

# Data Pipeline for Customer Account Analysis

## Data Ingestion (Backend Storage to Raw(Bronze) Container)

Step 1: Created Resource group

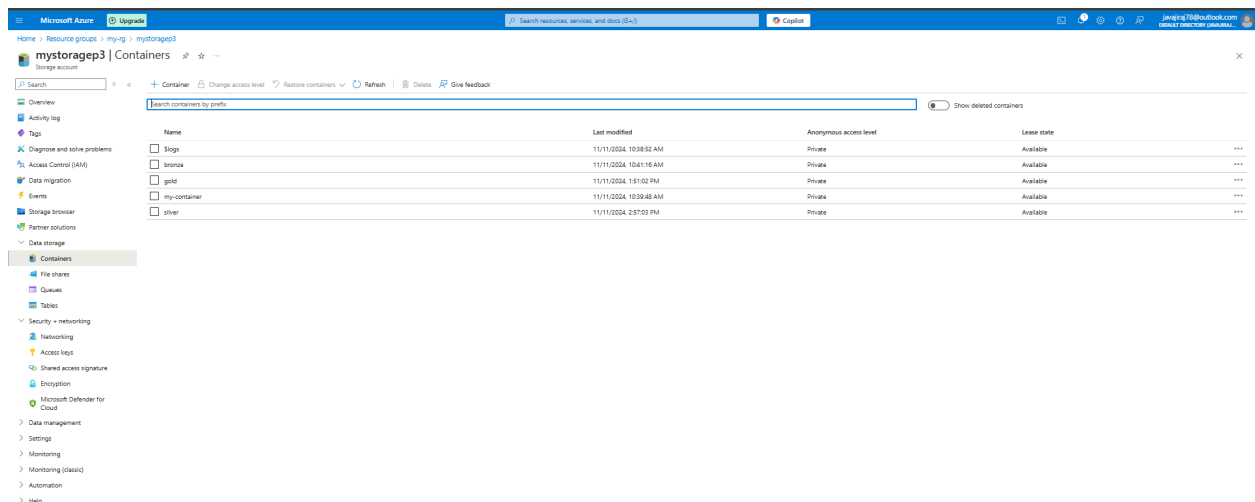
- Resource group name: “my-rg”

Step 2: Created Storage account

- Storage account name: “mystoragep3”

Step 3: Created 4 containers

- 1st Container name: “my-container”
- Uploaded accounts, customers, loan\_payments, loans, transactions files in the container “my-container”.
- 2nd Container name: “bronze”
- Keeping the container empty.
- 3rd Container name: “silver”
- Keeping the container empty.
- 4rd Container name: “gold”
- Keeping the container empty.



Step 4: Created Azure Data Factory

- Azure Data Factory name: “A-datafactory1”
- Launch studio
- Select - Author
- Select - New Pipeline

- Copy Data - Drag and drop

#### Linked Services: “Azure Data lake Gen2”

- Select - Source
- Click on - New
- Select - “Azure Data lake Gen2”
- Select file format - “CSV”
- Name - “1dlstorageegen2”
- Linked service - New
- Name - “lservice1”
- Select - Storage account : “mystorageep3”
- Click on - Test Connection
- Select - Create
- Select - File Path - “my-container” - Ok - Ok
- In Source: file path type : select - wild card file path (All the files in the selected container in source will be selected together)

#### Linked Services: “Azure Data lake Gen2”

- Select - Sink
- Click on - New
- Select - “Azure Data lake Gen2”
- Select file format - “CSV”
- Name - “2dlstorageegen2”
- Linked service - select existing 1st linked service “lservice1”
- File path - Container - “bronze” - ok
- Validate
- Publish all
- Debug

The screenshot displays the Microsoft Azure Data Factory web interface. On the left, the 'Factory Resources' pane shows a list of pipelines, with 'pipeline2' selected. The main canvas shows a pipeline diagram with a single activity named 'Copy data1'. The 'Properties' pane on the right shows the 'General' tab for 'pipeline2'. Below the canvas, the 'Output' tab displays the pipeline run details, including the Pipeline run ID, Pipeline status (Succeeded), and a table of activity results.

Activity name	Activity status	Activity type	Run start	Duration	In
Copy data1	Succeeded	Copy data	11/6/2024, 11:16:25 AM	12s	AI

- Cross check in the storage account “mystoragep1” in the container “bronze1”
- All the files in the “my-container” have been moved to the “Bronze1” container.

Name	Modified	Access tier	Archive status	Blob type	Size	Lease state
accounts.csv	11/6/2024, 11:16:35 AM	Hot (Inferred)		Block blob	2.28 KiB	Available
customers.csv	11/6/2024, 11:16:35 AM	Hot (Inferred)		Block blob	4.5 KiB	Available
loan_payments.csv	11/6/2024, 11:16:35 AM	Hot (Inferred)		Block blob	2.55 KiB	Available
loans.csv	11/6/2024, 11:16:35 AM	Hot (Inferred)		Block blob	2.29 KiB	Available
transactions.csv	11/6/2024, 11:16:35 AM	Hot (Inferred)		Block blob	3.43 KiB	Available

## Databricks Activity (Incremental/Delta Processing)

### Step 1: Created Azure DataBricks

- Azure DataBricks name: “my-rg-databrick”
- Launch workspace

### Step 2: Creating a cluster

- Select compute
- Select - create compute

**Compute**

**Rahul Gupta's Cluster**

Policy: Unrestricted

Access mode: No isolation shared

Performance: Databricks Runtime Version 15.4 LTS (includes Apache Spark 3.5.0, Scala 2.12)

Use Photon Acceleration: ☒

Node type: Standard\_DS3\_v2 (14 GB Memory, 4 Cores)

Terminate after: 120 minutes of inactivity

Tags: No custom tags

Summary:

- 1 Driver, 14 GB Memory, 4 Cores
- Runtime: 15.4.x-scala2.12
- Photon, Standard\_DS3\_v2, 1.5 DBU/h

### Step 3: Create a notebook:

- Select workspace - Create - folder
- Folder name “my-rg-dbfolder”
- In this folder creating notebook

- Select - Create - Notebook
- Rename the notebook as “Incremental-Delta Processing”

Code 1:

```
spark.conf.set(
    "fs.azure.account.key.mystoragep3.dfs.core.windows.net",
    "uGFLgPdIWGqegoV7j+dG1+szuKQQJDT/800sK5YxwDPaPmx+8PSitHVB4dlserdt2Y7pZ8/F9T7D+ASTf9HWmQ==")
```

Code 2:

```
display(dbutils.fs.ls("abfss://bronze@mystoragep3.dfs.core.windows.net"))
```

Free trial ends in 14 days. [Upgrade to Premium](#) in Azure Portal

12:01 PM (<1s) 1

```
spark.conf.set(
    "fs.azure.account.key.mystoragep1.dfs.core.windows.net",
    "uGFLgPdIWGqegoV7j+dG1+szuKQQJDT/800sK5YxwDPaPmx+8PSitHVB4dlserdt2Y7pZ8/F9T7D+ASTf9HWmQ==")
```

4 minutes ago (8s) 2

```
display(dbutils.fs.ls("abfss://bronze@mystoragep1.dfs.core.windows.net"))
```

(2) Spark Jobs

	path	name	size	modificationTime
1	abfss://bronze1@mystoragep1.dfs.core.windows.net/accounts.csv	accounts.csv	2331	1730909795000
2	abfss://bronze1@mystoragep1.dfs.core.windows.net/customers.csv	customers.csv	4603	1730909795000
3	abfss://bronze1@mystoragep1.dfs.core.windows.net/loan_payments.c...	loan_payments.c...	2613	1730909795000
4	abfss://bronze1@mystoragep1.dfs.core.windows.net/loans.csv	loans.csv	2340	1730909795000
5	abfss://bronze1@mystoragep1.dfs.core.windows.net/transactions.csv	transactions.csv	3513	1730909795000

5 rows | 7.69 seconds runtime Refreshed 3 minutes ago

Code 3:

## Read Data from Raw (Bronze) Container

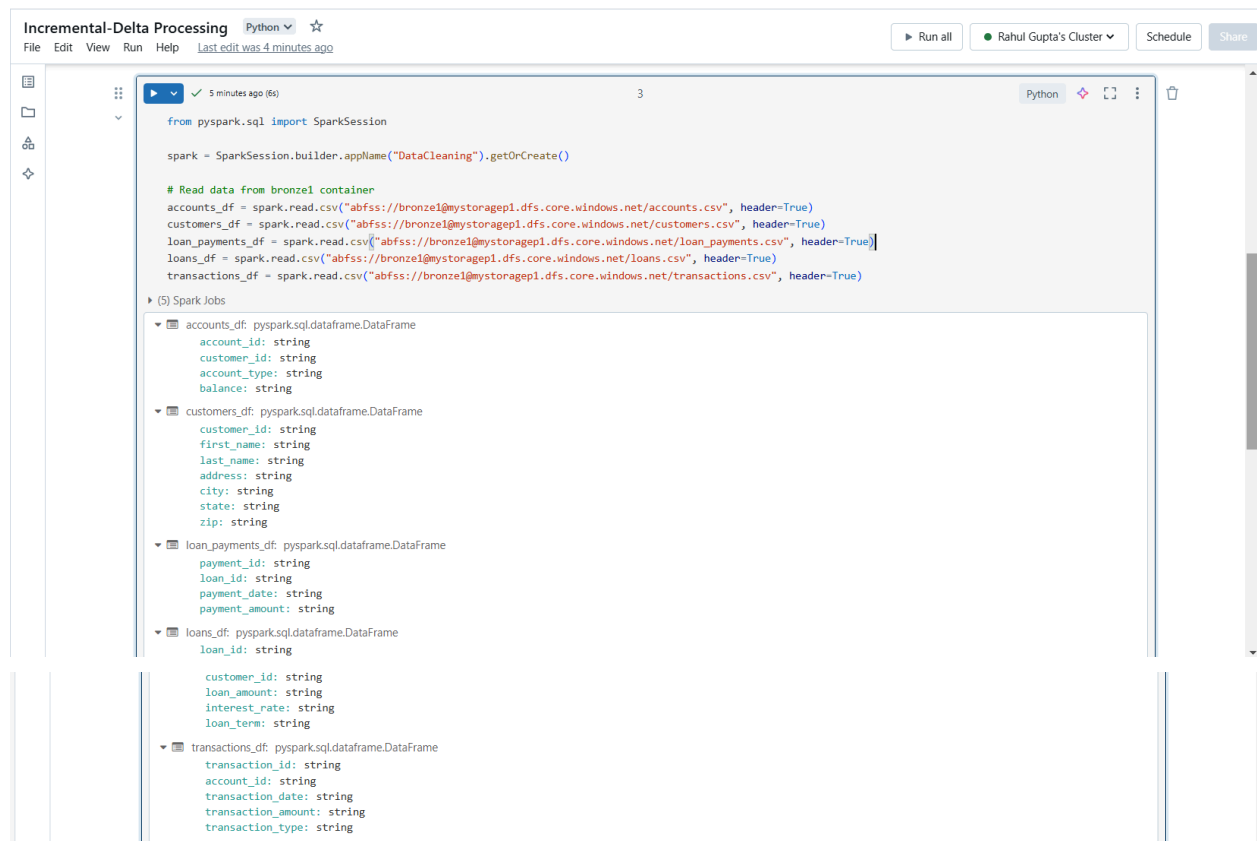
```
from pyspark.sql import SparkSession
```

```
spark = SparkSession.builder.appName("DataCleaning").getOrCreate()
```

```
# Read data from bronze container
```

```
accounts_df =
spark.read.csv("abfss://bronze@mystoragep3.dfs.core.windows.net/accounts.csv",
header=True)
customers_df =
spark.read.csv("abfss://bronze@mystoragep3.dfs.core.windows.net/customers.csv",
header=True)
loan_payments_df =
spark.read.csv("abfss://bronze@mystoragep3.dfs.core.windows.net/loan_payments.csv", header=True)
loans_df =
spark.read.csv("abfss://bronze@mystoragep3.dfs.core.windows.net/loans.csv",
header=True)
transactions_df =
spark.read.csv("abfss://bronze@mystoragep3.dfs.core.windows.net/transactions.csv", header=True)

accounts_df.show(5)
customers_df.show(5)
loan_payments_df.show(5)
loans_df.show(5)
transactions_df.show(5)
```



## Data Cleaning and Data Transformation:

Implement logic to handle null values, remove duplicates, and clean data.

- Remove null values

```
accounts_df = accounts_df.dropna()

customers_df = customers_df.dropna()

loan_payments_df = loan_payments_df.dropna()

loans_df = loans_df.dropna()

transactions_df = transactions_df.dropna()
```

- Remove Duplicates

```
accounts_df = accounts_df.dropDuplicates()
```

```
customers_df = customers_df.dropDuplicates()
loan_payments_df = loan_payments_df.dropDuplicates()
loans_df = loans_df.dropDuplicates()
transactions_df = transactions_df.dropDuplicates()
```

- Renaming the column name from zip to postal\_code from customer file.

```
customers_df = customers_df.withColumnRenamed("zip", "postal_code")
```

- Change data types for **customer** table Converting 'customer\_id' and 'postal\_code' to Integer

```
from pyspark.sql import functions as F

# Change data types for customer table Converting 'customer_id' and
# 'postal_code' to Integer
customers_df = (customers_df
    .withColumn("customer_id", F.col("customer_id").cast("int"))
    .withColumn("zip", F.col("zip").cast("int"))
)

# Display the updated schema to verify changes
customers_df.printSchema()

# Display the updated schema to verify changes
customers_df.show(5)
```

- Change data types for **accounts** table Converting 'customer\_id', 'account\_id' to integer and 'balance' to double

```
accounts_df = (accounts_df
    .withColumn("account_id", F.col("account_id").cast("int"))
    .withColumn("customer_id", F.col("customer_id").cast("int"))
    .withColumn("balance", F.col("balance").cast("double")) # Change to
    "float" if desired
)
```

```
# Display the updated schema to verify changes
accounts_df.printSchema()
```

```
# Display the updated schema to verify changes
accounts_df.show(5)
```

- Change data types for **loan\_payments** table converting 'payment\_id', 'loan\_id' to Int and 'payment\_date' to date and 'payment\_amount' to double.

```
loan_payments_df = (loan_payments_df
    .withColumn("payment_id", F.col("payment_id").cast("int"))
    .withColumn("loan_id", F.col("loan_id").cast("int"))
    .withColumn("payment_date", F.to_date(F.col("payment_date"), "yyyy-MM-dd"))
# Adjust date format as needed
    .withColumn("payment_amount", F.col("payment_amount").cast("double")) #
Use "float" if desired
)
```

```
# Display the updated schema to verify changes
```

```
loan_payments_df.printSchema()
```

```
# Display the updated schema to verify changes
```

```
loan_payments_df.show(5)
```

- Change data types for **loans** table converting 'loan\_id', 'customer\_id' and 'loan\_term' to Int and 'loan\_amount', 'interest\_rate' to double

```
loans_df = (loans_df
    .withColumn("loan_id", F.col("loan_id").cast("int"))
    .withColumn("customer_id", F.col("customer_id").cast("int"))
    .withColumn("loan_amount", F.col("loan_amount").cast("double"))
    .withColumn("interest_rate", F.col("interest_rate").cast("double"))
    .withColumn("loan_term", F.col("loan_term").cast("int"))
)
```

```
# Display the updated schema to verify changes
```



```
loans_df.printSchema()
```

```
# Display the updated schema to verify changes
```

```
loans_df.show(5)
```

- Change data types for **transactions** table converting 'transaction\_id', 'account\_id' and 'transaction\_date' to Date and 'transaction\_amount' to double

```
transactions_df = (transactions_df
    .withColumn("transaction_id", F.col("transaction_id").cast("int"))
    .withColumn("account_id", F.col("account_id").cast("int"))
    .withColumn("transaction_date", F.to_date(F.col("transaction_date"),
"yyyy-MM-dd"))
    .withColumn("transaction_amount",
F.col("transaction_amount").cast("double"))
)
```

```
# Display the updated schema to verify changes
```

```
transactions_df.printSchema()
```

```
# Display the updated schema to verify changes
```

```
transactions_df.show(5)
```

- Writing the path to the silver container:

```
# Define the paths to the Silver container
```

```
silver_accounts =
```

```
"abfss://silver@mystoragep3.dfs.core.windows.net/delta/accounts_delta"
```

```
silver_customers =
```

```
"abfss://silver@mystoragep3.dfs.core.windows.net/delta/customers_delta"
```

```
silver_loan_payments =
```

```
"abfss://silver@mystoragep3.dfs.core.windows.net/delta/loan_payments_delta"
```

```

silver_loans =
"abfss://silver@mystoragep3.dfs.core.windows.net/delta/loans_delta"

silver_transactions =
"abfss://silver@mystoragep3.dfs.core.windows.net/delta/transactions_delta"

```

- Writing the cleaned data back to the silver container:

**# Write the cleaned data back to the Silver container**

```

accounts_df.write.format("delta").mode("overwrite").save(silver_accounts)

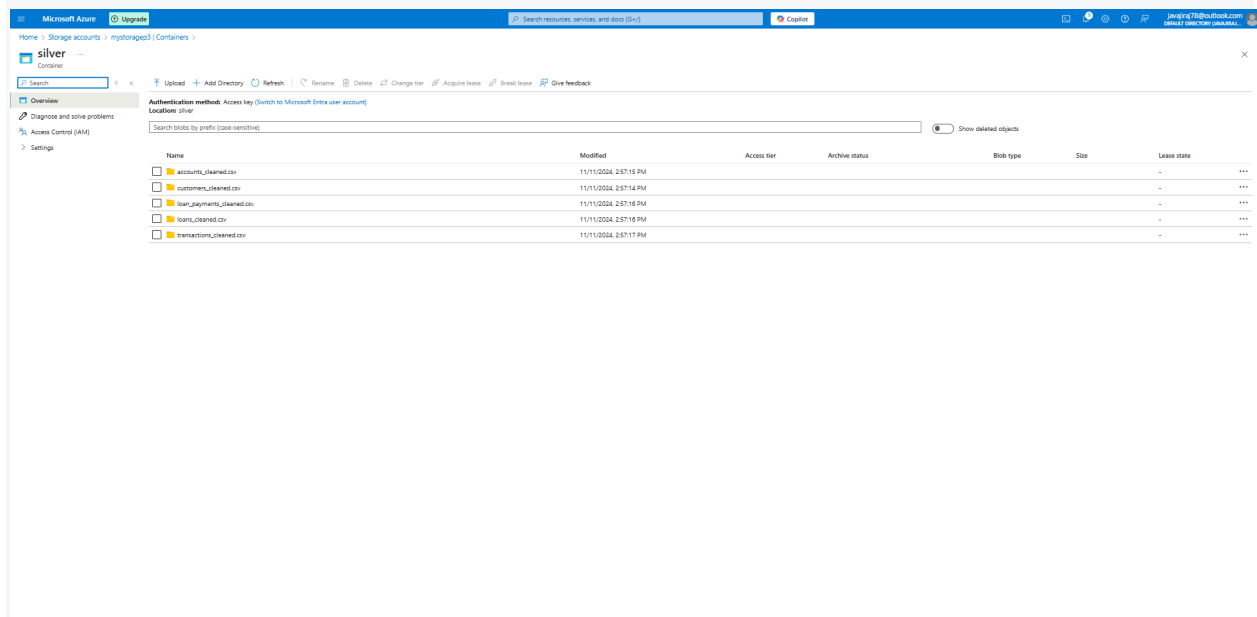
customers_df.write.format("delta").mode("overwrite").save(silver_customers)

loan_payments_df.write.format("delta").mode("overwrite").save(silver_loan_payments)

loans_df.write.format("delta").mode("overwrite").save(silver_loans)

transactions_df.write.format("delta").mode("overwrite").save(silver_transactions)

```



# Databricks Activity (ETL Processing)

Step 1: Create a notebook:

- In the same Folder name "my-rg-dbfolder"
- In this folder creating new notebook
- Select - Create - Notebook
- Rename the notebook as "ETL Processing"

Step 2: Read Data from Silver Container:

```
spark.conf.set (
    "fs.azure.account.key.mystorage3.dfs.core.windows.net",

"/mskNq6Zye/YglZVl0L9BIG0KNhXMrg59eX49h0Q+ekAw9BuggYKDX8CuhM8o+qMsK1BzNyA83mP+A
StbAh9Vw==")

# Read Data from Silver Container:
accounts_df =
spark.read.format("delta").load("abfss://silver@mystorage3.dfs.core.windows.ne
t/delta/accounts_delta")
customers_df =
spark.read.format("delta").load("abfss://silver@mystorage3.dfs.core.windows.ne
t/delta/customers_delta")
loan_payments_df =
spark.read.format("delta").load("abfss://silver@mystorage3.dfs.core.windows.ne
t/delta/loan_payments_delta")
loans_df =
spark.read.format("delta").load("abfss://silver@mystorage3.dfs.core.windows.ne
t/delta/loans_delta")
transactions_df =
spark.read.format("delta").load("abfss://silver@mystorage3.dfs.core.windows.ne
t/delta/transactions_delta")
```

### Step 3: Apply Business Logic for Transformation:

Write a query to calculate the total balance across all accounts for each customer, ensuring that all columns from the accounts and customers tables are selected and included

```
from pyspark.sql import functions as F

# Cast balance to Double to ensure correct aggregation

accounts_df = accounts_df.withColumn("balance",
F.col("balance").cast("double"))

# Join the accounts and customers DataFrames on customer_id

customer_accounts_df = accounts_df.join(customers_df, on="customer_id",
how="inner")

# Calculate the total balance for each customer, include all customer columns,
and keep individual account columns

result_df = (customer_accounts_df

    .groupBy("customer_id", "first_name", "last_name", "address", "city",
"state", "zip")

    .agg(

        F.sum("balance").alias("total_balance")

    )

    .join(accounts_df.select("customer_id", "account_id", "account_type",
"balance"), on="customer_id", how="left")

)
```

```
# Show the result
```

```
result_df.show(truncate=False)
```

Step 4: Data Loading: Store the transformed data in the Refined(gold) container of your data lake.

```
# Define the path for gold container
```

```
gold_delta = "abfss://gold@mystoragep3.dfs.core.windows.net/delta/gold_delta"
```

```
# Save the DataFrame in Delta format, overwriting if it exists
```

```
result_df.write.format("delta").mode("overwrite").save(gold_delta)
```

## Azure Synapse Analytics

Step 1: Create azure synapse analytics

### Basics

- Resource group - Create New - "synapseresource"
- Workspace name: "sa-my-rg"
- Region: "Central US"
- Select Data Lake Storage Gen2: "From Subscription"
- Account name - create new - "mystoragep3"
- File system name - create new - "silver"
- Next Security.

Home > Resource groups > my-rg > Marketplace

## Create Synapse workspace

\* Basics \* Security Networking Tags Review + create

Create a Synapse workspace to develop an enterprise analytics solution in just a few clicks.

**Project details**

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all of your resources.

Subscription \*

☒ The Synapse and SQL resource providers are now registered with this subscription.

Resource group \*

[Create new](#)

Managed resource group

**Workspace details**

Name your workspace, select a location, and choose a primary Data Lake Storage Gen2 file system to serve as the default location for logs and job output.

Workspace name \*

Region \*

Select Data Lake Storage Gen2 \* ☒ From subscription ☐ Manually via URL

Account name \*

[Create new](#)

File system name \*

[Create new](#)

☒ Assign myself the Storage Blob Data Contributor role on the Data Lake Storage Gen2 account to interactively query it in the workspace.

☒ We will automatically grant the workspace identity data access to the specified Data Lake Storage Gen2 account, using the [Storage Blob Data Contributor](#) role. To enable other users to use this storage account after you create your workspace, perform these tasks:

- Assign other users to the **Contributor** role on workspace
- Assign other users the appropriate **Storage Blob Data Contributor** role using Synapse Studio
- Assign yourself and other users to the **Storage Blob Data Contributor** role on the storage account

[Learn more](#)

[Review + create](#) [Previous](#) [Next: Security >](#)

<https://portal.azure.com/#>

\* Basics \* Security Networking Tags Review + create

Configure security options for your workspace.

### Authentication

Choose the authentication method for access to workspace resources such as SQL pools. The authentication method can be changed later on. [Learn more](#)

Authentication method ☒ Use both local and Microsoft Entra ID authentication ☐ Use only Microsoft Entra ID authentication

SQL Server admin login \*

SQL Password  ☒

Confirm password  ☒

- Next Networking (no changes - keep it default)
- Next Tags (no changes - keep it default)
- Next Review + create

### Step 2: Launch Synapse studio

- Develop - SQL script - copy paste the script -

### -- Example for creating an external data source for the Silver container

```
CREATE EXTERNAL DATA SOURCE SilverDataSource
WITH (
    LOCATION = 'abfss://silver@strgacc2831.dfs.core.windows.net'
);
```

- Data + sql database - serverless - name: database-my-rg1
- For the above script - run the script.
- Run the 2nd script:

**-- Example for creating an external data source for the Gold container**

```
CREATE EXTERNAL DATA SOURCE GoldDataSource
WITH (
    LOCATION = 'abfss://silver@strgacc2831.dfs.core.windows.net'
);
```

- Run the 3rd Script:

**-- Example for creating a CSV file format**

```
CREATE EXTERNAL FILE FORMAT CsvFileFormat
WITH (
    FORMAT_TYPE = DELIMITEDTEXT,
    FORMAT_OPTIONS (FIELD_TERMINATOR = ',', STRING_DELIMITER = '"', FIRST_ROW = 2)
);
```

- 
- Select - Linked - Select : "sa-my-rg" - You will be able to see the lists of containers which is already created.
- Select - gold - Right click on select : "customer\_account\_summary" select : new sql script - create external table
- Continue - External table name: dbo.customer\_account\_summary1

### New external table

Select target database

[Learn more](#)

Select SQL pool\* ⓘ

✓ Built-in
⌵
↺

Select a database\* ⓘ

database-my-rg1
⌵

External table name

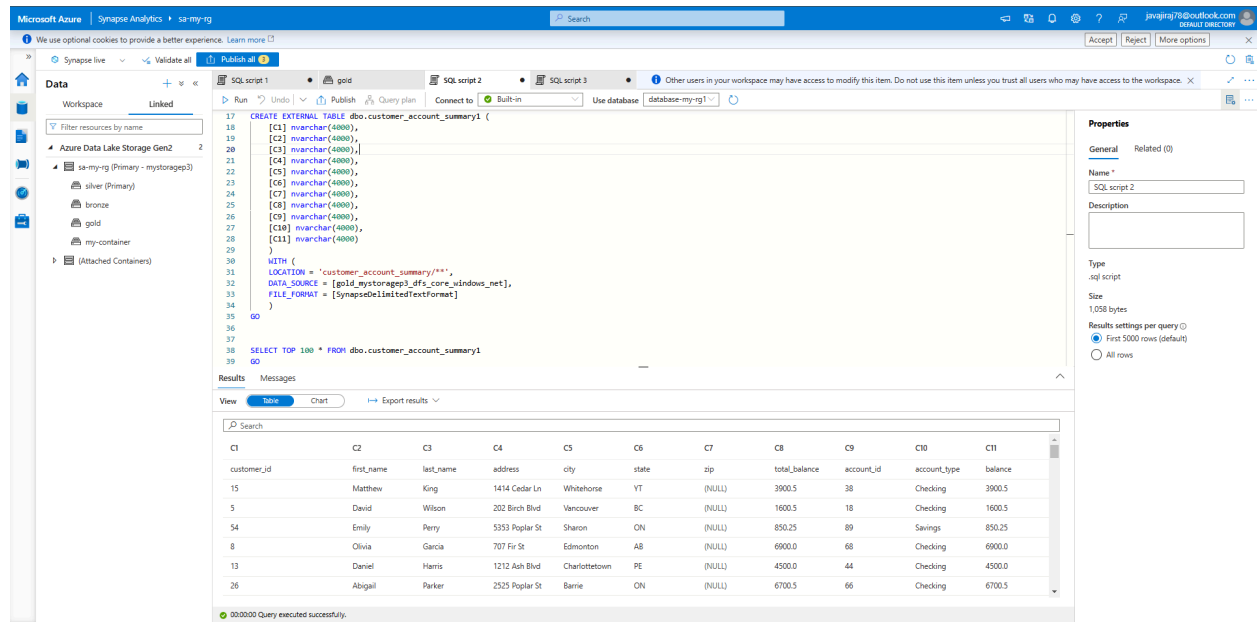
dbo.customer\_account\_summary1

Create external table

☐ Automatically
 ☒ Using SQL script

ⓘ This will generate a SQL script and you will be required to run the SQL script.

- Open script - Run the entire script.



The screenshot shows the Microsoft Azure Synapse Analytics interface. The left sidebar displays the workspace structure, including 'Azure Data Lake Storage Gen2' and 'sa-my-rg (Primary - mystorage1)'. The main pane shows a SQL script being executed. The script creates an external table named 'customer\_account\_summary1' and then selects the top 100 rows. The results are displayed in a table view.

```
17 CREATE EXTERNAL TABLE dbo.customer_account_summary1 (  
18 [C1] nvarchar(4000),  
19 [C2] nvarchar(4000),  
20 [C3] nvarchar(4000),  
21 [C4] nvarchar(4000),  
22 [C5] nvarchar(4000),  
23 [C6] nvarchar(4000),  
24 [C7] nvarchar(4000),  
25 [C8] nvarchar(4000),  
26 [C9] nvarchar(4000),  
27 [C10] nvarchar(4000),  
28 [C11] nvarchar(4000)  
29 )  
30 WITH (  
31 LOCATION = 'customer_account_summary/**',  
32 DATA_SOURCE = [gold_mystorage3_dfs_core_windows_net],  
33 FILE_FORMAT = [SynapseDelimitedTextFormat]  
34 )  
35 GO  
36  
37  
38 SELECT TOP 100 * FROM dbo.customer_account_summary1  
39 GO
```

The results table shows the following data:

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11
customer_id	first_name	last_name	address	city	state	zip	total_balance	account_id	account_type	balance
15	Matthew	King	1414 Cedar Ln	Whitehorse	VT	(NULL)	3900.5	38	Checking	3900.5
5	David	Wilson	202 Birch Blvd	Vancouver	BC	(NULL)	1600.5	18	Checking	1600.5
54	Emily	Perry	5353 Poplar St	Sharon	ON	(NULL)	850.25	89	Savings	850.25
8	Olivia	Garcia	707 Fir St	Edmonton	AB	(NULL)	6900.0	68	Checking	6900.0
13	Daniel	Harris	1212 Ash Blvd	Charlottetown	PE	(NULL)	4500.0	44	Checking	4500.0
26	Abigail	Parker	2525 Poplar St	Barrie	ON	(NULL)	6700.5	66	Checking	6700.5

00:00:00 Query executed successfully.