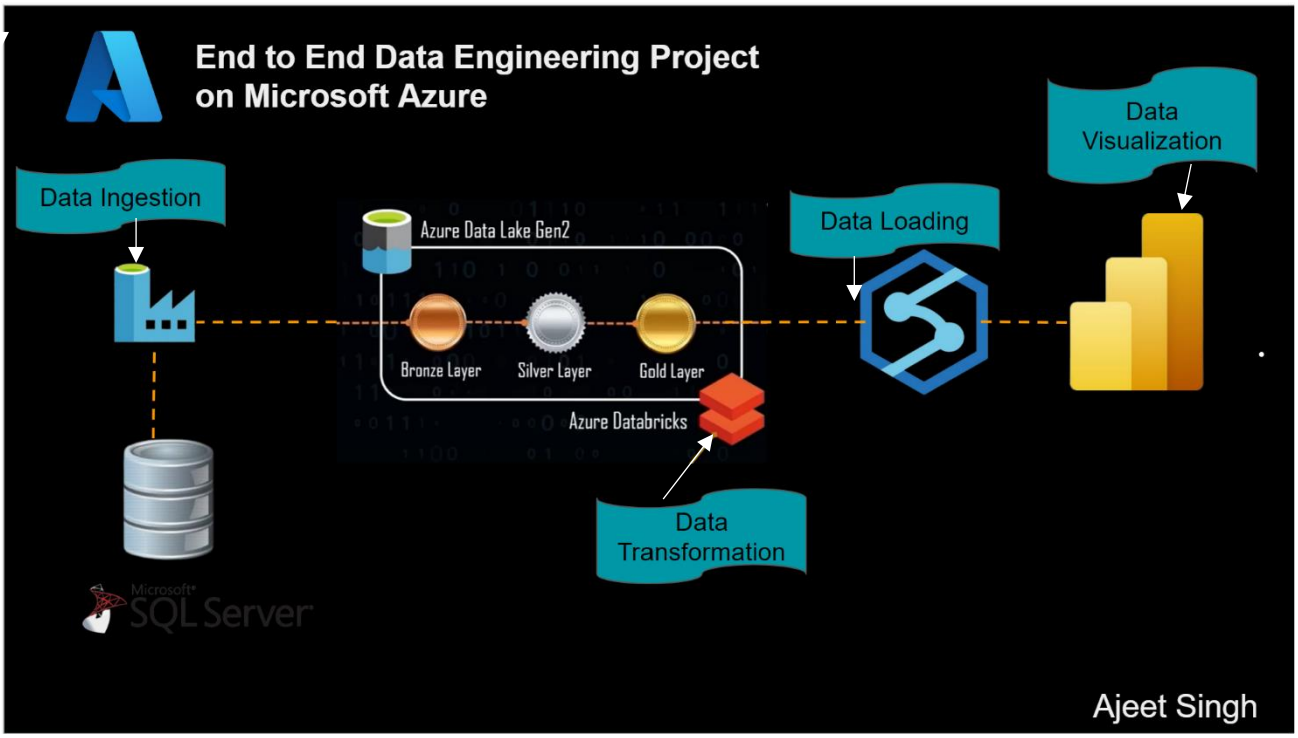
End-to-End Data Engineering Project

Objective:

The objective of this project of this project is to build a data pipeline to store data in Data Lake from a local computer, perform different types of transformation, load the data cloud-based database, and connect the data visualization tool to the cloud-based database to perform data visualization and perform end to end data pipeline testing.



Different steps involved in this project:

I. Environment Setup
II. Data Ingestion
III. Data Transformation
IV. Data Loading
V. Data Reporting
VI. End-to-end Pipeline Testing

Different tools and technology used in this project:

MS SQL Server:

MS SQL Server is a Relational Database Management System (RDBMS) developed by Microsoft. A Relational database is based on a Relational Model architecture. The data is

organized in tables(relations), and the tables are related to each other. Each table has rows and columns(attributes). MS SQL Server is a software product used to administer the database and retrieve information.

Self-Hosted Integration Runtime:

A self-hosted integration runtime (SHIR) is a secure, managed, and serverless computing infrastructure that allows data integration between on-premises systems and the Azure cloud. It is a component of Azure Integration Runtime (AIR) that is used when an on-premises data source cannot be accessed directly from the cloud. SHIR serves as a bridge between on-premises and Azure services.

Azure Data Factory:

Azure Data Factory (ADF) is a cloud-based data integration service that allows users to create workflows for data transformation and movement. It's serverless and fully managed and can simplify hybrid data integration at an enterprise scale. ADF is available in over 25 regions and offers a pay-as-you-go service that scales on demand.

Azure Data Lake Gen2:

Azure Data Lake Storage Gen2 (ADLS Gen2) is a service that stores and analyzes big data in its original file formats. It's built on Azure Blob Storage and combines the capabilities of Azure Data Lake Storage Gen1 with Azure Blob Storage. ADLS Gen2 can store text, CSV, JSON, XML, image, and video files. It's designed to work with Hadoop and frameworks that use the Apache Hadoop Distributed File System (HDFS).

Azure Synapse Analyst:

Azure Synapse Analytics is a cloud-based enterprise analytics service that combines SQL technologies, big data, and Azure Data Explorer to speed up time to insight across data warehouses and big data systems. It offers cloud data warehousing, dashboarding, and machine learning analytics. Azure Synapse Analytics is designed for data mining and data exploration and can handle high volumes of data and very complex queries and aggregation. It also supports up to 128 concurrent queries.

Azure Key Vault:

Azure Key Vault is a cloud service that provides a secure store for secrets. You can securely store keys, passwords, certificates, and other secrets. Azure key vaults may be created and managed through the Azure portal. In this QuickStart, you create a key vault and then use it to store a secret.

Power BI:

Microsoft Power BI is a collection of apps, services, and connectors that help users visualize and connect to data from different sources. Power BI can help users make better decisions, and can be used to: Uncover insights with AI, Turn data into visuals, Establish a governed source of truth, Unify enterprise scale and self-service, and Infuse data experiences everywhere.

Project Explanation:

Step:1 (Download the Data from Microsoft Learn)

Link of Data: https://learn.microsoft.com/en-us/sql/samples/adventureworks-install-configure?view=sql-server-ver16&tabs=ssms

Download backup files

Use these links to download the appropriate sample database for your scenario.

- **OLTP** data is for most typical online transaction processing workloads.
- **Data Warehouse (DW)** data is for data warehousing workloads.
- **Lightweight (LT)** data is a lightweight and pared down version of the **OLTP** sample.

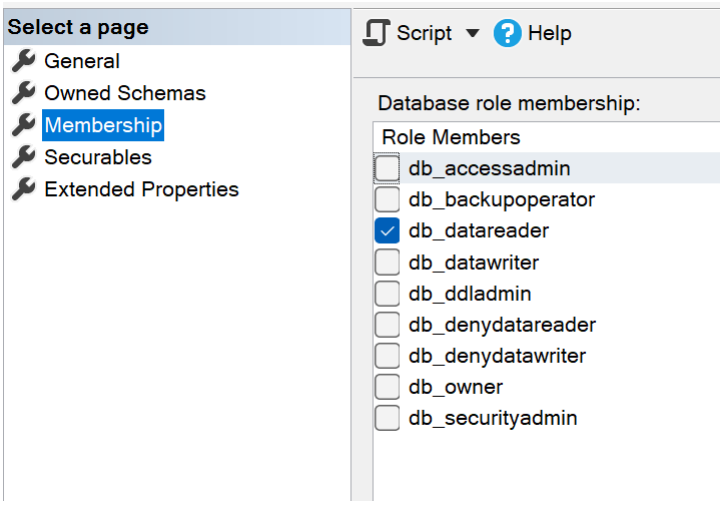
If you're not sure what you need, start with the OLTP version that matches your SQL Server version.

OLTP	Data Warehouse	Lightweight
AdventureWorks2022.bak	AdventureWorksDW2022.bak	AdventureWorksLT2022.bak
AdventureWorks2019.bak	AdventureWorksDW2019.bak	AdventureWorksLT2019.bak
AdventureWorks2017.bak	AdventureWorksDW2017.bak	AdventureWorksLT2017.bak
AdventureWorks2016.bak	AdventureWorksDW2016.bak	AdventureWorksLT2016.bak

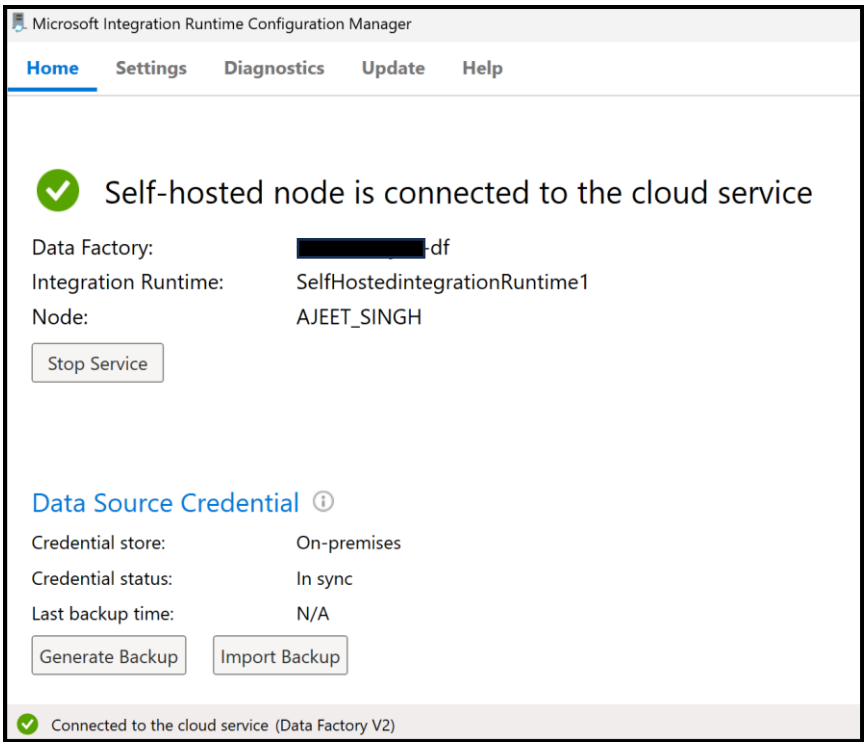
Different tables in the dataset:

- SalesLT.Address
- SalesLT.Customer
- SalesLT.CustomerAddress
- SalesLT.Product
- SalesLT.ProductCategory
- SalesLT.ProductDescription
- SalesLT.ProductModel
- SalesLT.ProductModelProductDescription
- SalesLT.SalesOrderDetail
- SalesLT.SalesOrderHeader

Give permission to the user for read the data:



Setup the Self-hosted integration runtime:



Step:2 Data Ingestion

Edit linked service

SQL server Learn more

Name *

OnPremSqlServer

Description

Connect via integration runtime *

SelfHostedintegrationRuntime1

Connection string | Azure Key Vault

Server name *

AJEET_SINGH\SQLEXPRESS

Database name *

AdventureWorksLT2017

Authentication type

SQL authentication

User name *

ajeet

Password | Azure Key Vault

AKV linked service *

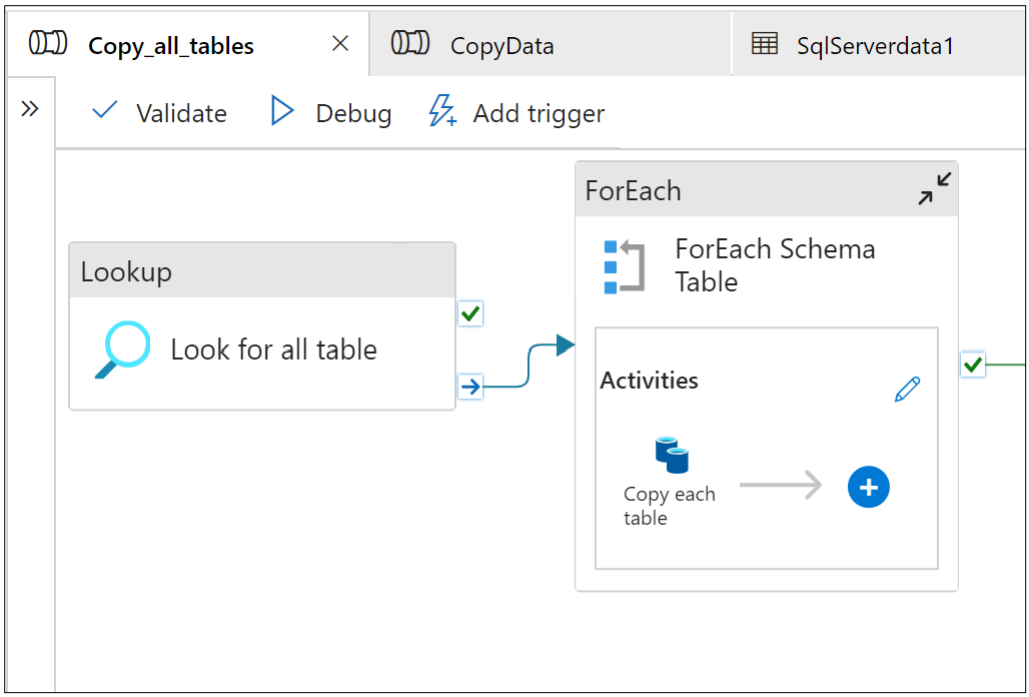
AzureKeyVault1

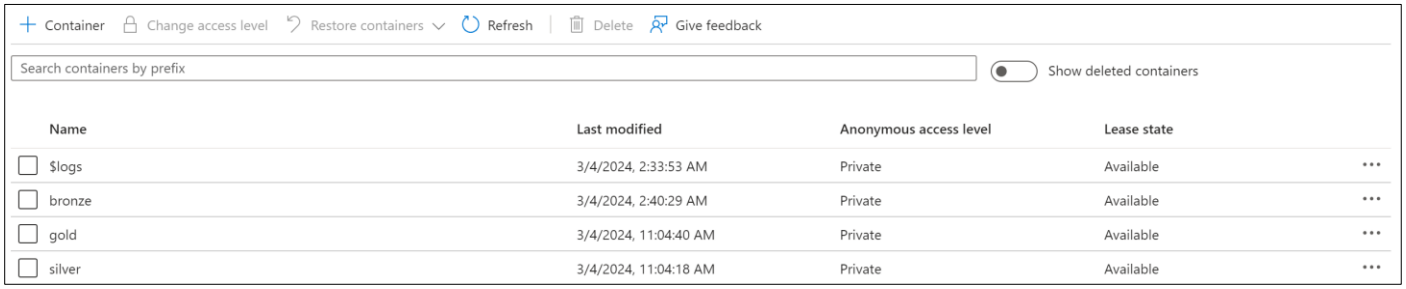
Secret name *

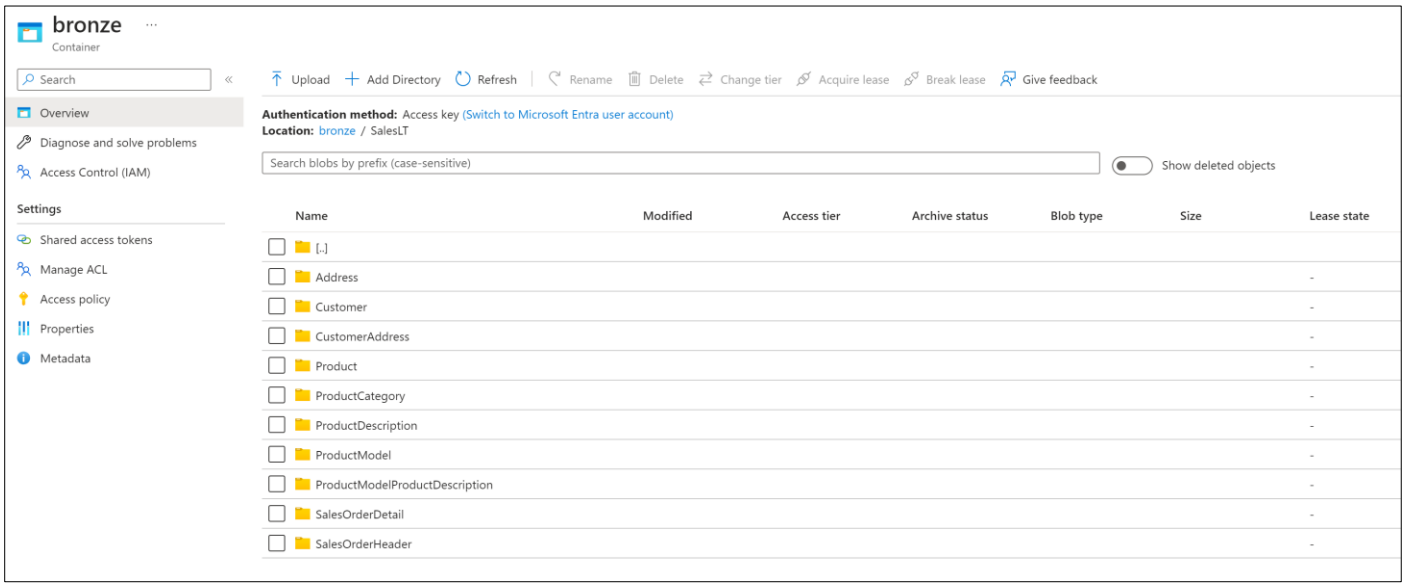
password

Edit

Secret version







Step:-3 Data Transformation:

mounting

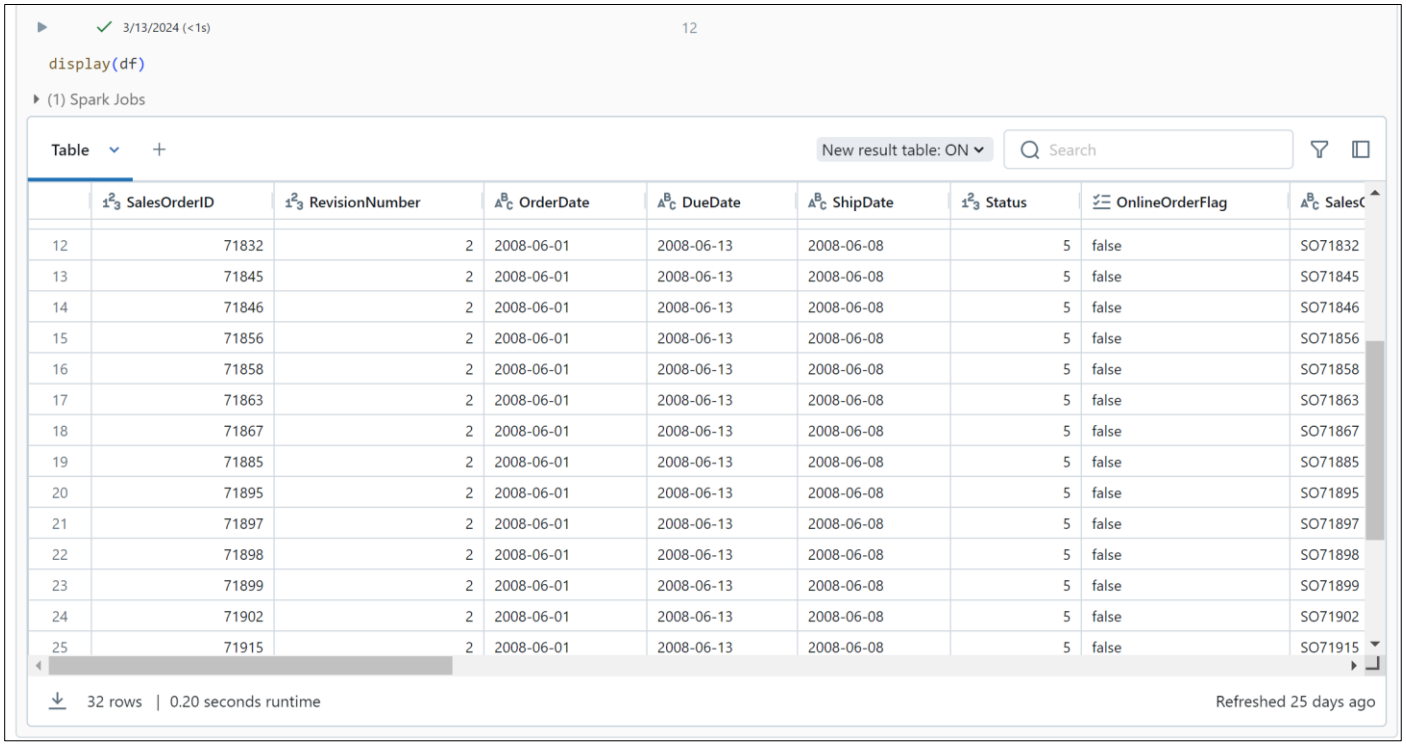
Silver level transformation

3/13/2024 (22s) 11

```
from pyspark.sql.functions import from_utc_timestamp, date_format
from pyspark.sql.types import TimestampType

# Assuming you have a list of table names
#table_names = [...]

for i in table_name:
    path = '/mnt/bronze/SalesLT/' + i + '/' + i + '.parquet'

    # Read the Parquet file
    df = spark.read.format('parquet').load(path)

    # Get column names
    columns = df.columns

    # Process date columns
    for col in columns:
        if "Date" in col or "date" in col:
            df = df.withColumn(col, date_format(from_utc_timestamp(df[col].cast(TimestampType()), "UTC"), "yyyy-MM-dd"))

    # Write the processed DataFrame to Delta format
    output_path = '/mnt/silver/SalesLT/' + i + '/'
    df.write.format('delta').mode("overwrite").save(output_path)
```

(50) Spark Jobs

df: pyspark.sql.dataframe.DataFrame = [SalesOrderID: integer, RevisionNumber: integer ... 20 more fields]



	SalesOrderID	RevisionNumber	OrderDate	DueDate	ShipDate	Status	OnlineOrderFlag	SalesO
12	71832	2	2008-06-01	2008-06-13	2008-06-08	5	false	SO71832
13	71845	2	2008-06-01	2008-06-13	2008-06-08	5	false	SO71845
14	71846	2	2008-06-01	2008-06-13	2008-06-08	5	false	SO71846
15	71856	2	2008-06-01	2008-06-13	2008-06-08	5	false	SO71856
16	71858	2	2008-06-01	2008-06-13	2008-06-08	5	false	SO71858
17	71863	2	2008-06-01	2008-06-13	2008-06-08	5	false	SO71863
18	71867	2	2008-06-01	2008-06-13	2008-06-08	5	false	SO71867
19	71885	2	2008-06-01	2008-06-13	2008-06-08	5	false	SO71885
20	71895	2	2008-06-01	2008-06-13	2008-06-08	5	false	SO71895
21	71897	2	2008-06-01	2008-06-13	2008-06-08	5	false	SO71897
22	71898	2	2008-06-01	2008-06-13	2008-06-08	5	false	SO71898
23	71899	2	2008-06-01	2008-06-13	2008-06-08	5	false	SO71899
24	71902	2	2008-06-01	2008-06-13	2008-06-08	5	false	SO71902
25	71915	2	2008-06-01	2008-06-13	2008-06-08	5	false	SO71915

Gold level transformation

3/13/2024 (<1s) 4

```
df = spark.read.format('delta').load('/mnt/silver/SalesLT/Address/')
```

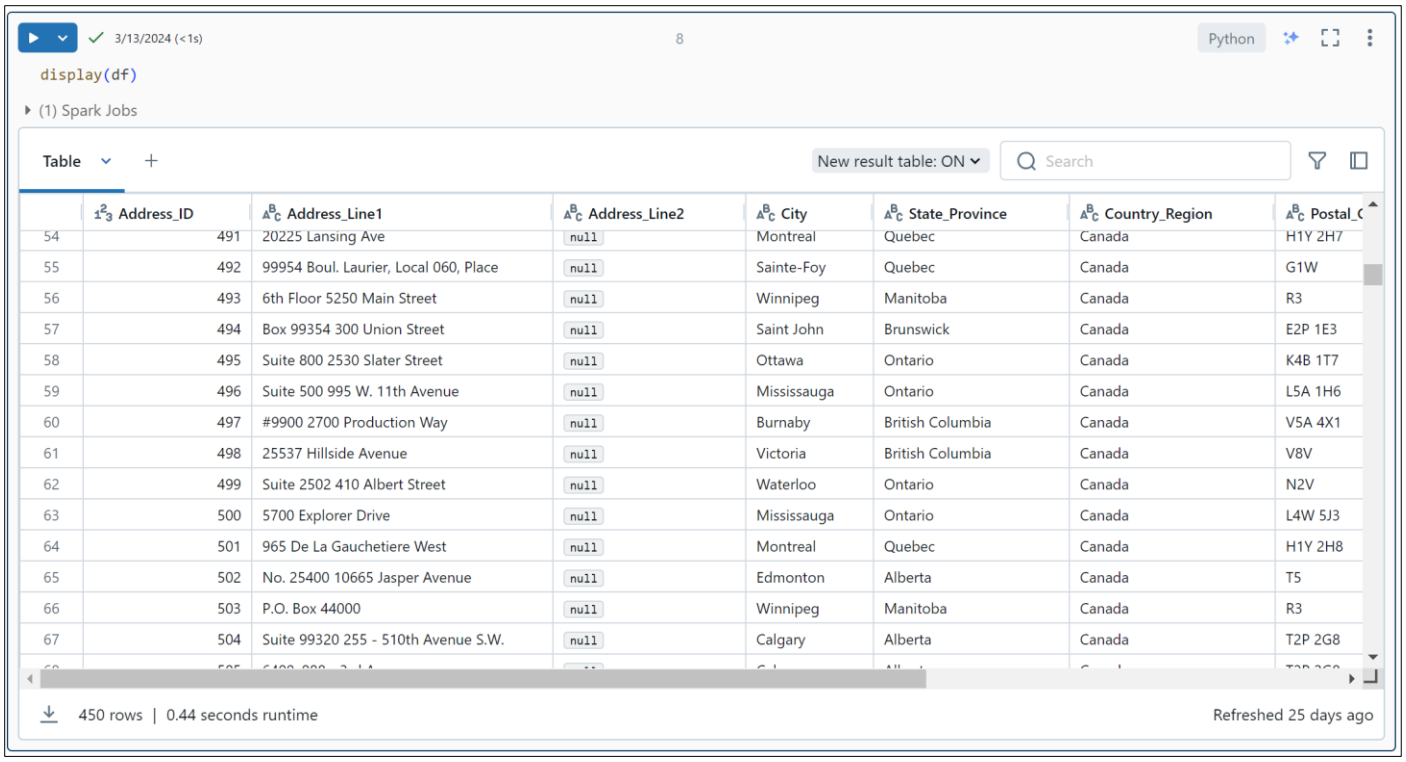
df: pyspark.sql.dataframe.DataFrame

Schema Details History

```
AddressID: integer
AddressLine1: string
AddressLine2: string
City: string
StateProvince: string
CountryRegion: string
PostalCode: string
rowguid: string
ModifiedDate: string
```

3/13/2024 (<1s) 7

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, regexp_replace

# Assuming you have a SparkSession created
#spark = SparkSession.builder.appName("example").getOrCreate()

# Assuming you have a DataFrame named df
# If not, you need to read your data first using df = spark.read.

# Get the list of column names
column_names = df.columns

for old_col_name in column_names:
    # Convert column name from ColumnName to Column_Name format
    new_col_name = "".join(["_" + char if char.isupper() and not old_col_name[i - 1].isupper() else char for i, char in enumerate(old_col_name)]).
    lstrip("_")
    # Change the column name using withColumnRenamed and regexp_replace
    df = df.withColumnRenamed(old_col_name, new_col_name)
```



3/13/2024 (<1s) 8 Python

```
display(df)
```

(1) Spark Jobs

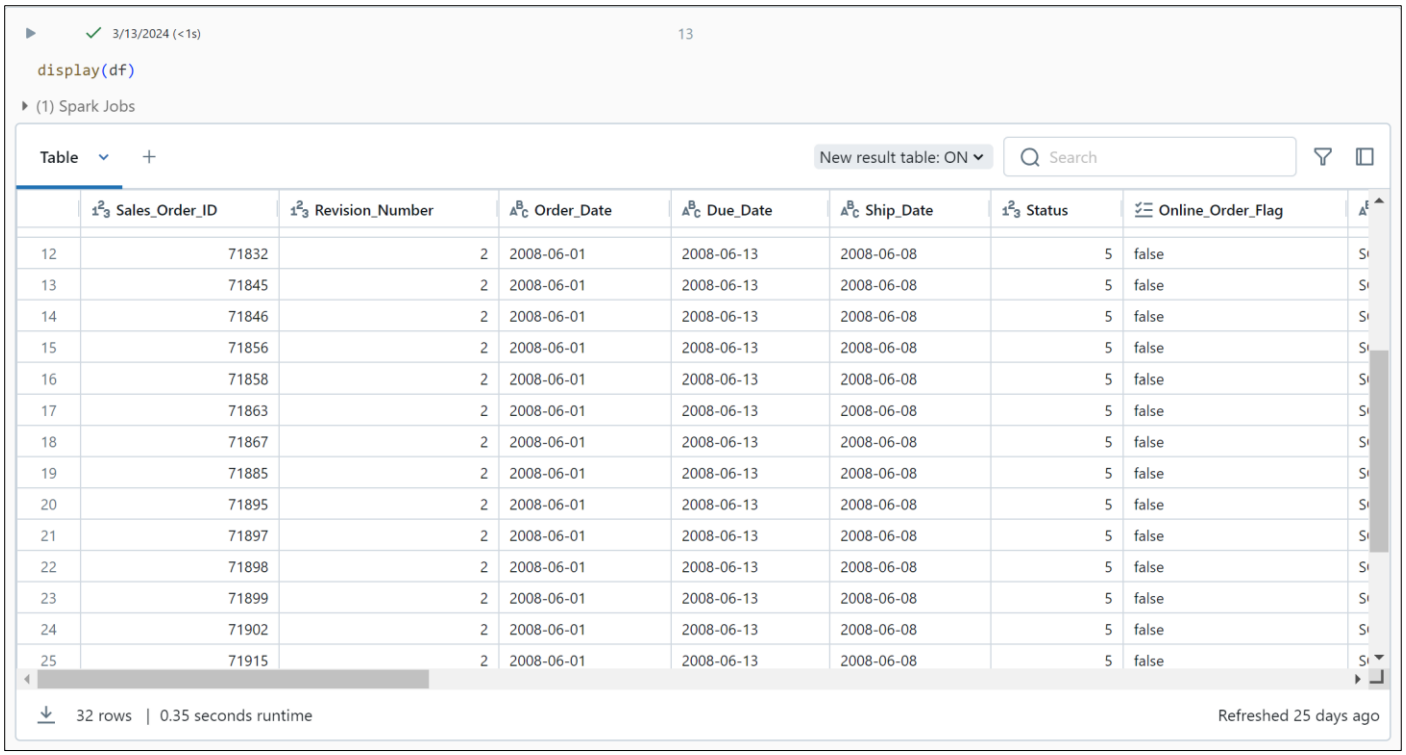
Table + New result table: ON Search

	Address_ID	Address_Line1	Address_Line2	City	State_Province	Country_Region	Postal_C
54	491	20225 Lansing Ave	null	Montreal	Quebec	Canada	H1Y 2H7
55	492	99954 Boul. Laurier, Local 060, Place	null	Sainte-Foy	Quebec	Canada	G1W
56	493	6th Floor 5250 Main Street	null	Winnipeg	Manitoba	Canada	R3
57	494	Box 99354 300 Union Street	null	Saint John	Brunswick	Canada	E2P 1E3
58	495	Suite 800 2530 Slater Street	null	Ottawa	Ontario	Canada	K4B 1T7
59	496	Suite 500 995 W. 11th Avenue	null	Mississauga	Ontario	Canada	L5A 1H6
60	497	#9900 2700 Production Way	null	Burnaby	British Columbia	Canada	V5A 4X1
61	498	25537 Hillside Avenue	null	Victoria	British Columbia	Canada	V8V
62	499	Suite 2502 410 Albert Street	null	Waterloo	Ontario	Canada	N2V
63	500	5700 Explorer Drive	null	Mississauga	Ontario	Canada	L4W 5J3
64	501	965 De La Gauchetiere West	null	Montreal	Quebec	Canada	H1Y 2H8
65	502	No. 25400 10665 Jasper Avenue	null	Edmonton	Alberta	Canada	T5
66	503	P.O. Box 44000	null	Winnipeg	Manitoba	Canada	R3
67	504	Suite 99320 255 - 510th Avenue S.W.	null	Calgary	Alberta	Canada	T2P 2G8

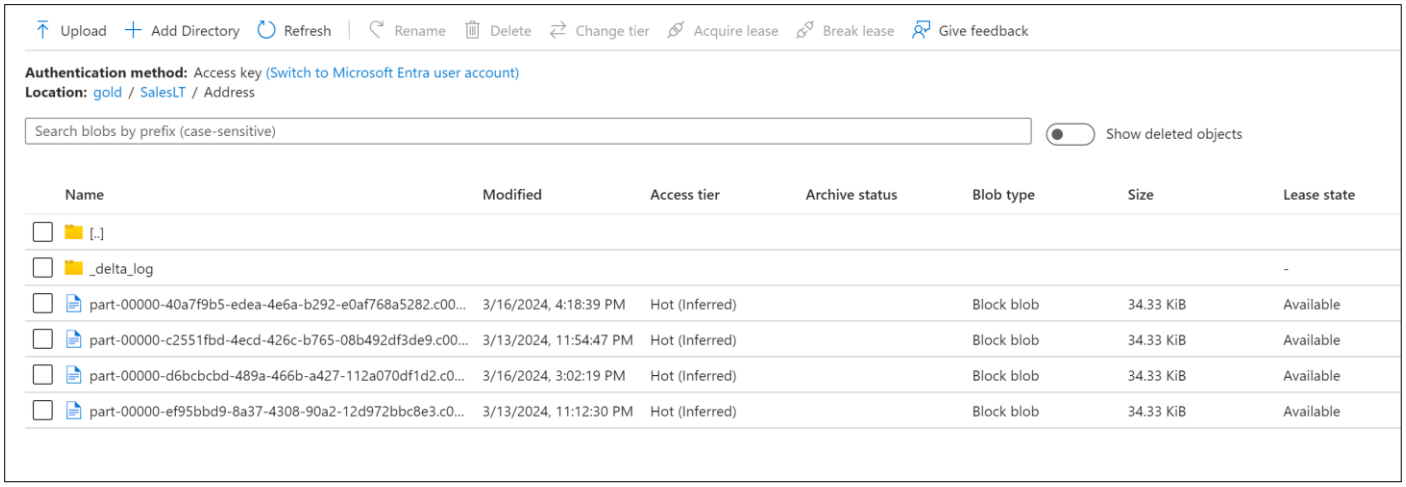
450 rows | 0.44 seconds runtime Refreshed 25 days ago

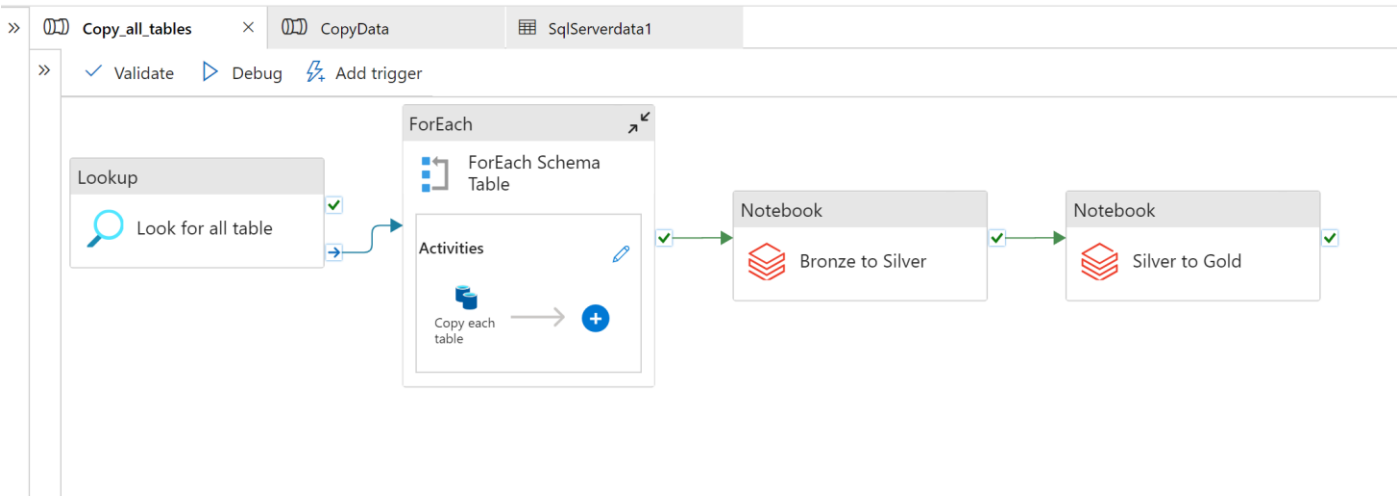
3/13/2024 (17s) 12 Python

```
print(path)
# Read the Delta file
df = spark.read.format('delta').load(path)

# Get list of column names
columns_names = df.columns

# Process date columns
for old_col_name in columns_names:
    # Convert column name from to Column_Name format
    new_col_name = "".join(["_" + char if char.isupper() and not old_col_name[i - 1].isupper() else char for i, char in enumerate(old_col_name)]).
    lstrip("_")
    # Update column name
    df = df.withColumnRenamed(old_col_name, new_col_name)

# Write the processed DataFrame to Delta format
output_path = '/mnt/gold/SalesLT/' + name + '/'
df.write.format('delta').mode("overwrite").save(output_path)
```

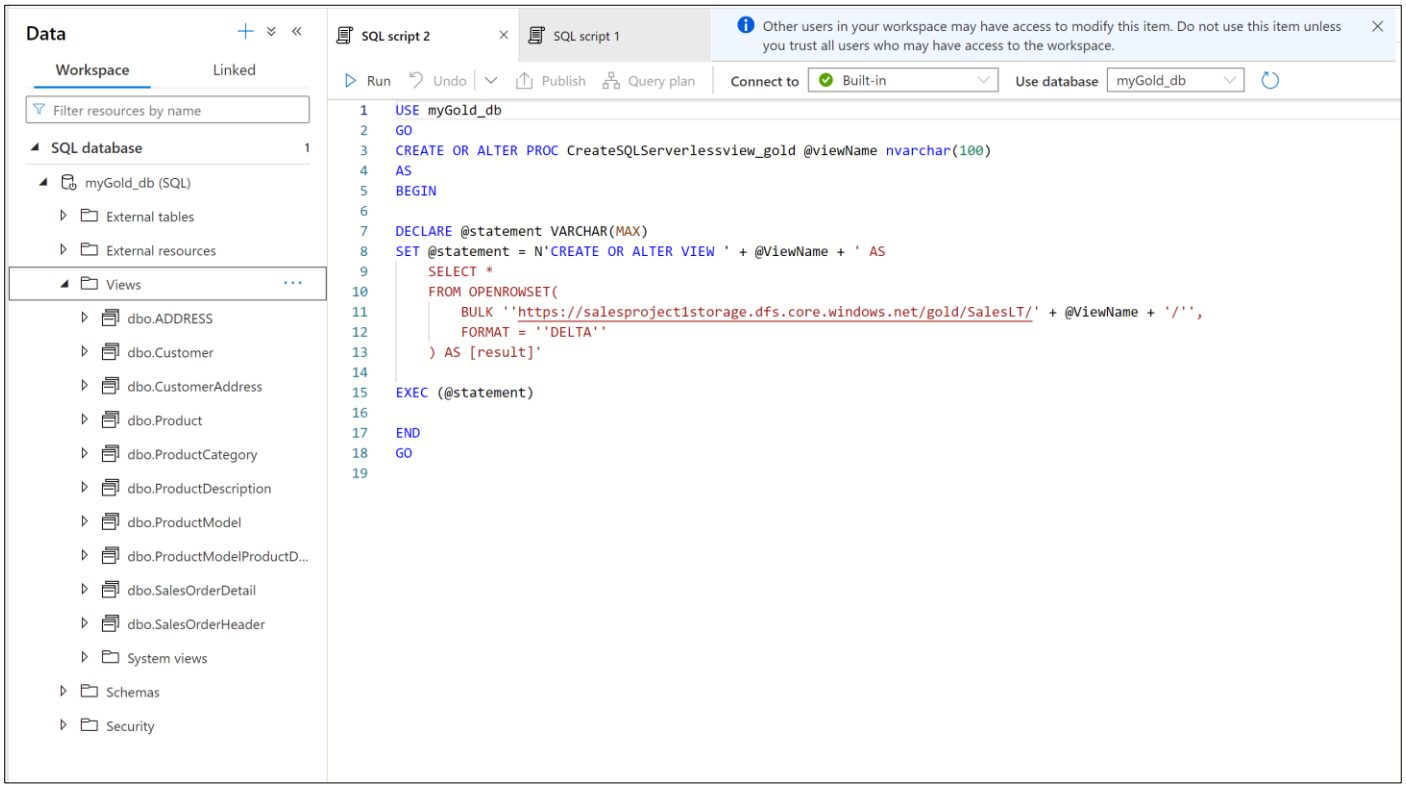


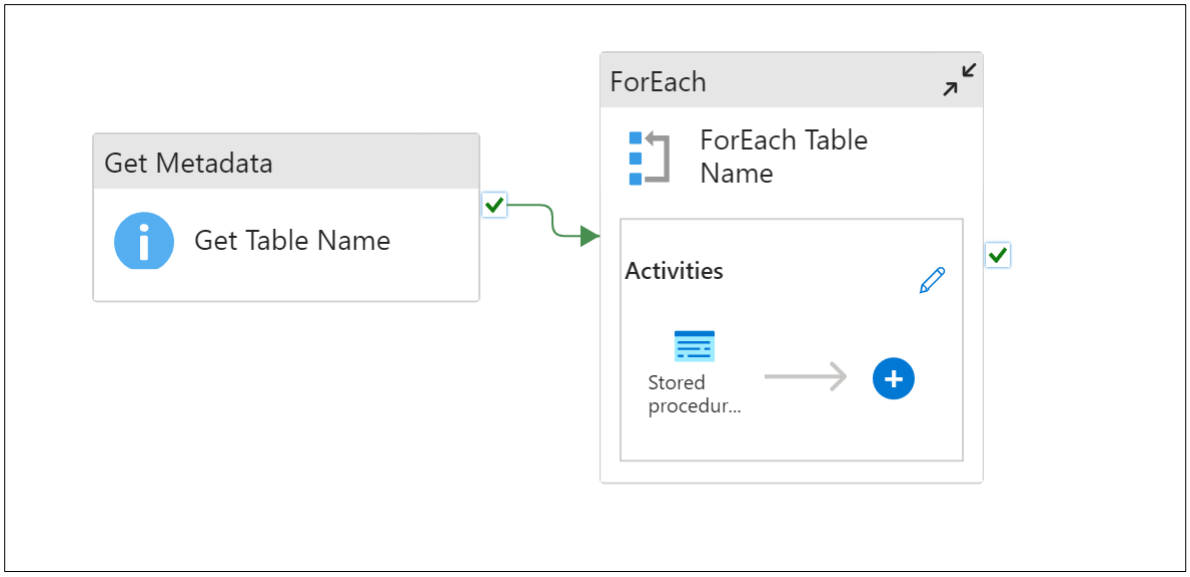
Delta format:



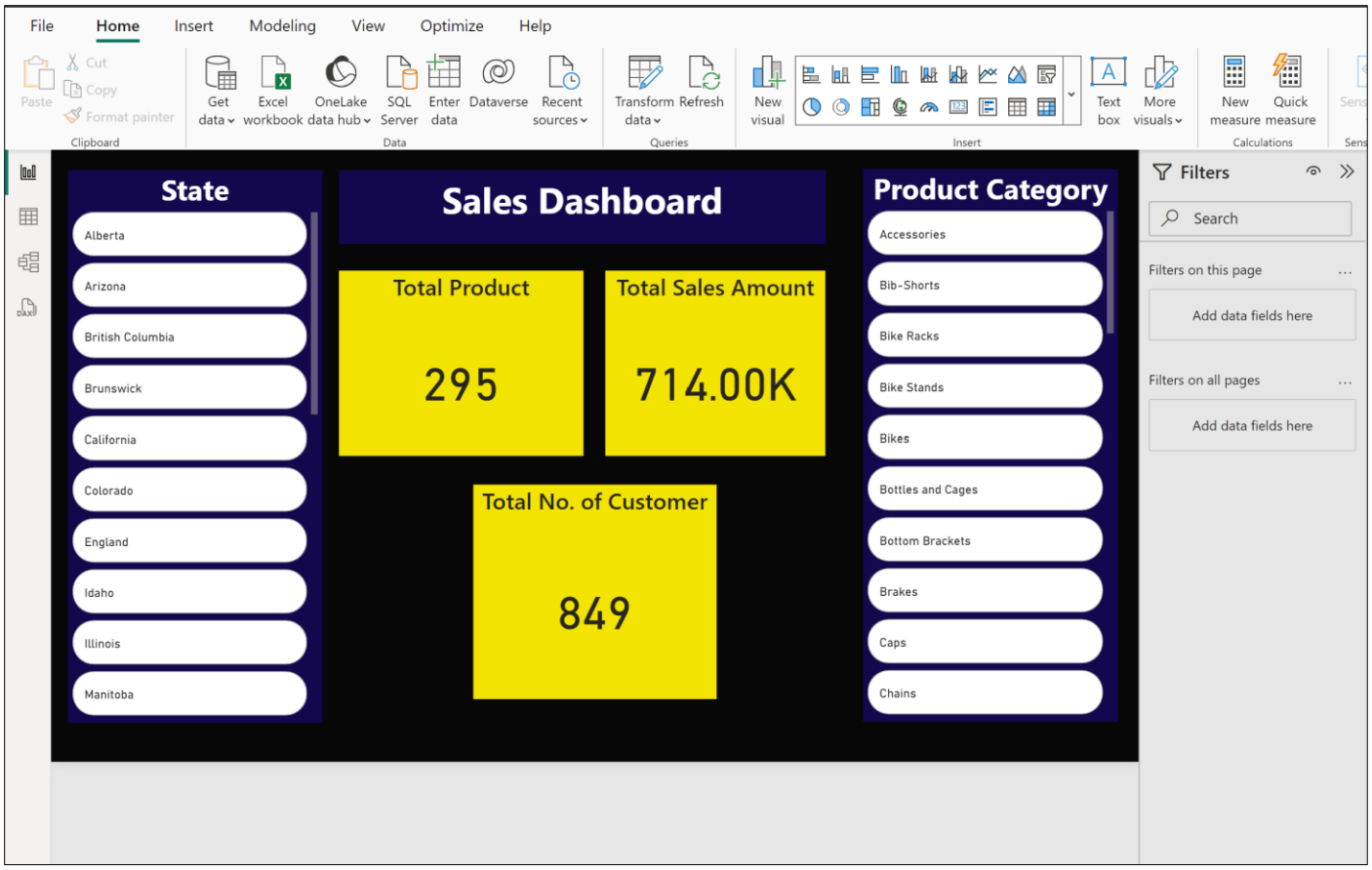


Step:4 Data Loading





Step-5 Data Reporting



Step- 6 Pipeline Testing

If I add a new row in the customer table the Power BI report will show the total no. of customers increased. From 848 to 849