concat(*df_src.columns)))
display(df_src1)
```

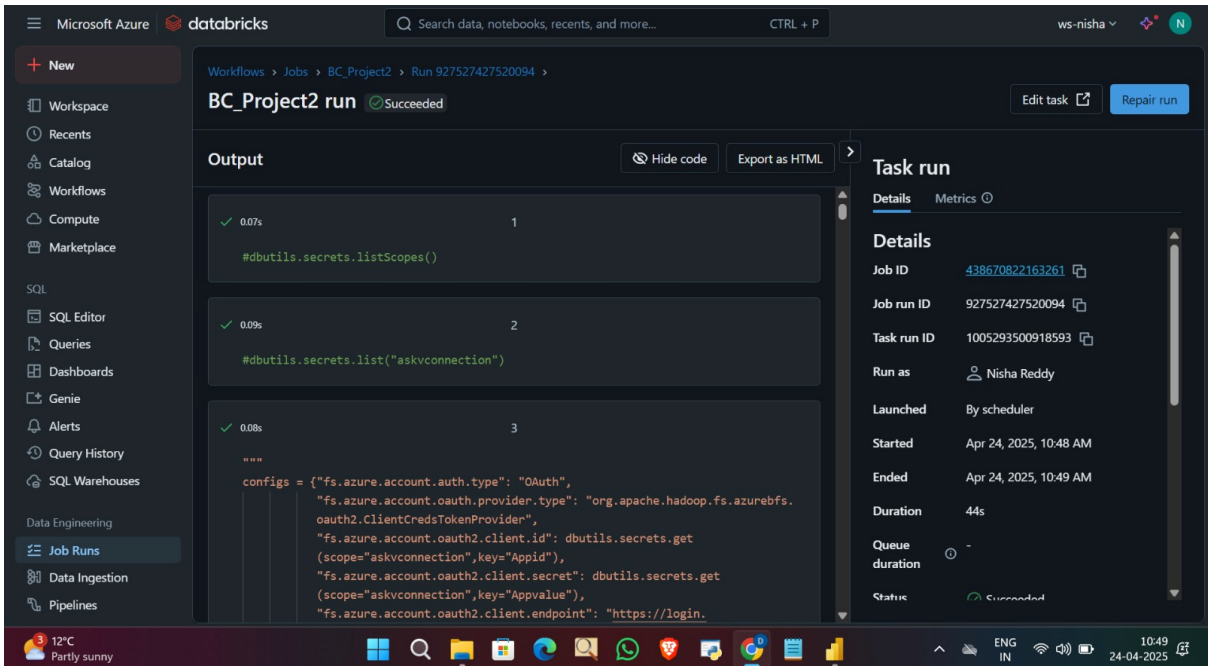
```
from delta.tables import DeltaTable
dbtable = DeltaTable.forPath(spark, tgt_path)
dbtable.toDF().show()
df_src1=df_src1.alias("src").join(dbtable.toDF().alias("tgt"), ((col("src.account_id") == col("tgt.account_id")) &
(col("src.hash_key") == col("tgt.hash_key"))), "anti").select(col("src.*"))
df_src1.show()

dbtable.alias("tgt").merge(df_src1.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.hash_key":"src.hash_key","tgt.
    updated_date":current_timestamp(),"tgt.updated_by":lit("databricks_Updated")})\
        .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.hash_key":"src.hash_key","tgt.
        created_date":current_timestamp(),"tgt.created_by":lit("databricks"),"tgt.updated_date":current_timestamp(),
        "tgt.updated_by":lit("databricks")}).execute()
display(spark.read.format("delta").option("header","true").load(tgt_path))
```

Once SCD Type 1 is done we have to create a schedule job to run pipeline everyday or any specified time.

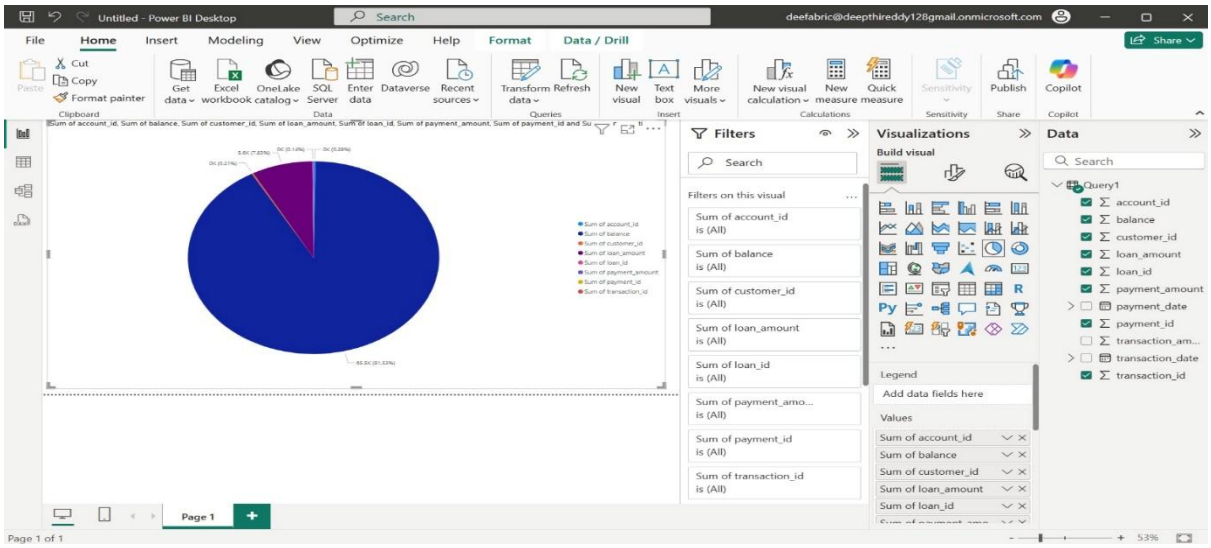




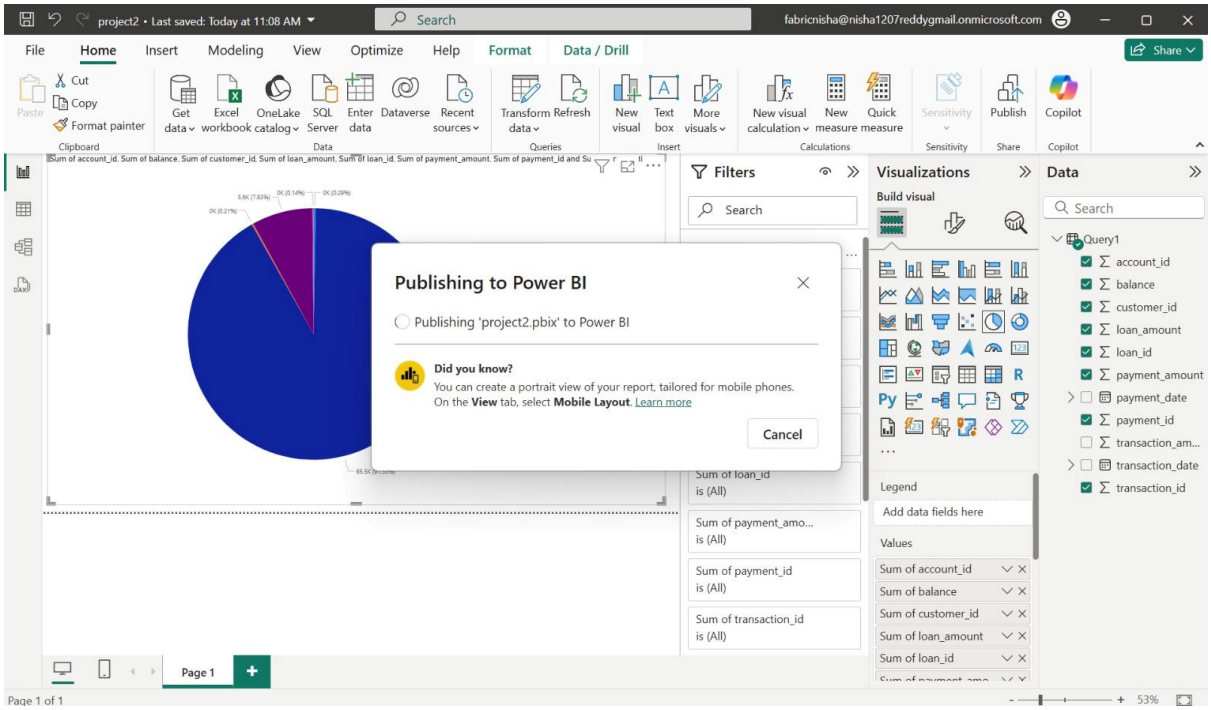
Schedule ran successfully.



Now loading the joined file into Power BI Desktop to generate a report.



Loaded this report to Fabric account.



Report is successfully published in fabric workspace.