**Bootcamp Project 4**

**Project Title: Incremental Data Loading and Automated Notifications using Fabrics**

**Project Overview**

This project focuses on building a robust, automated data pipeline using Microsoft Fabric to enable incremental data loading from on-premises systems into the Fabric Lakehouse. It leverages the AI Bank Dataset as the source, cleans and transforms the data using Dataflow Gen 1, and loads it into a Fabric Warehouse. The pipeline applies Slowly Changing Dimension (SCD) Type 1 logic through Fabric Notebooks and ensures automated email notifications are triggered upon successful execution. The entire process is scheduled and monitored, with a detailed architecture diagram and documentation capturing each phase—from ingestion to notification setup.

**Tools Required**

Microsoft Fabrics

On-premises Data Gateway

Fabrics lakehouse and warehouse

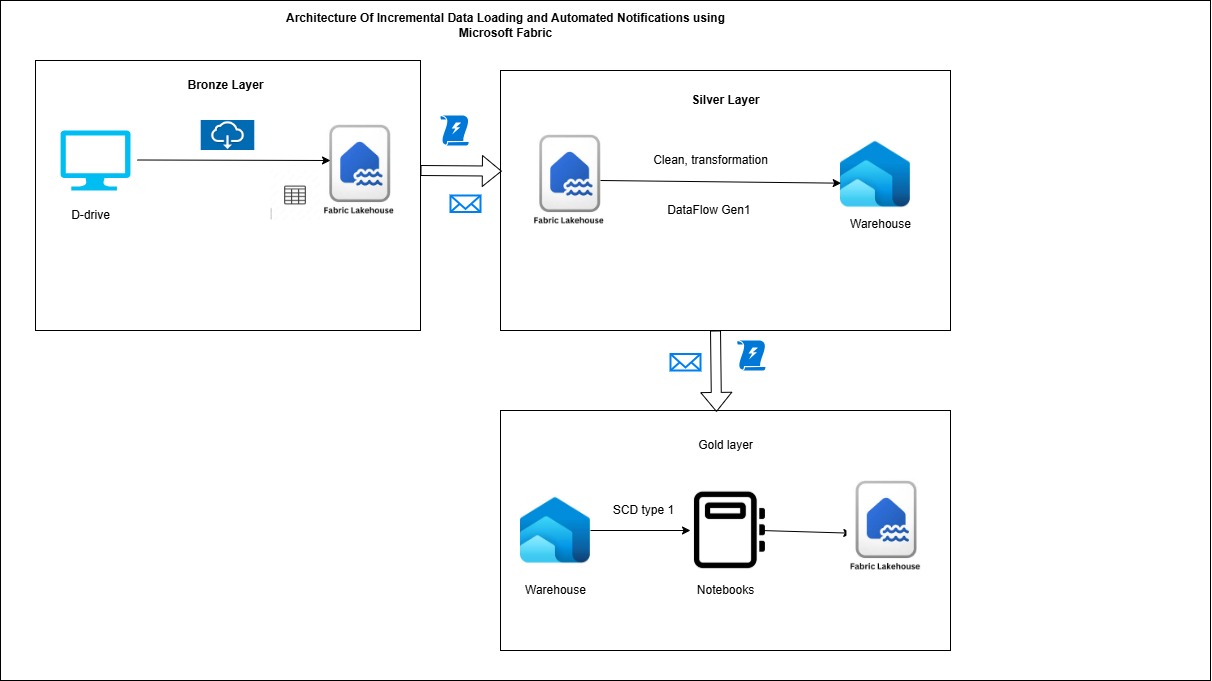
Fabric Dataflow gen 1

Fabrics Notebook

Email Notification

Draw.io

**Architecture**

****

**Bronze Layer – On-premises to Lakehouse.**

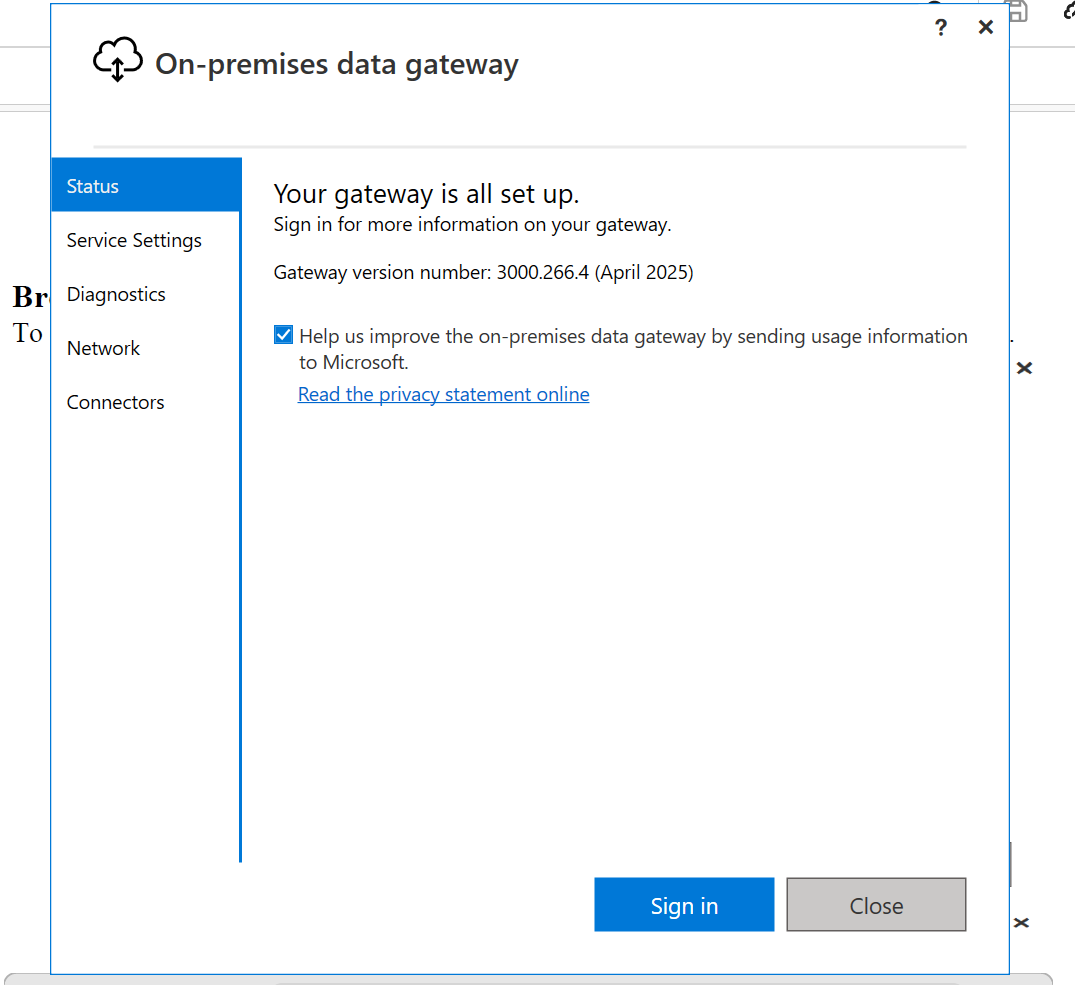
To connect on-premises to pipeline we need to install On-prem gateway

S1-Give fabrics email address and sign in.

S2- Select register a new gateway on this computer option

S3- Give data gateway name, recovery key and configure it.

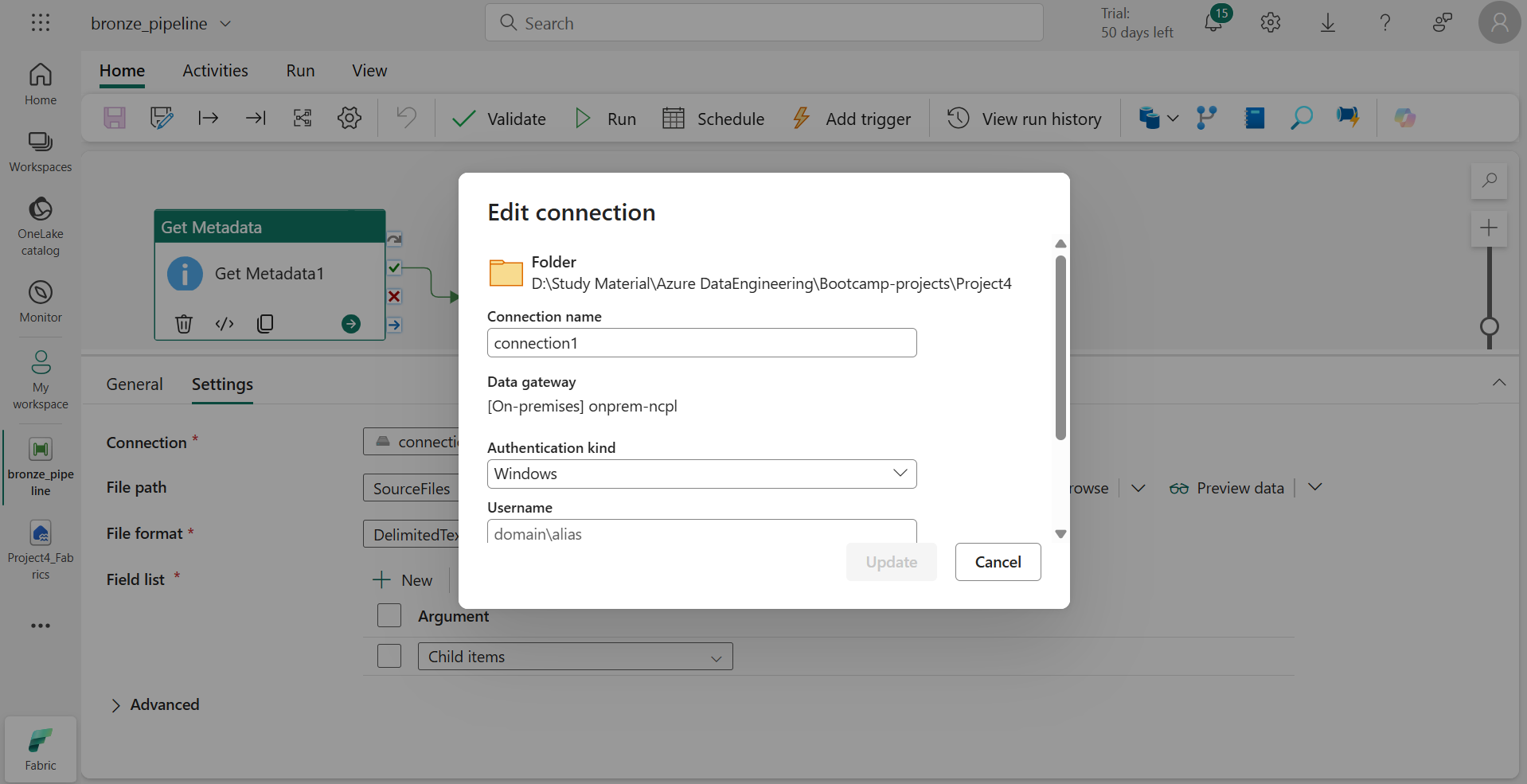
On-prem gateway is created using the above steps.

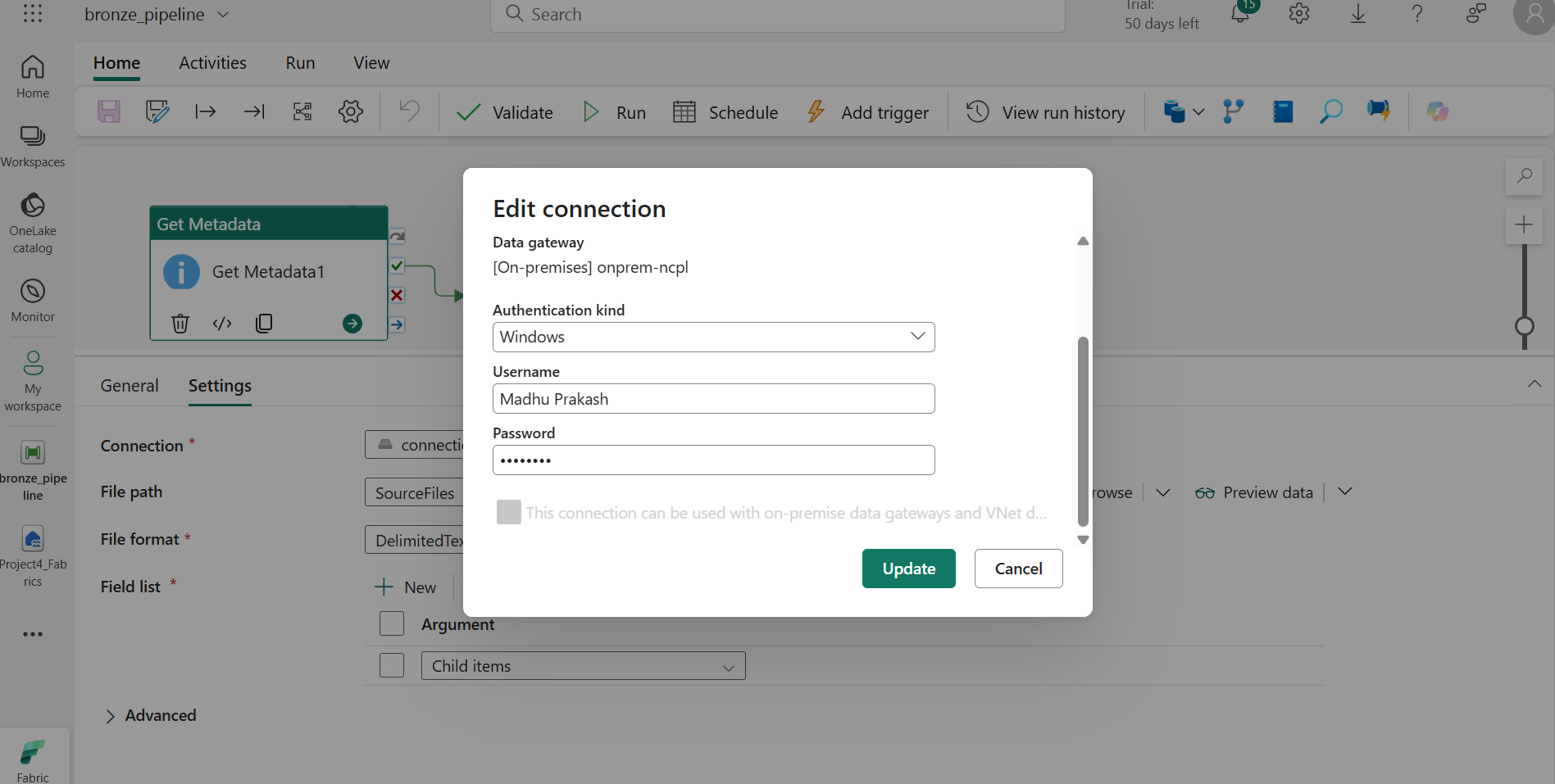


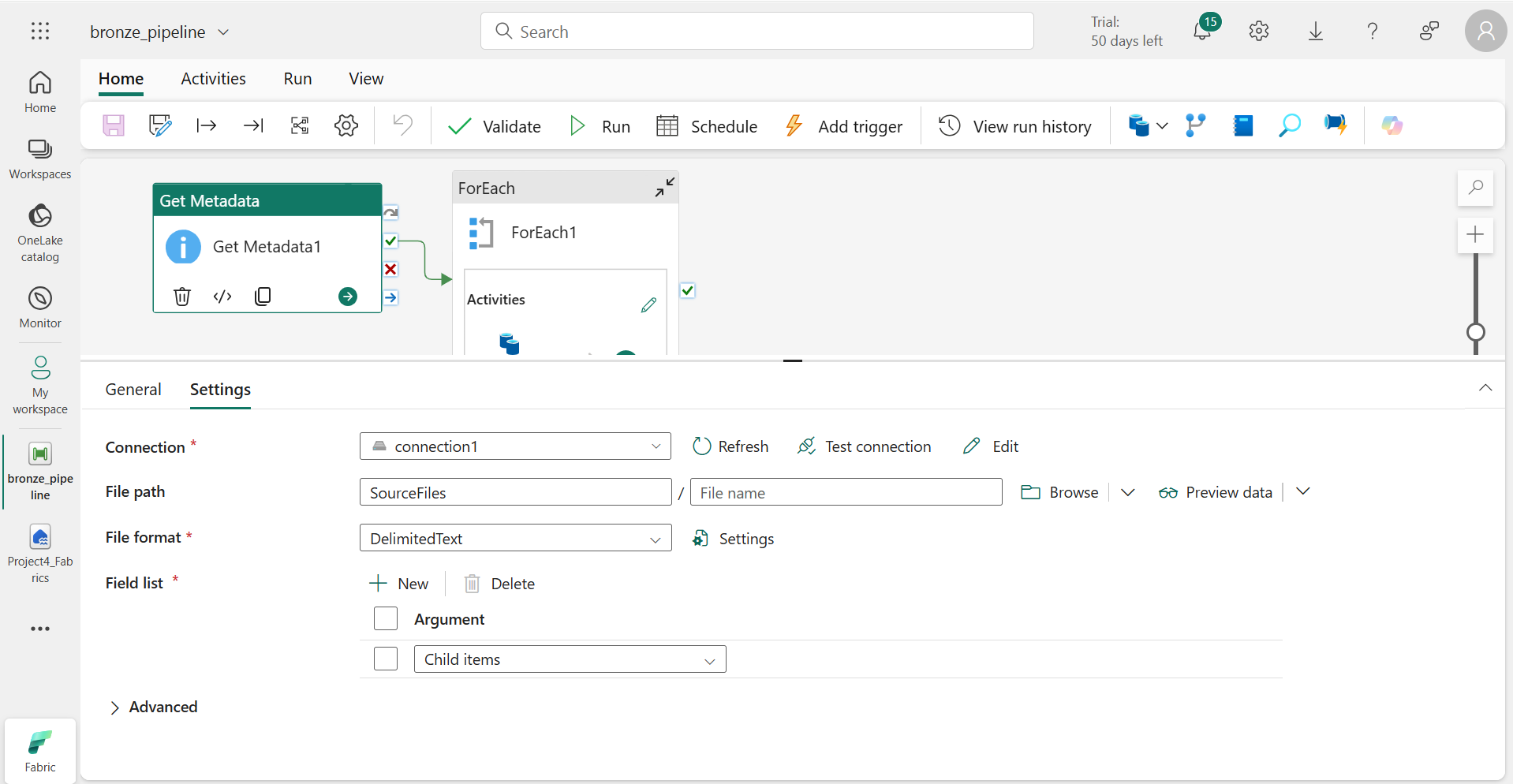
Create a pipeline with copy activity to load data from on-prem to fabric lake house.

Firstly, Drag and drop the Get Metadata activity and

Add connection and configure the gateway, select the file path and add child Items as an argument.

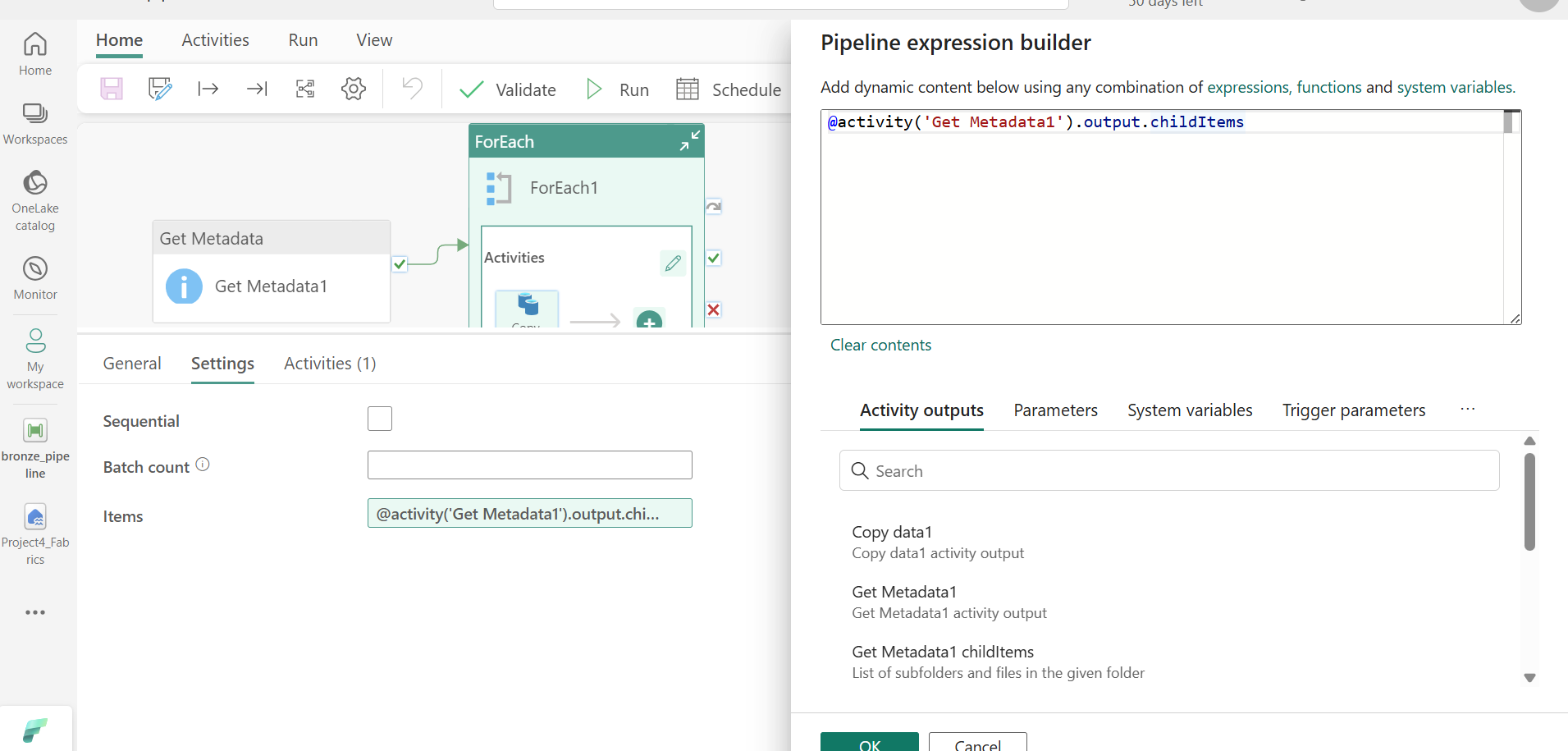






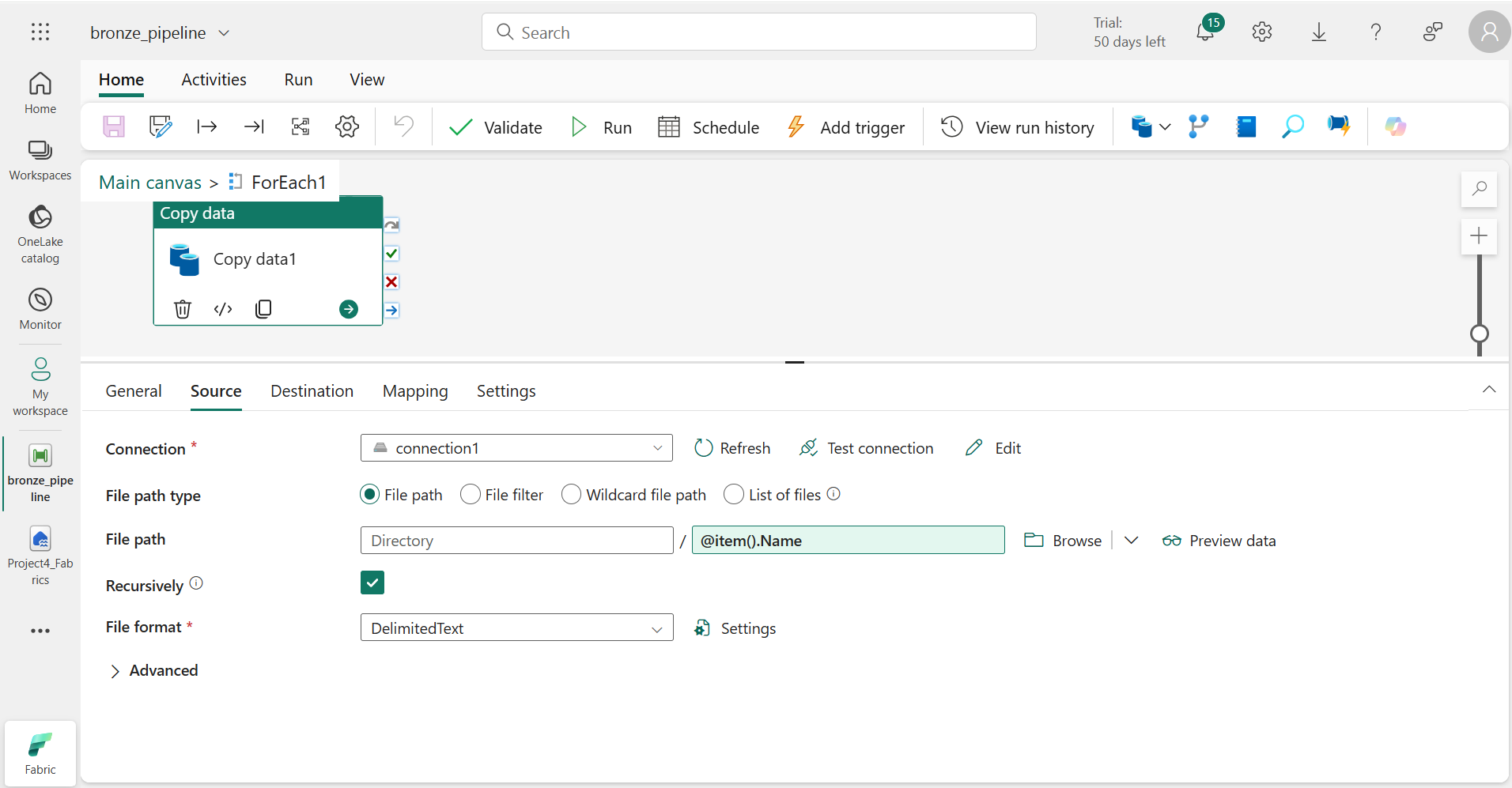
Drap and drop the foreach activity and on success connect GetMetadata with foreach loop

Add **@activity('Get Metadata1').output.childItems under** items.



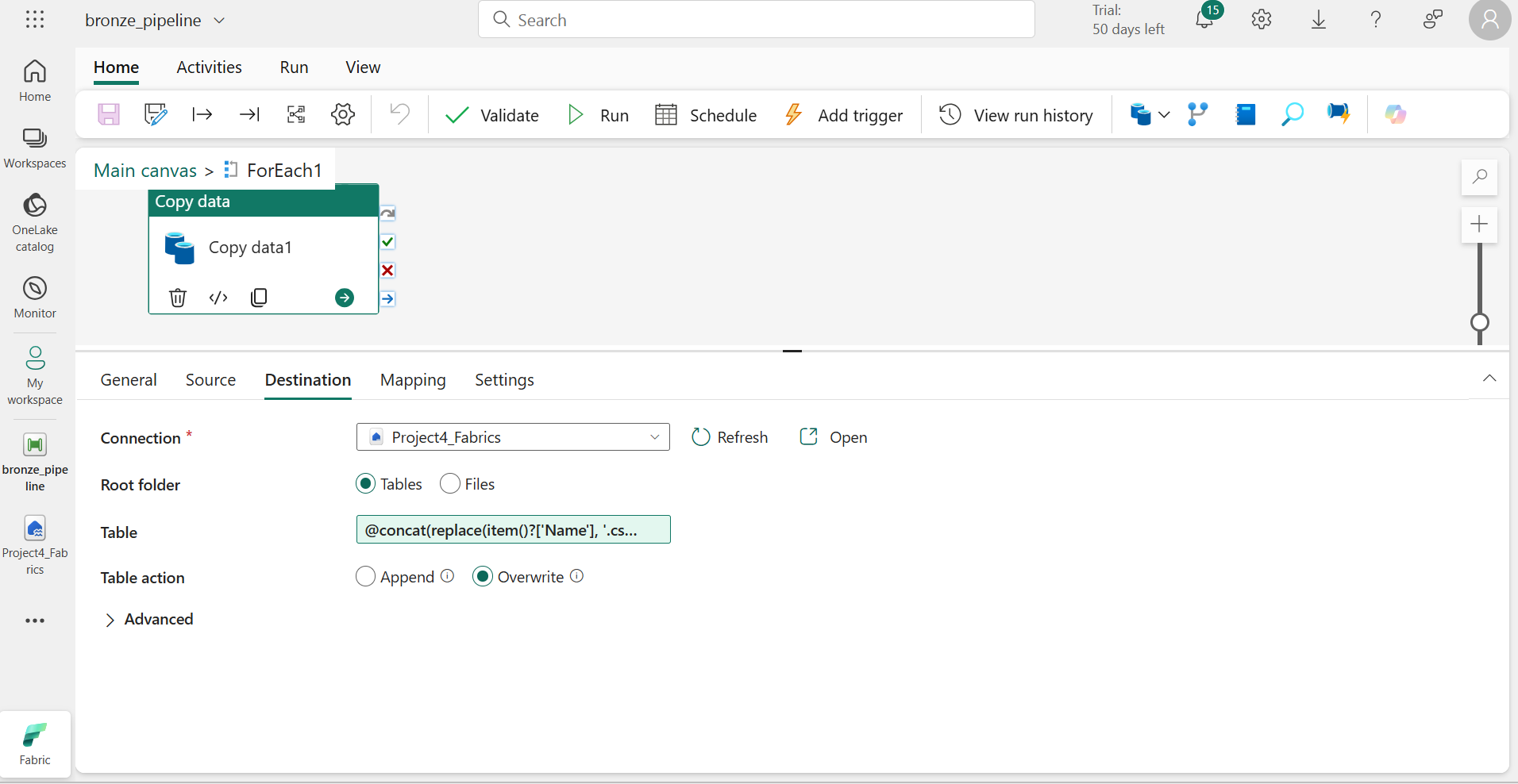
Add copy data inside the foreach loop

Connect the gateway under the connections, select file path add @item().Name and select delimitedText in file format.



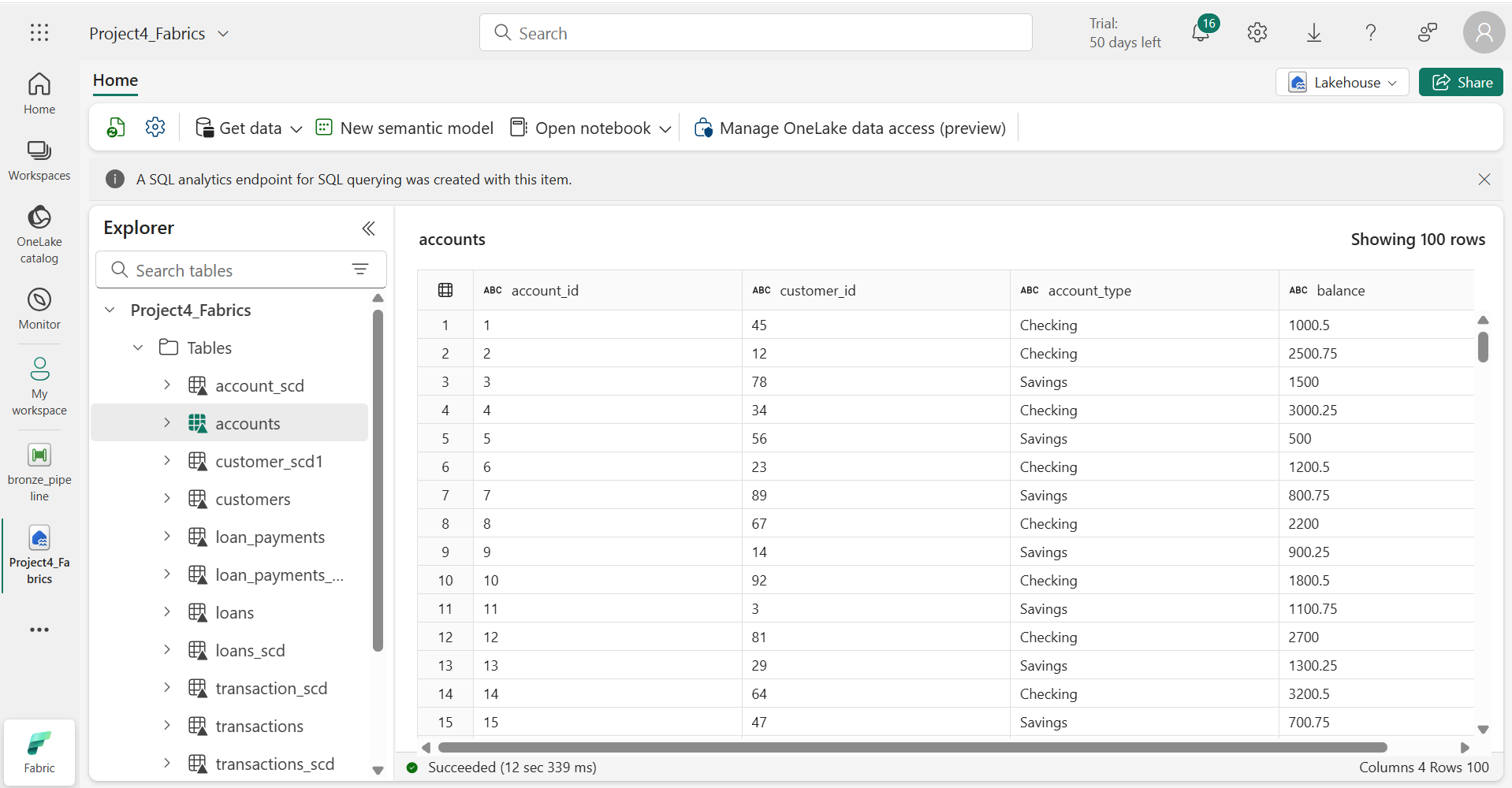
In sink, connect lakehouse in connection, select tables in root folder and add

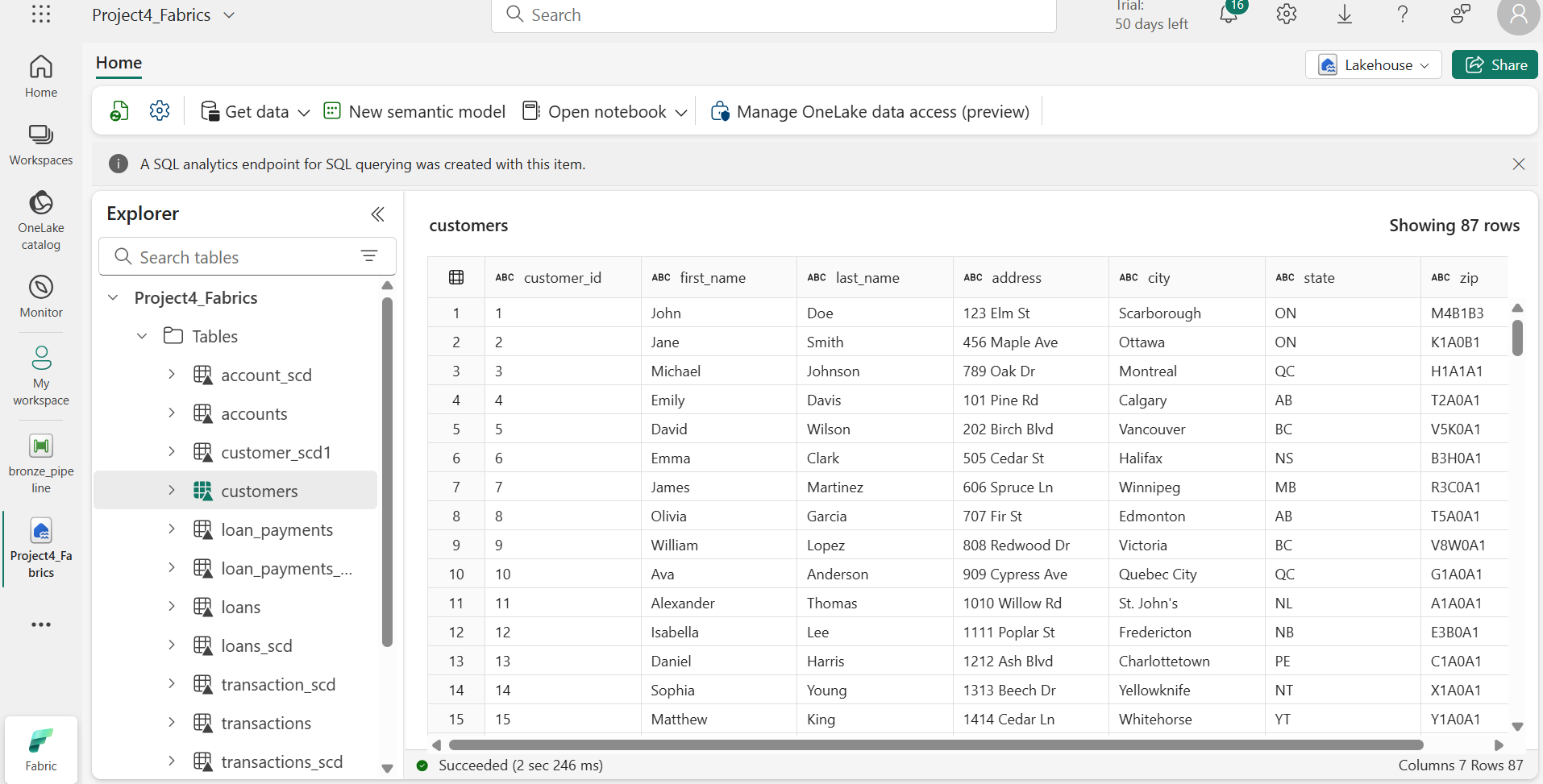
**@concat(replace(item()?['Name'], '.csv', '')).** Check the overwrite.

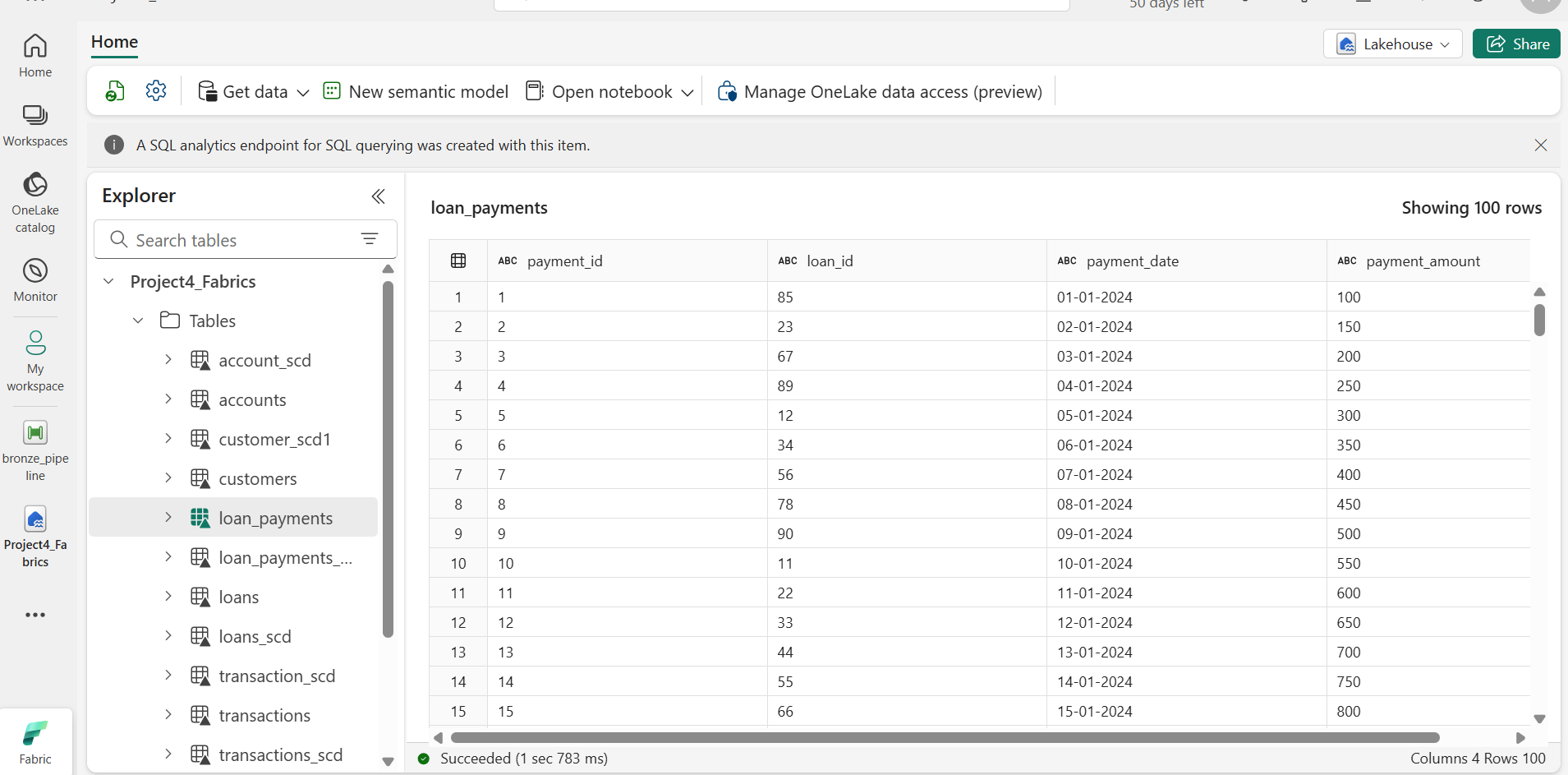


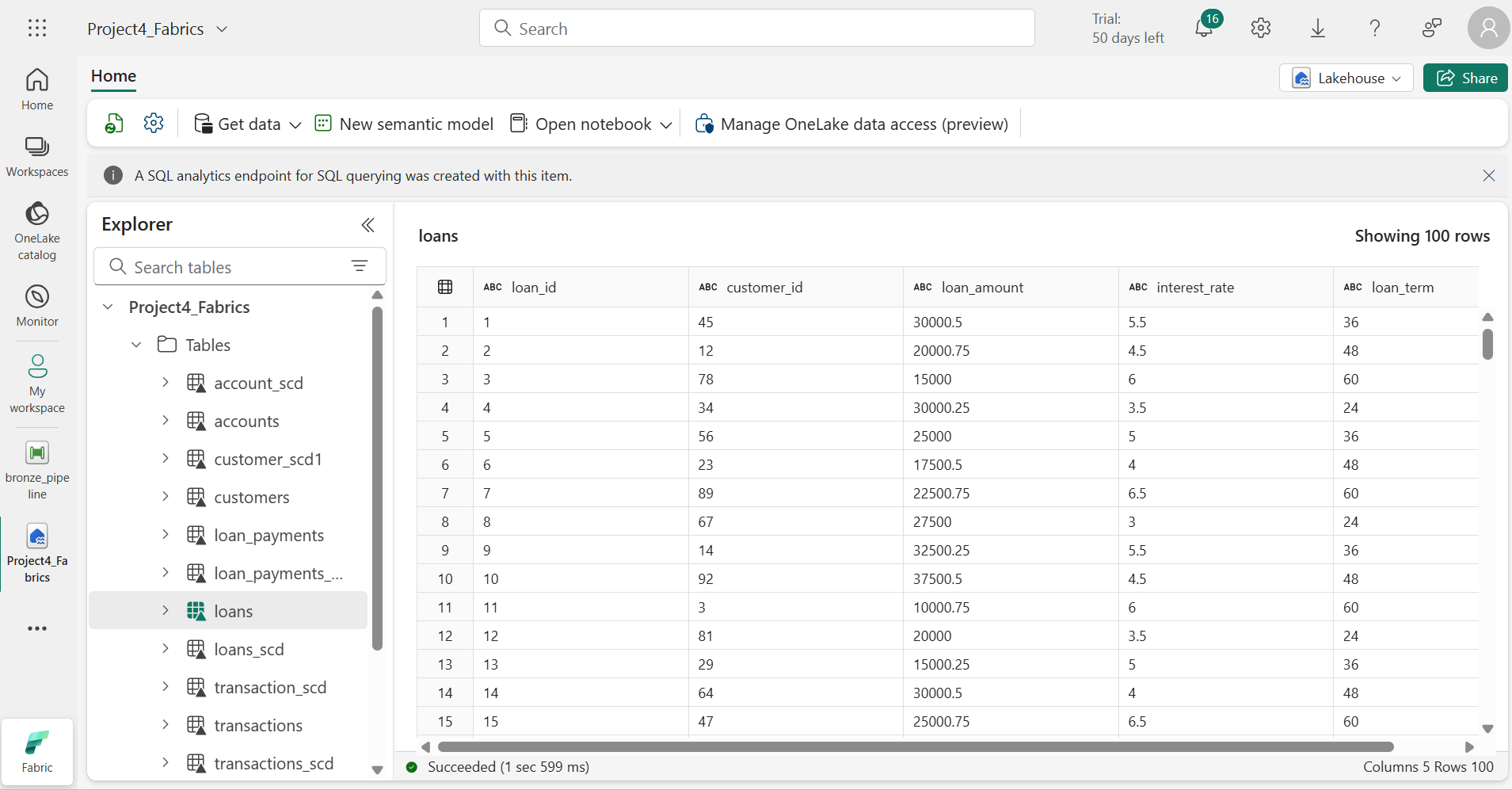


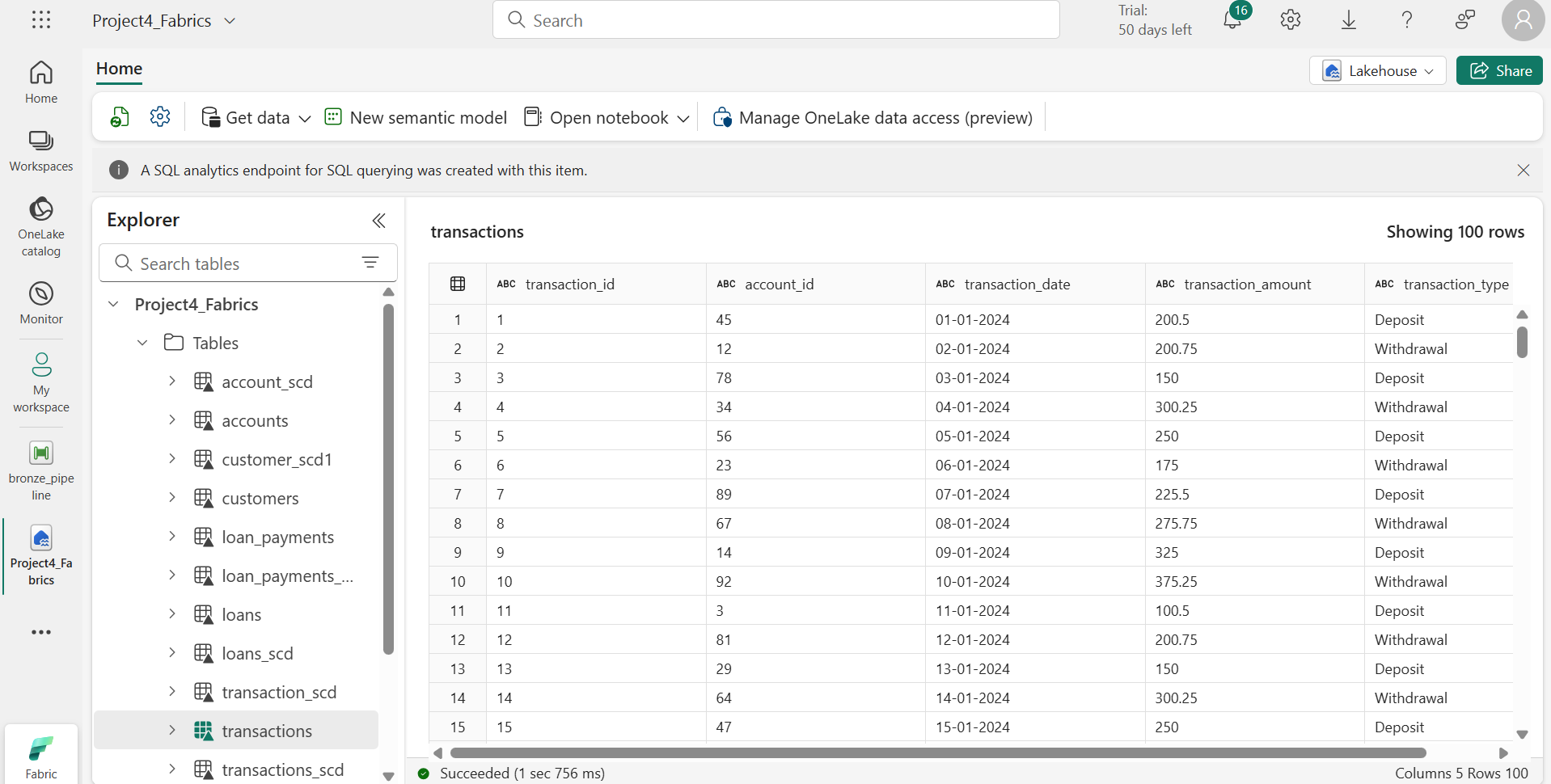
Lakehouse – output





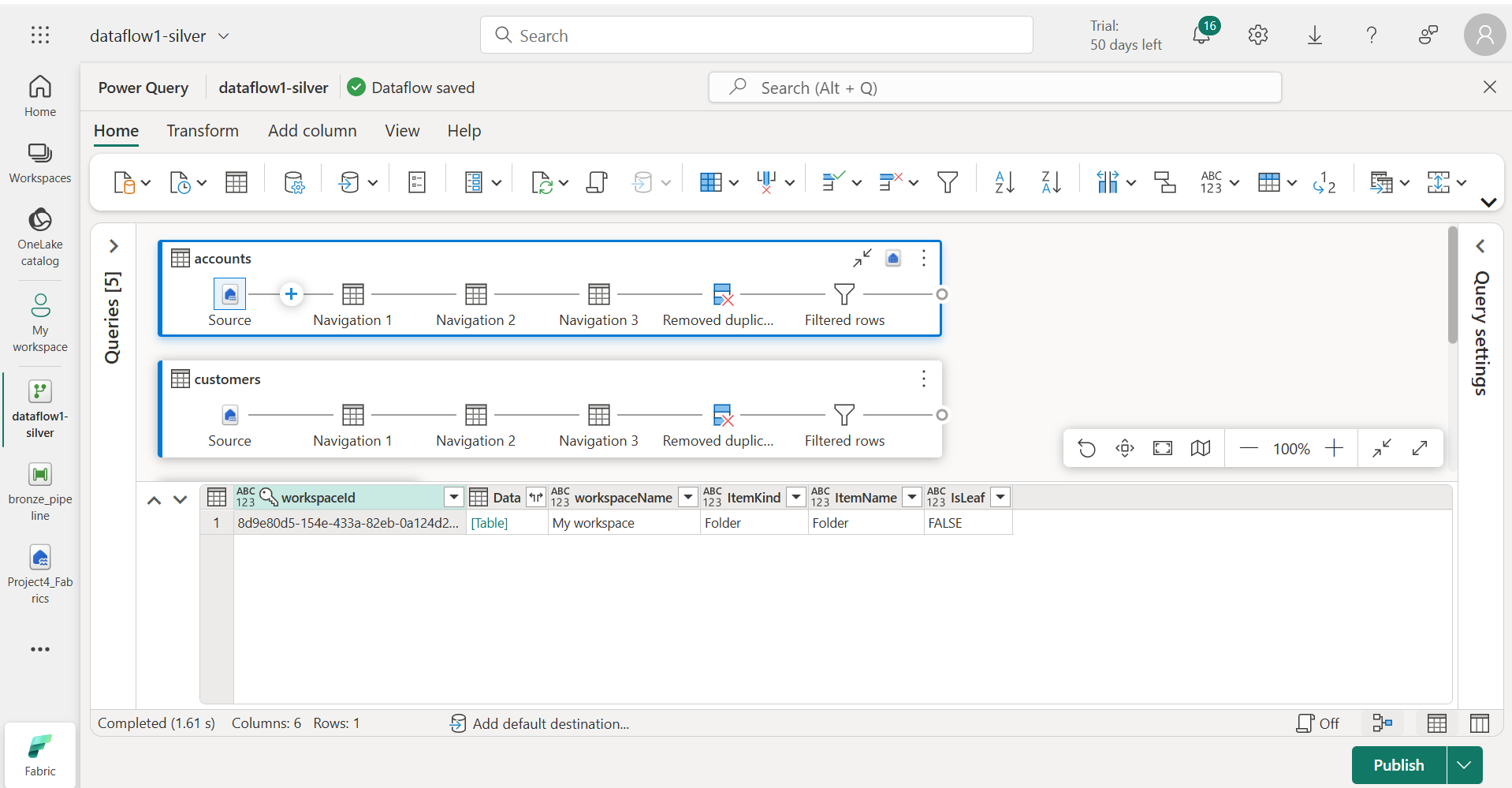




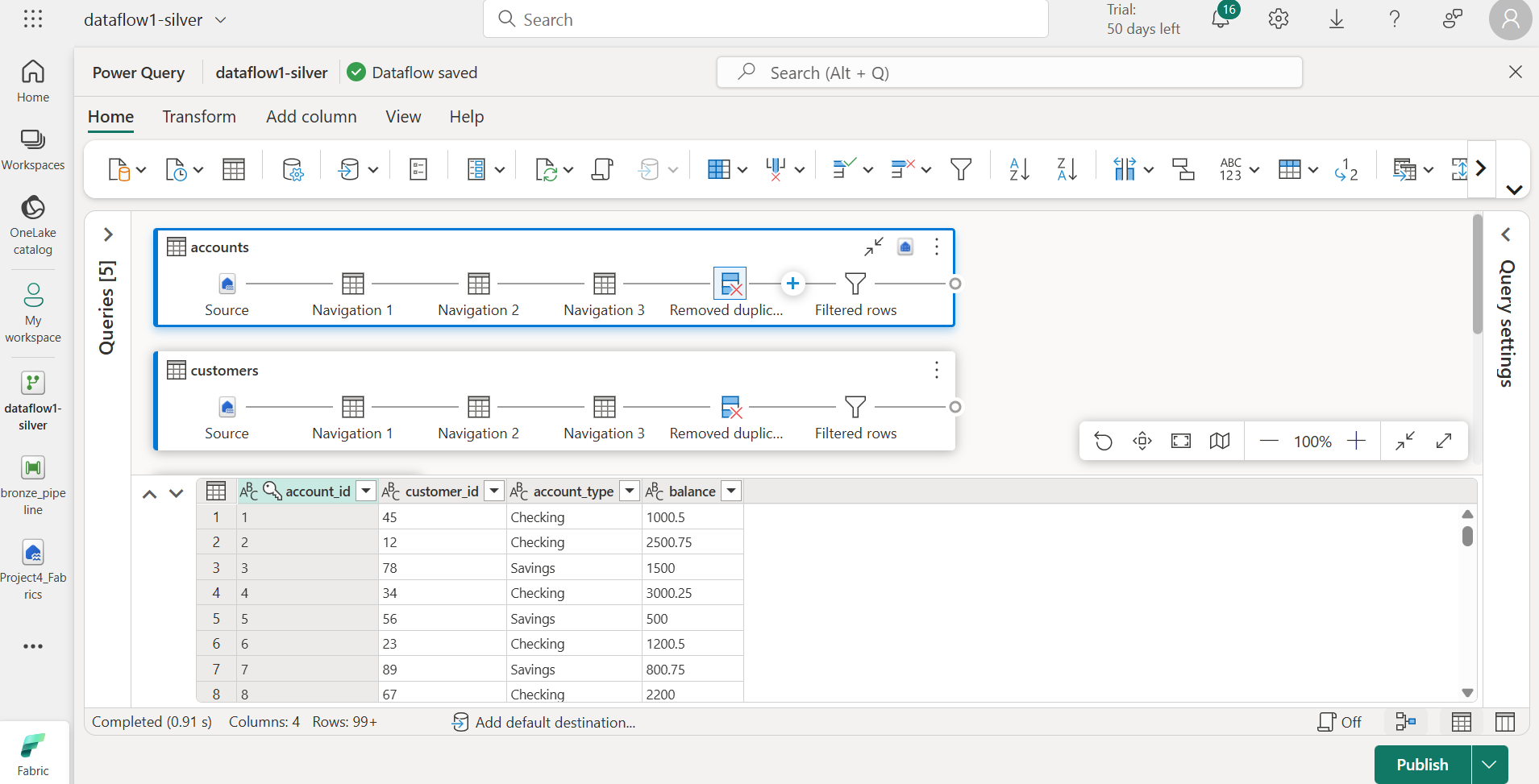


Silver Layer – Lakehouse to Warehouse

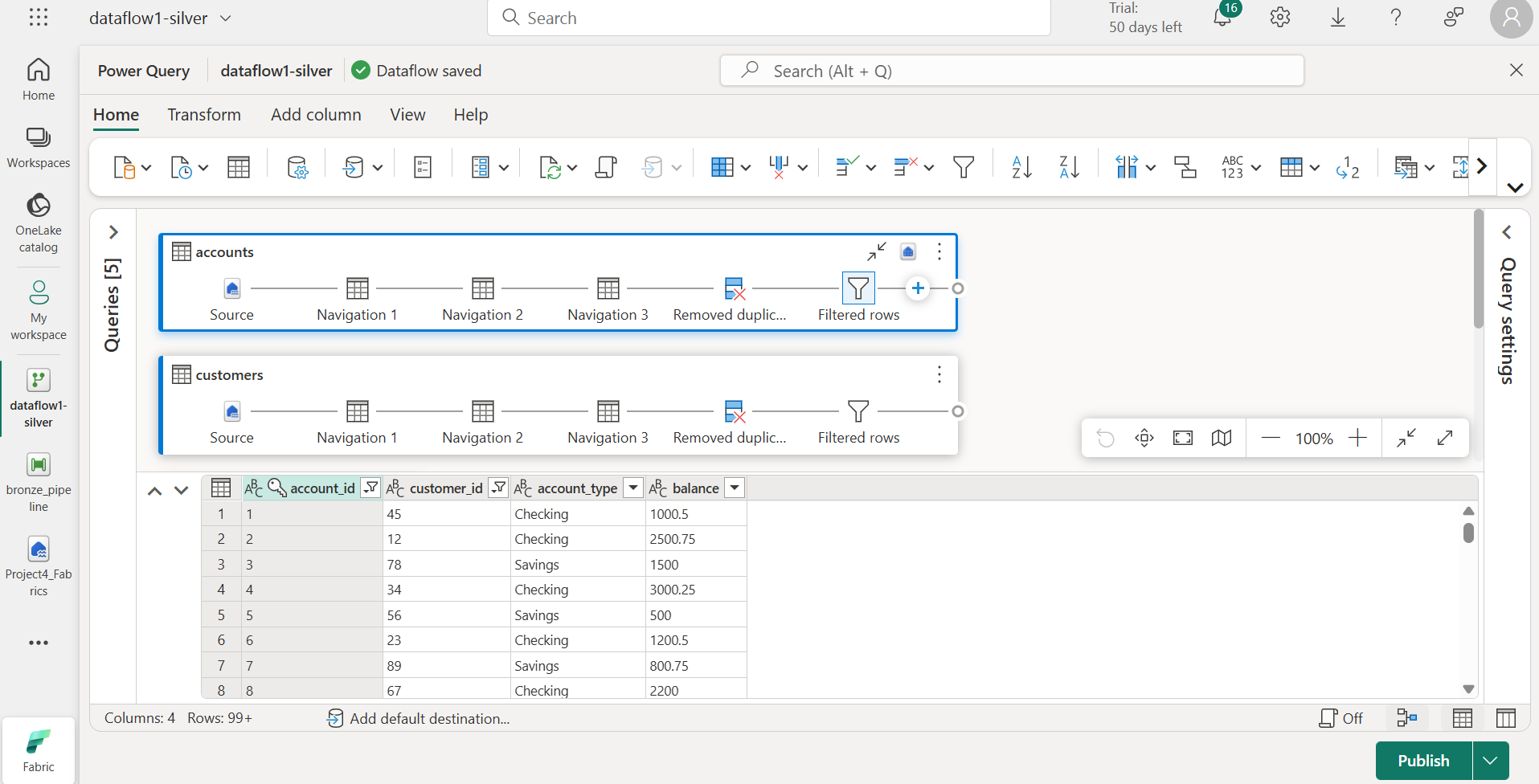
Load data from lakehouse and using data flow gen1 clean and transform data and load it in warehouse.



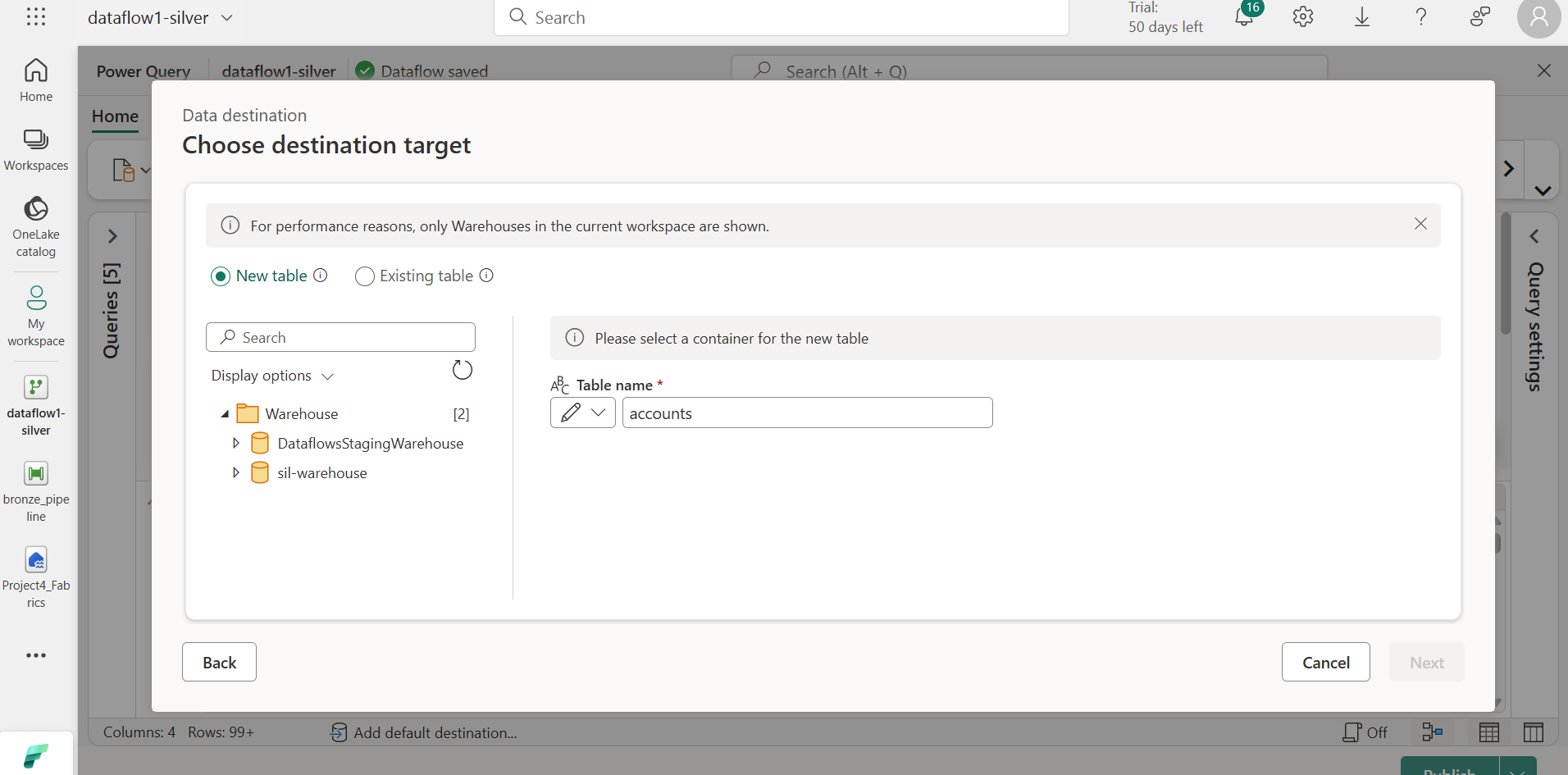
Remove duplicates from the accounts table

****

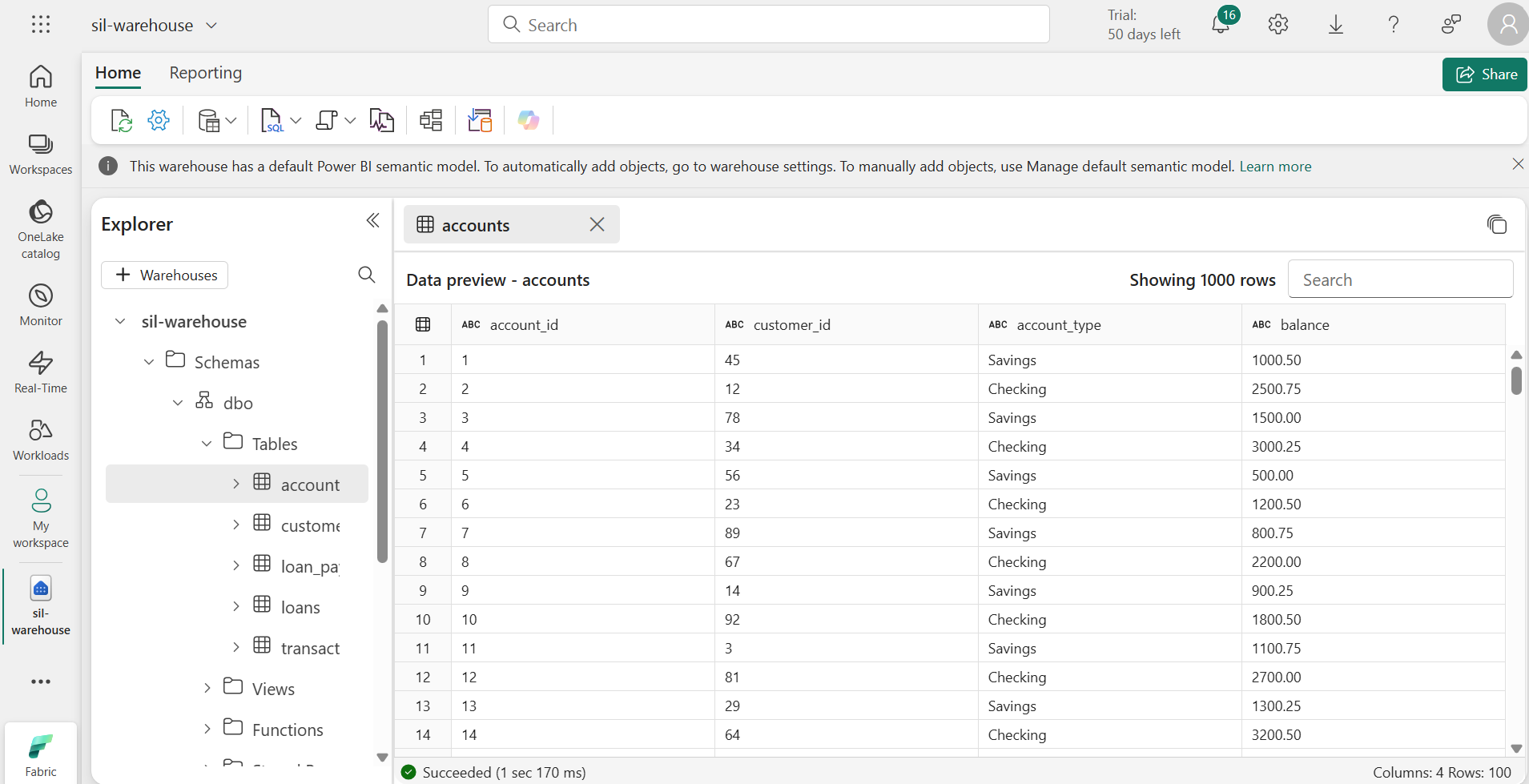
Using filter rows, give condition account\_id doesn’t equal to Null

****

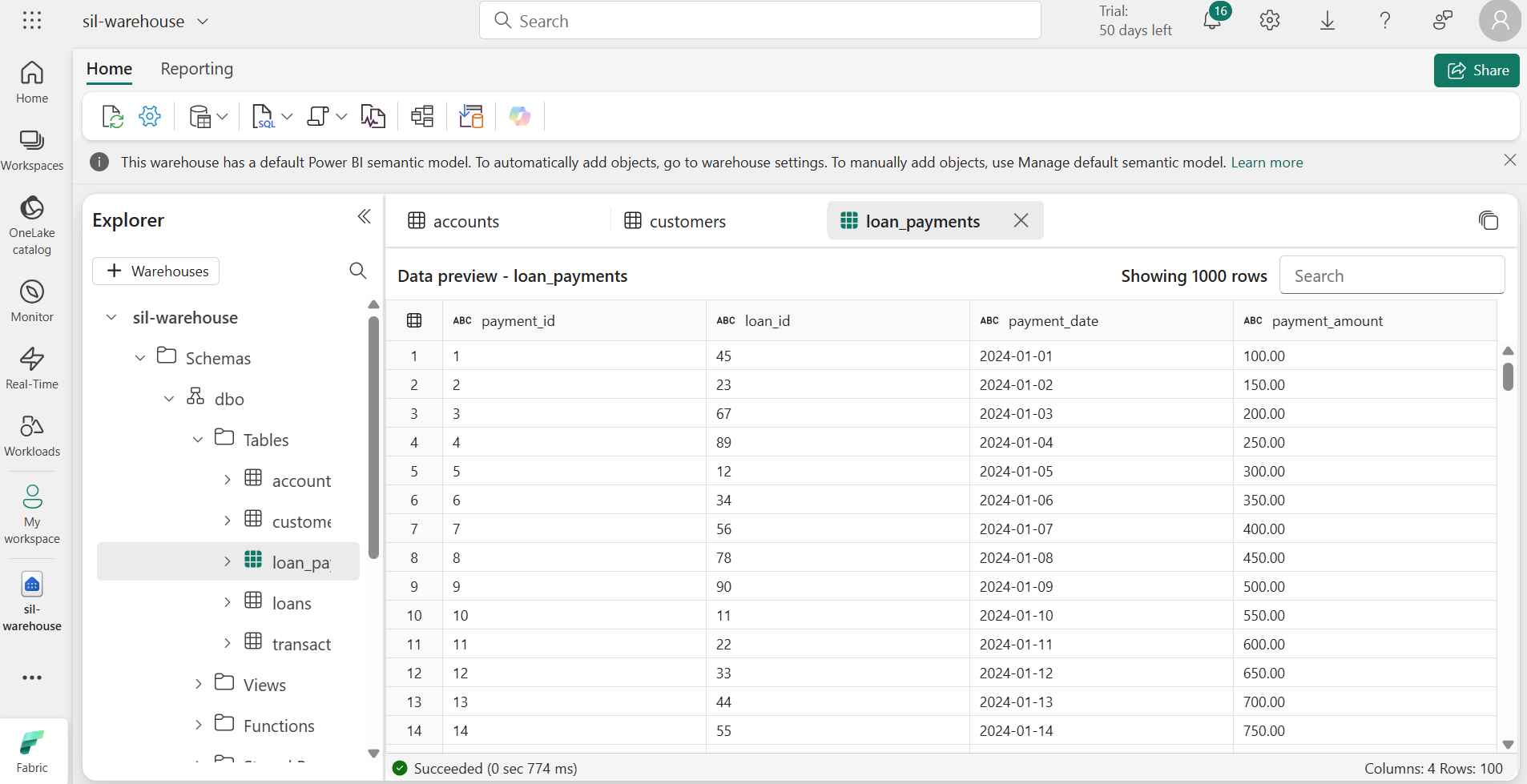
Add destination as warehouse

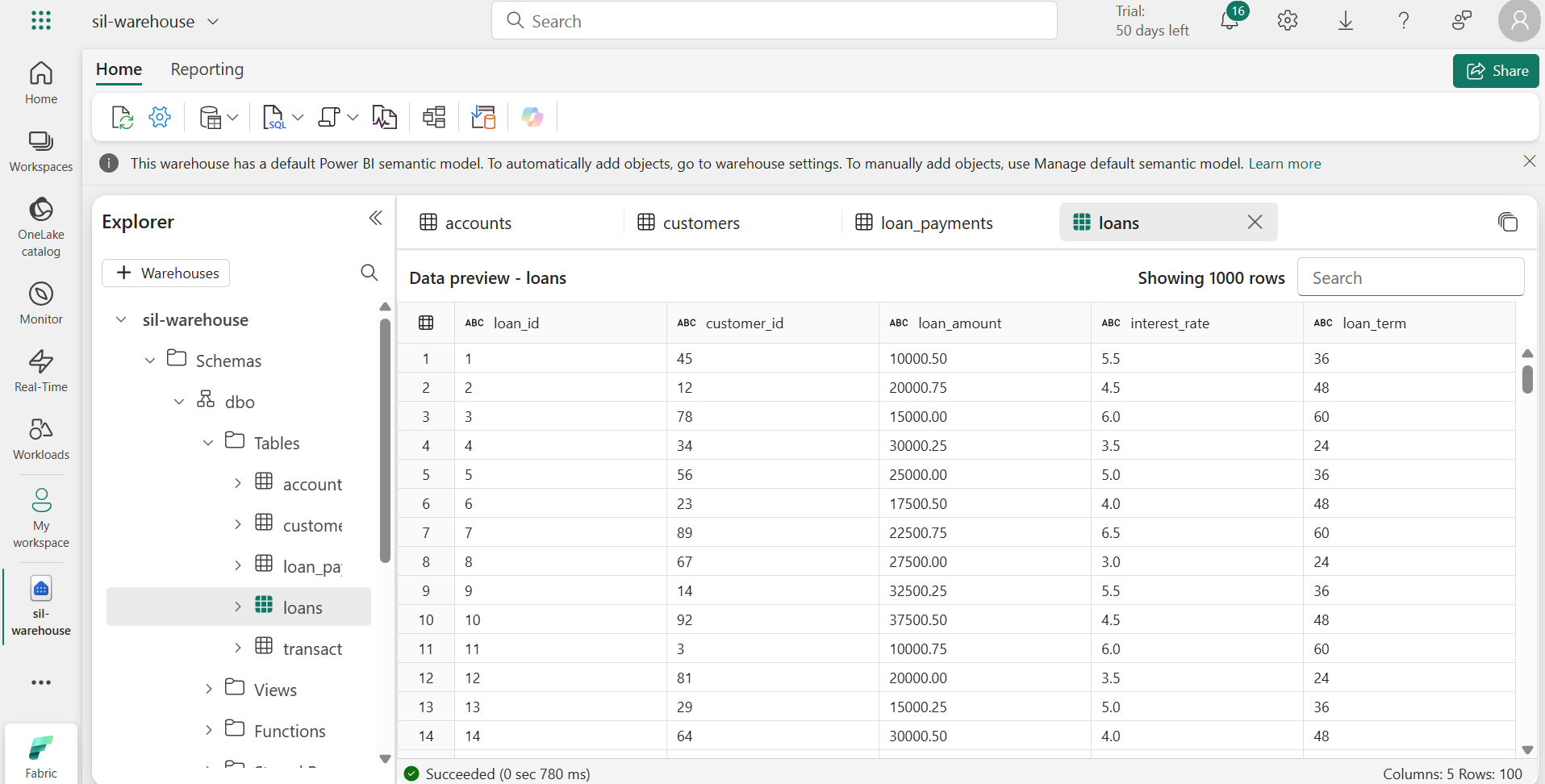
****

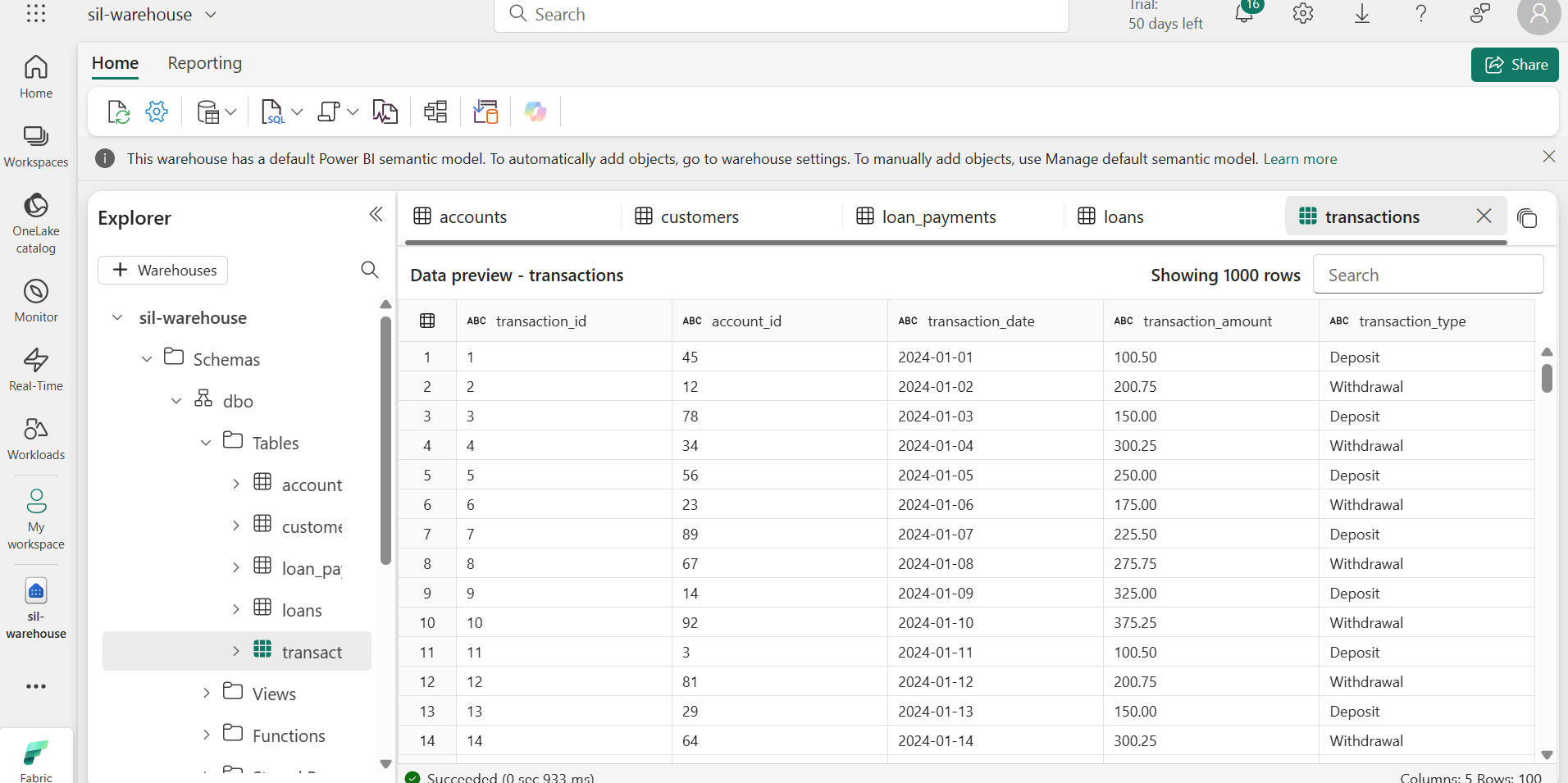
**Output-Warehouse**

****

****

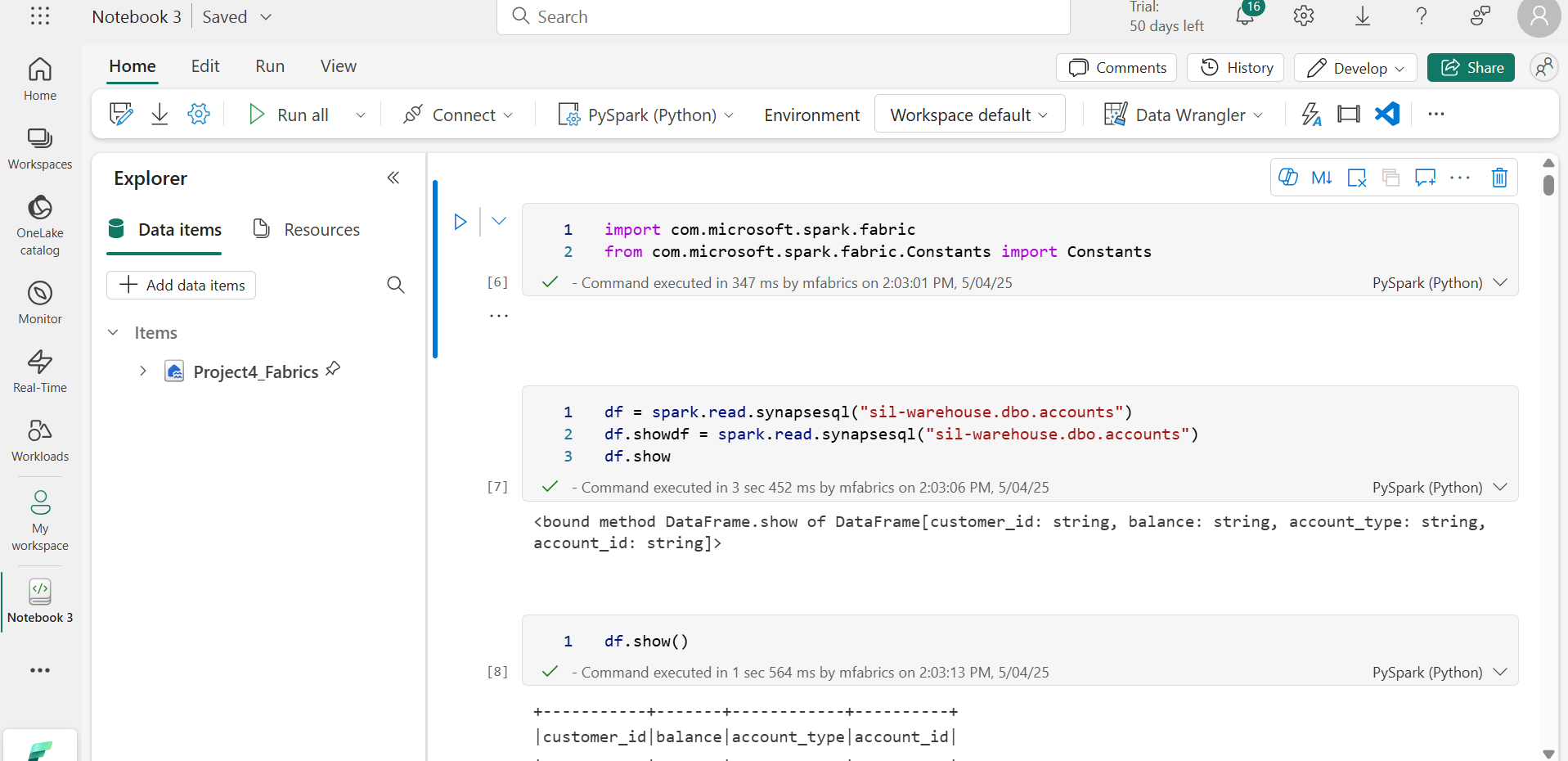
****

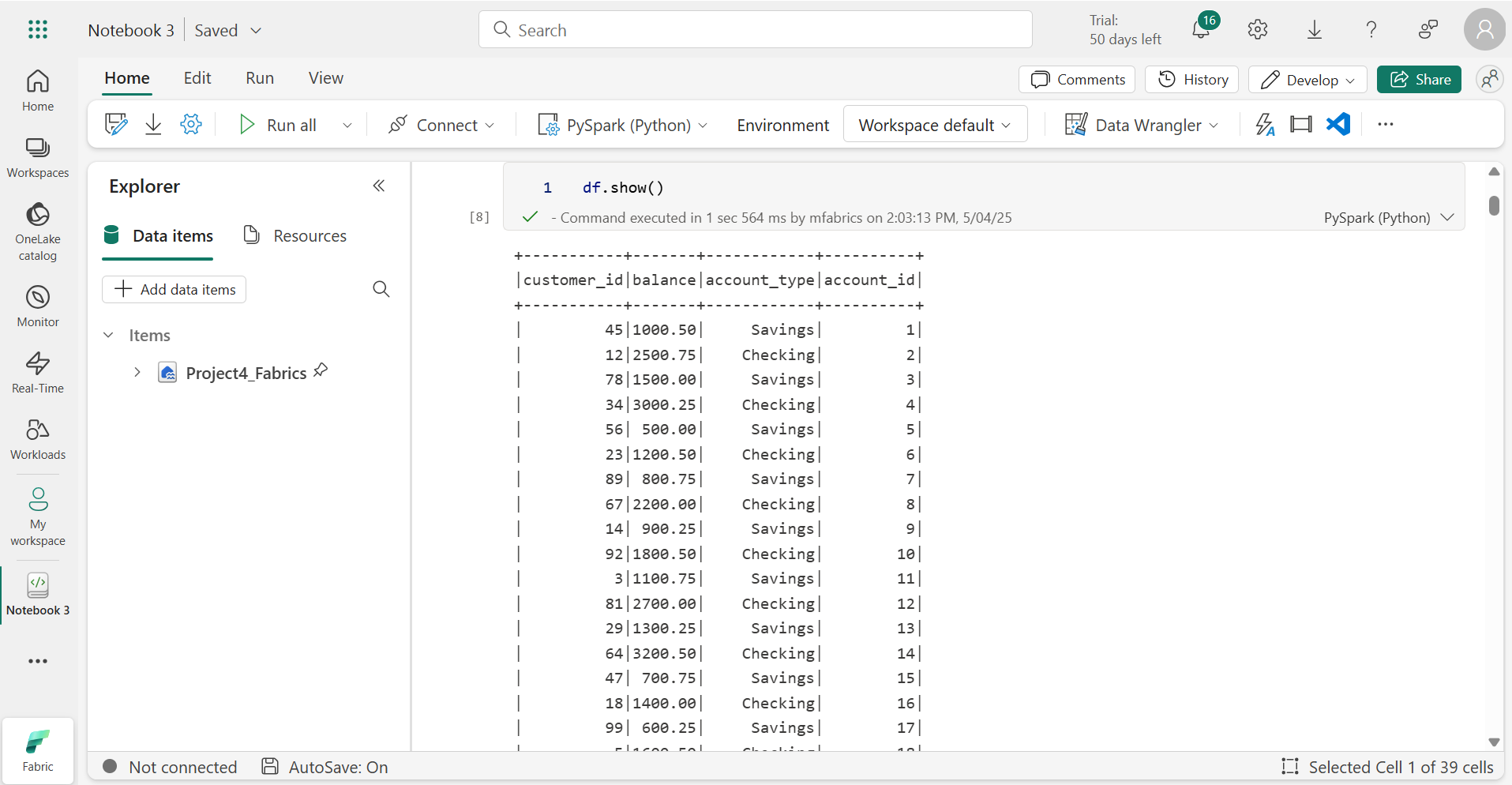
****

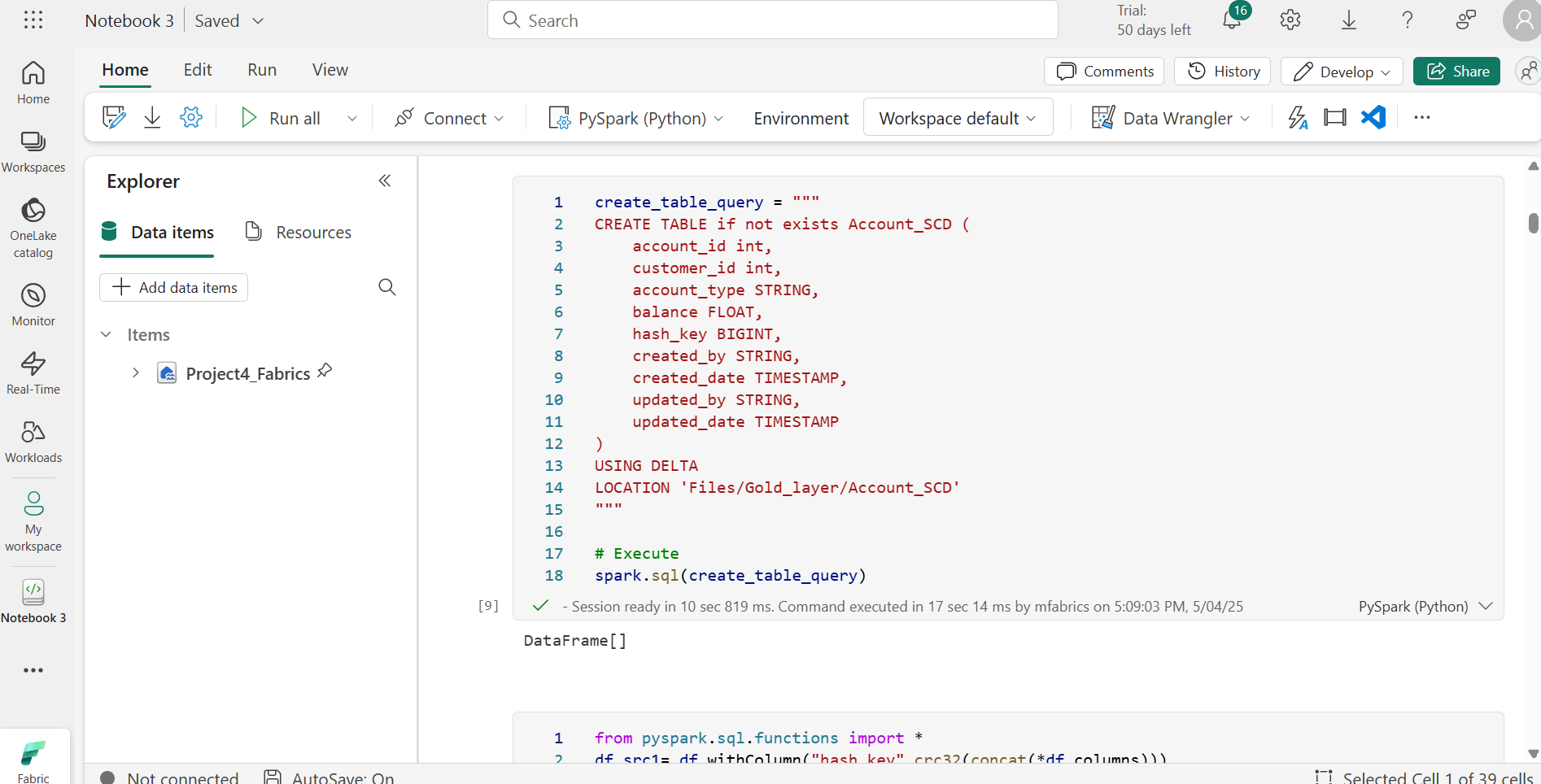
****

Gold Layer – Warehouse to lakehouse using Notebook

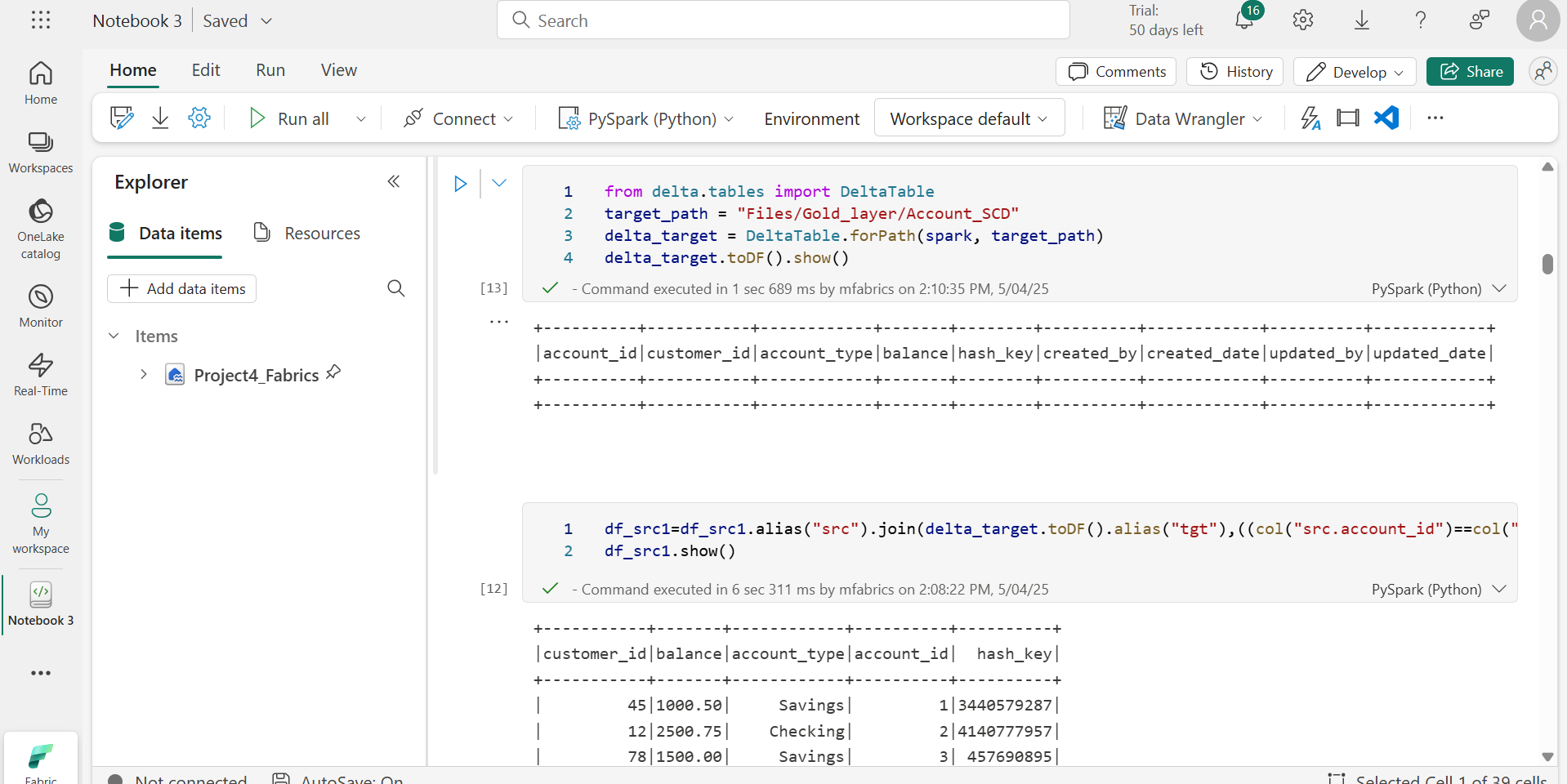
Using synapsesql connector, we can connect warehouse using notebooks.

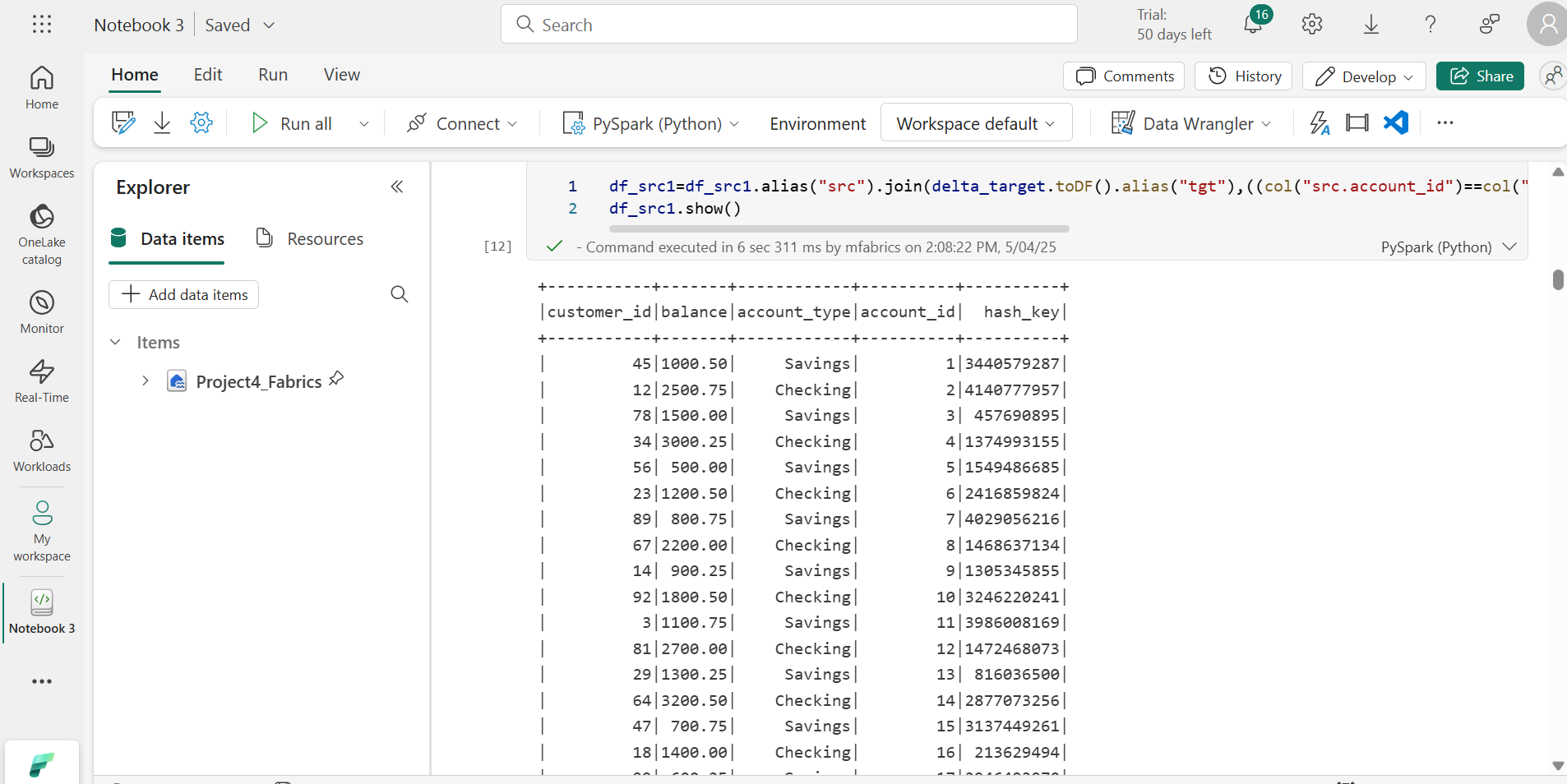














**Code**

**Using the below connectors to connect warehouse**

import com.microsoft.spark.fabric

from com.microsoft.spark.fabric.Constants import Constants

df = spark.read.synapsesql("sil-warehouse.dbo.accounts")

df.showdf = spark.read.synapsesql("sil-warehouse.dbo.accounts")

df.show()

**Create a target table**

create\_table\_query = """

CREATE TABLE if not exists Account\_SCD (

    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

)

USING DELTA

LOCATION 'Files/Gold\_layer/Account\_SCD'

"""

# Execute

spark.sql(create\_table\_query)

**Generating hash key**

from pyspark.sql.functions import \*

df\_src1= df.withColumn("hash\_key",crc32(concat(\*df.columns)))

display(df\_src1)

**Create delta table**

from delta.tables import DeltaTable

target\_path = "Files/Gold\_layer/Account\_SCD"

delta\_target = DeltaTable.forPath(spark, target\_path)

delta\_target.toDF().show()

df\_src1=df\_src1.alias("src").join(delta\_target.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()

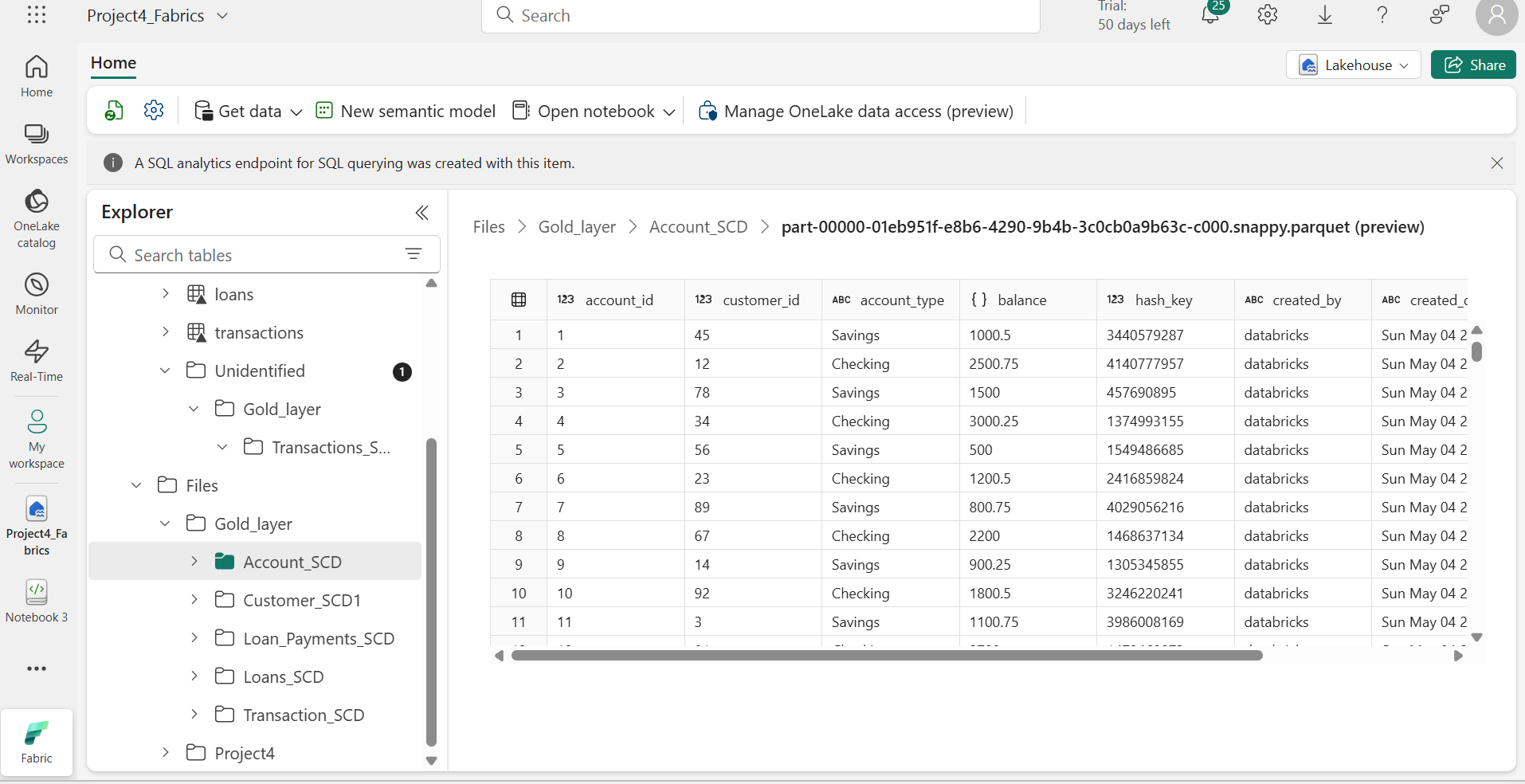
delta\_target.alias("tgt").merge(df\_src1.alias("src"),"tgt.account\_id = src.account\_id")\

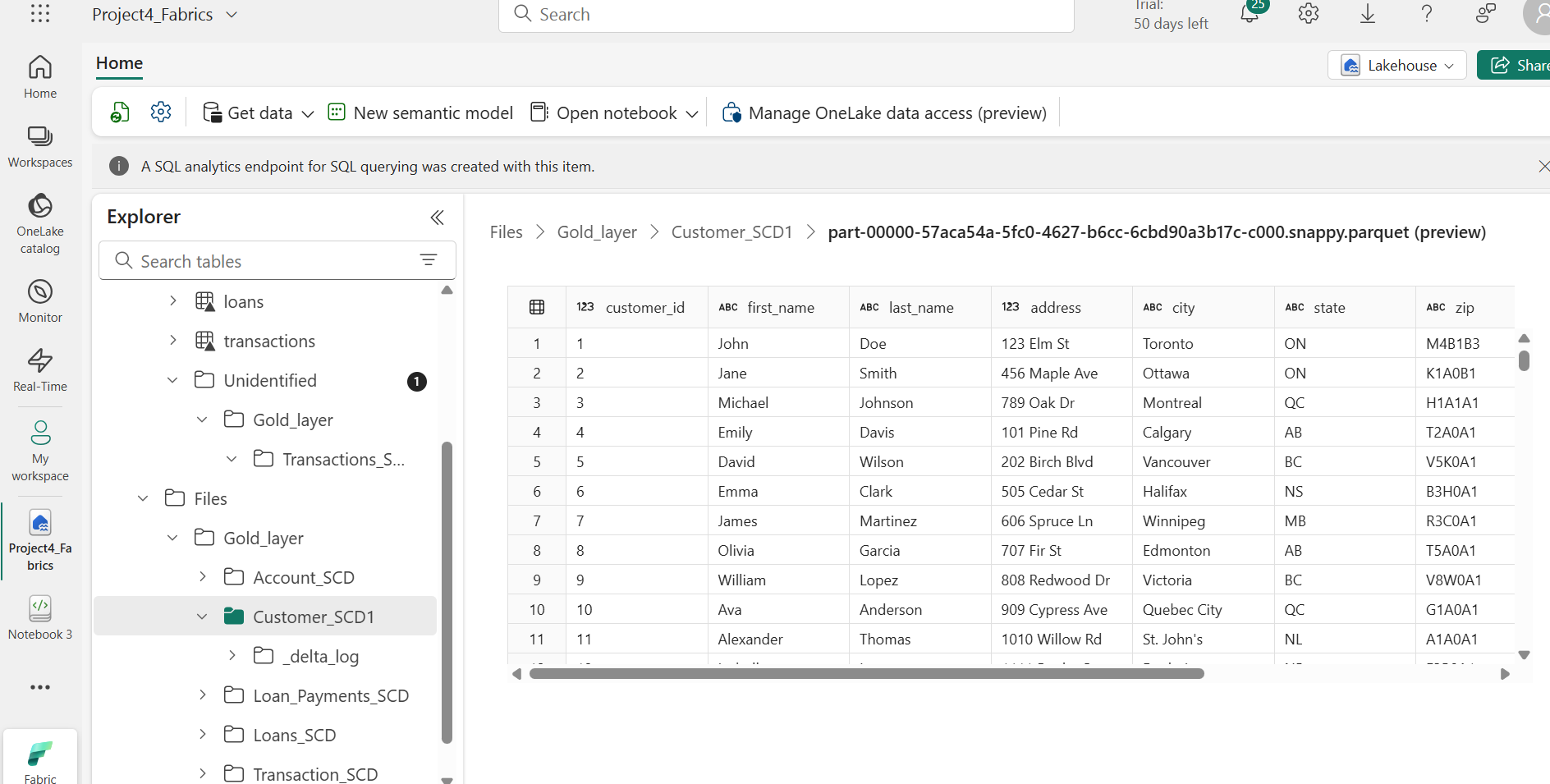
    .whenMatchedUpdate(set={"tgt.account\_id":"src.account\_id","tgt.customer\_id":"src.customer\_id","tgt.account\_type":"src.account\_type","tgt.balance":"src.balance","tgt.hash\_key":"src.hash\_key","tgt.updated\_date":current\_timestamp(),"tgt.updated\_by":lit("databricks\_Updated")})\

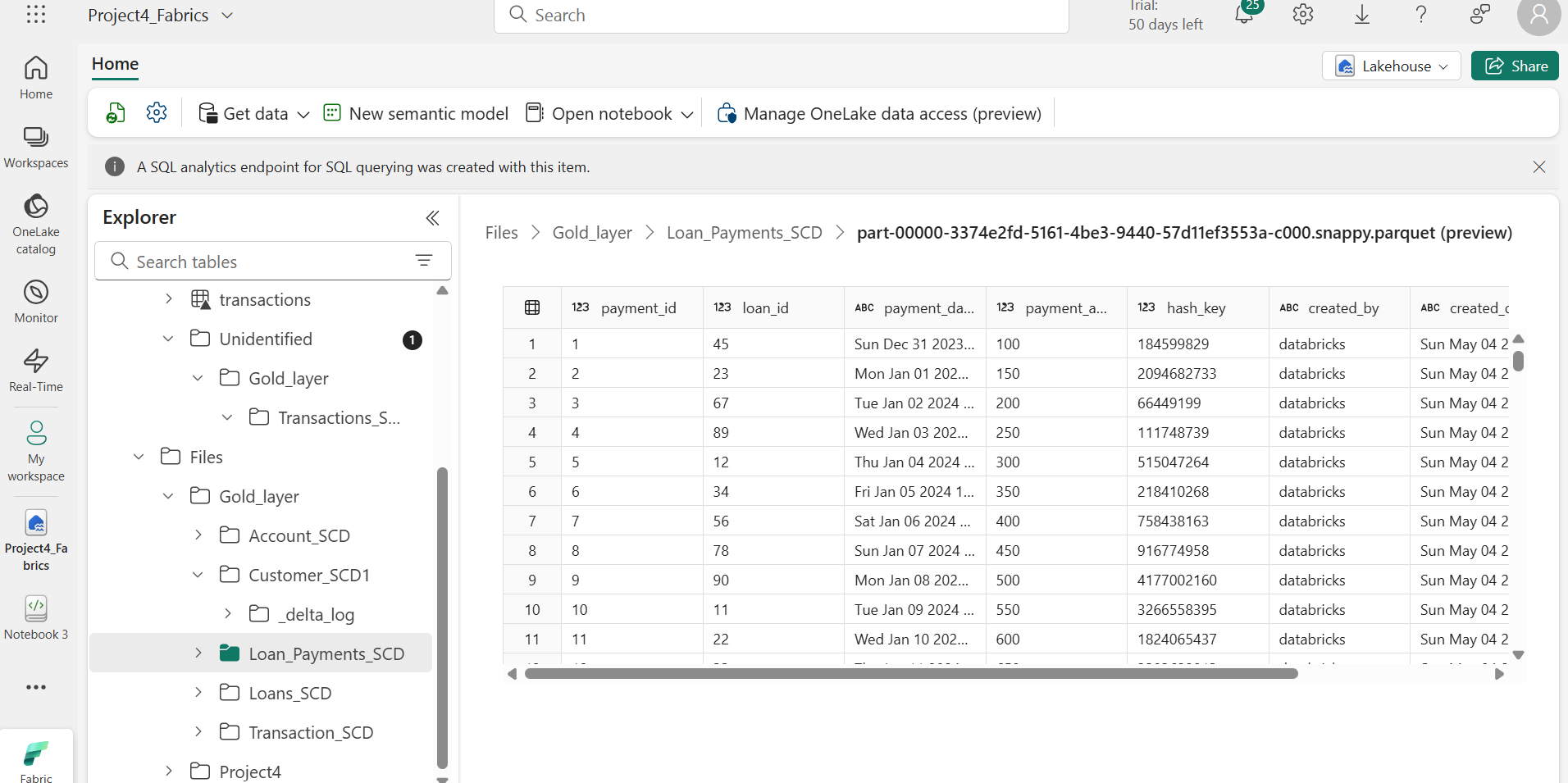
        .whenNotMatchedInsert(values={"tgt.account\_id":"src.account\_id","tgt.customer\_id":"src.customer\_id","tgt.account\_type":"src.account\_type","tgt.balance":"src.balance","tgt.hash\_key":"src.hash\_key","tgt.created\_date":current\_timestamp(),"tgt.created\_by":lit("databricks"),"tgt.updated\_date":current\_timestamp(),"tgt.updated\_by":lit("databricks")}).execute()

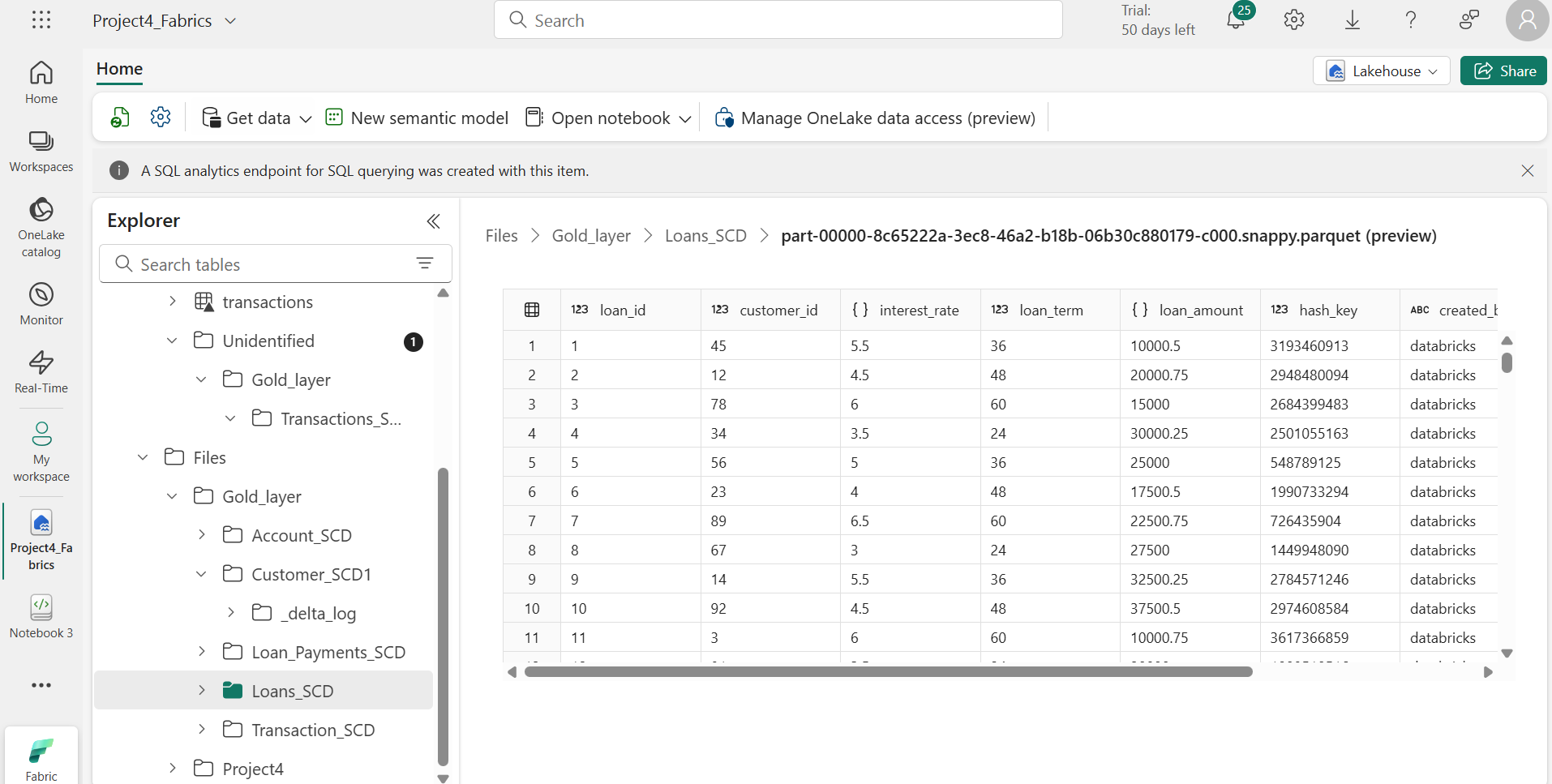
display(spark.read.format("delta").option("header","true").load(target\_path))

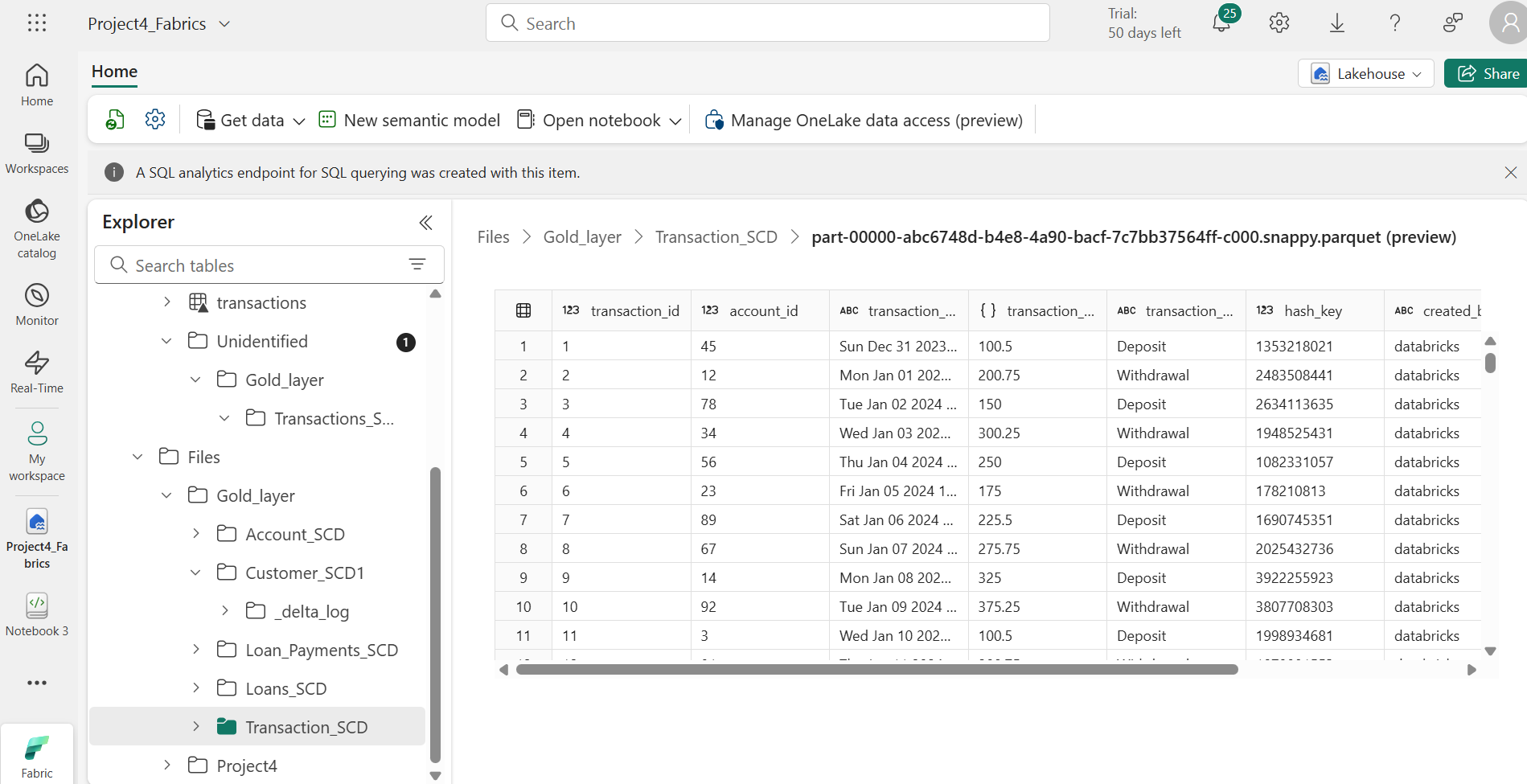
**Output**



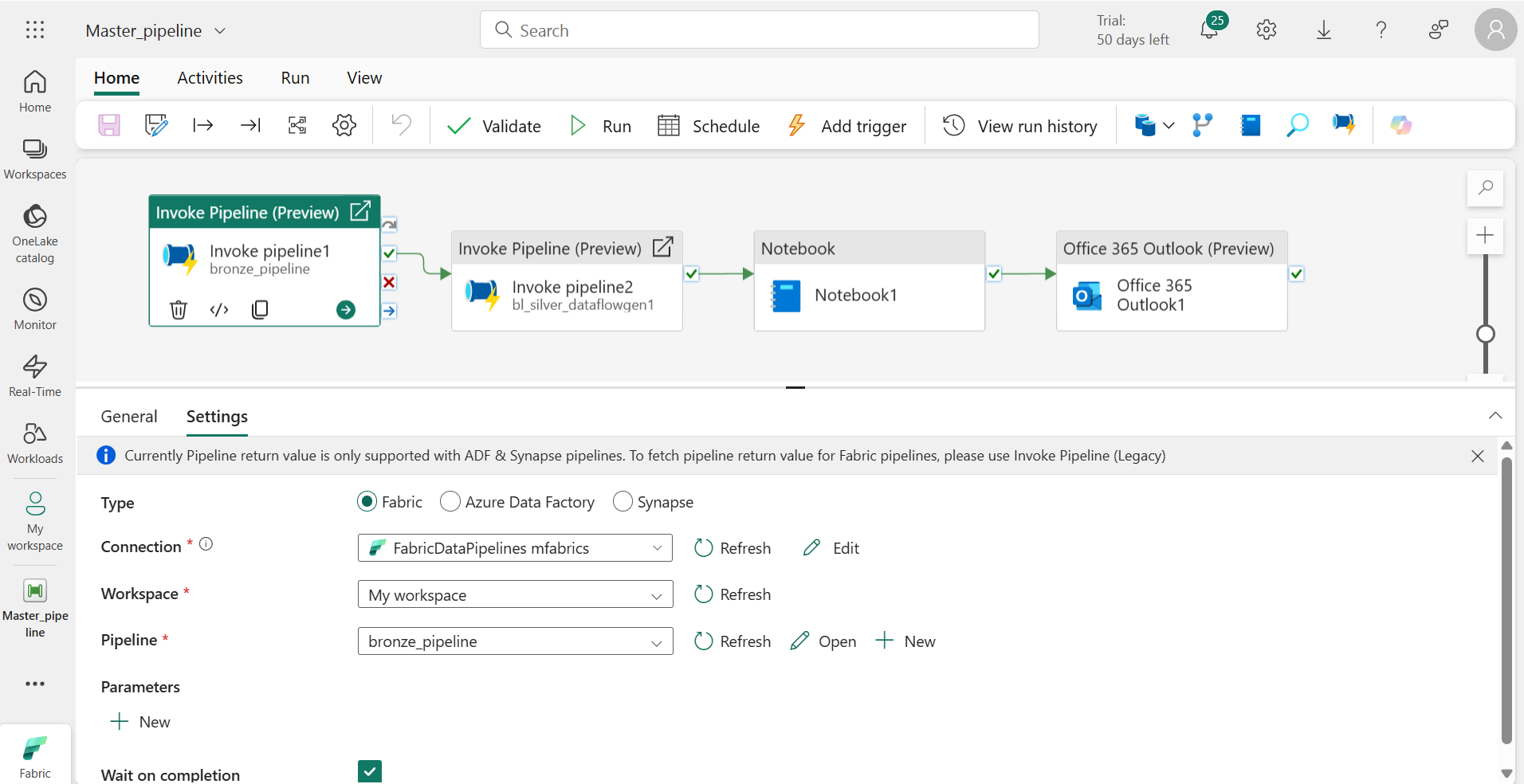


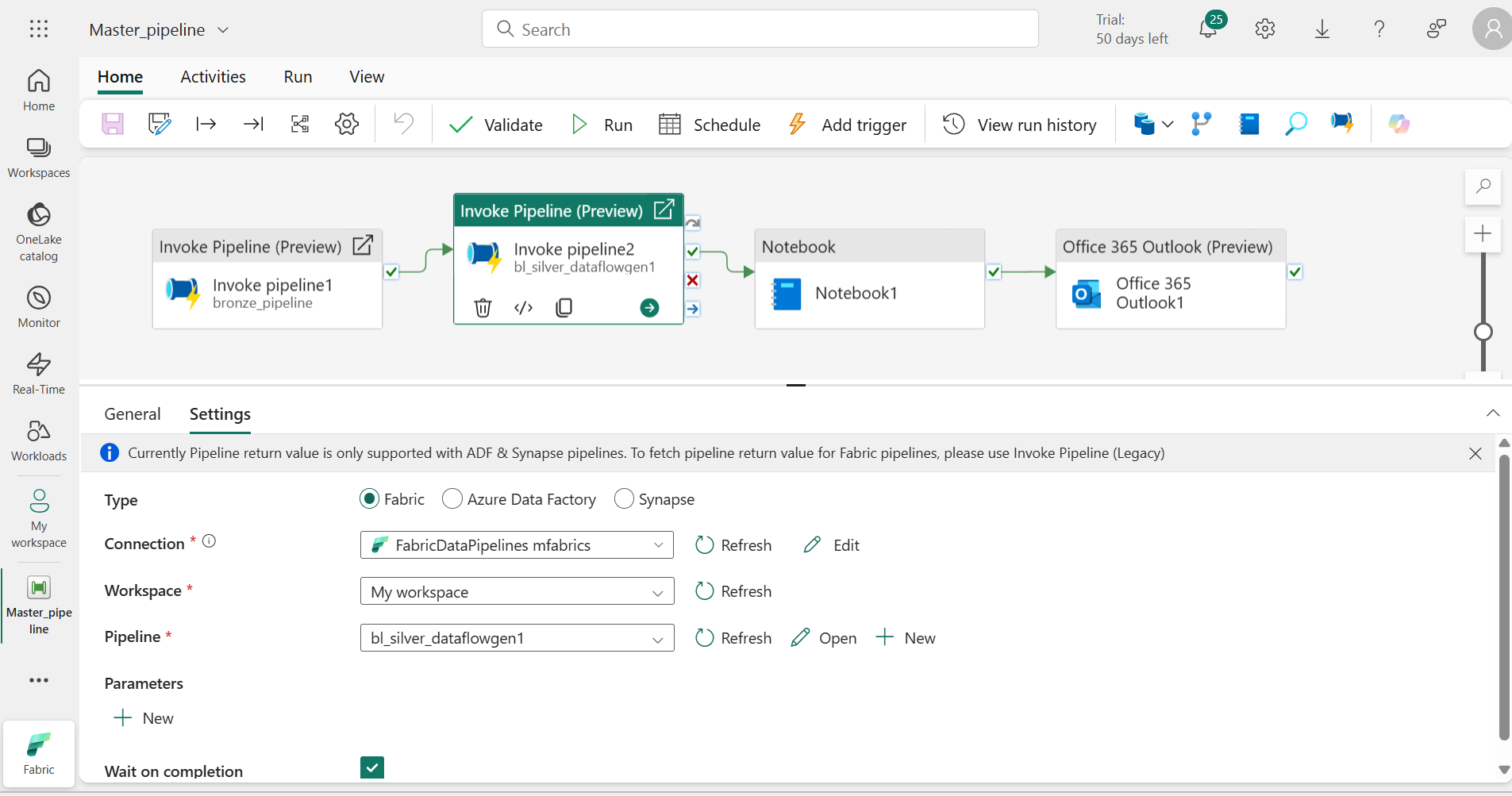


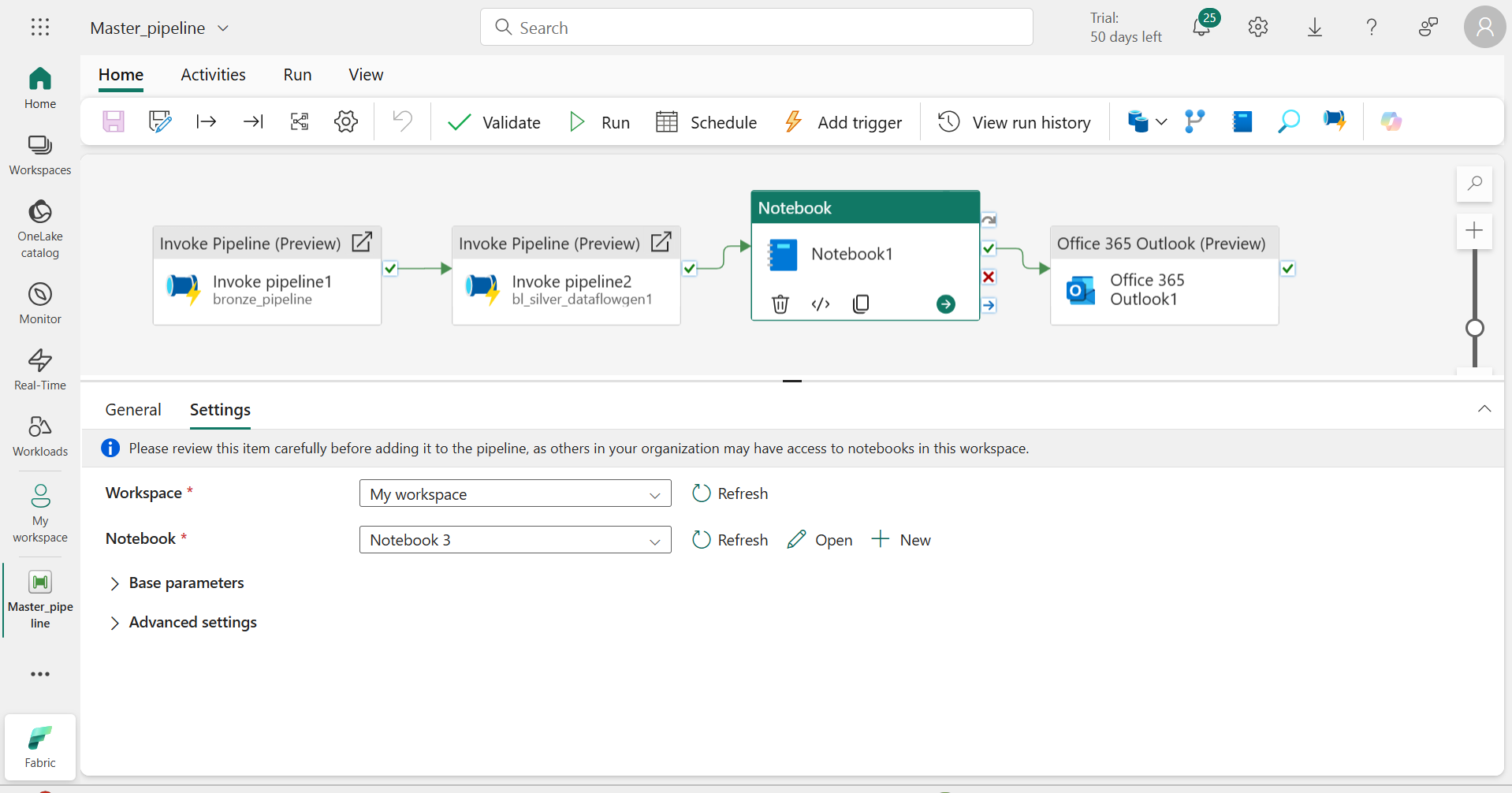


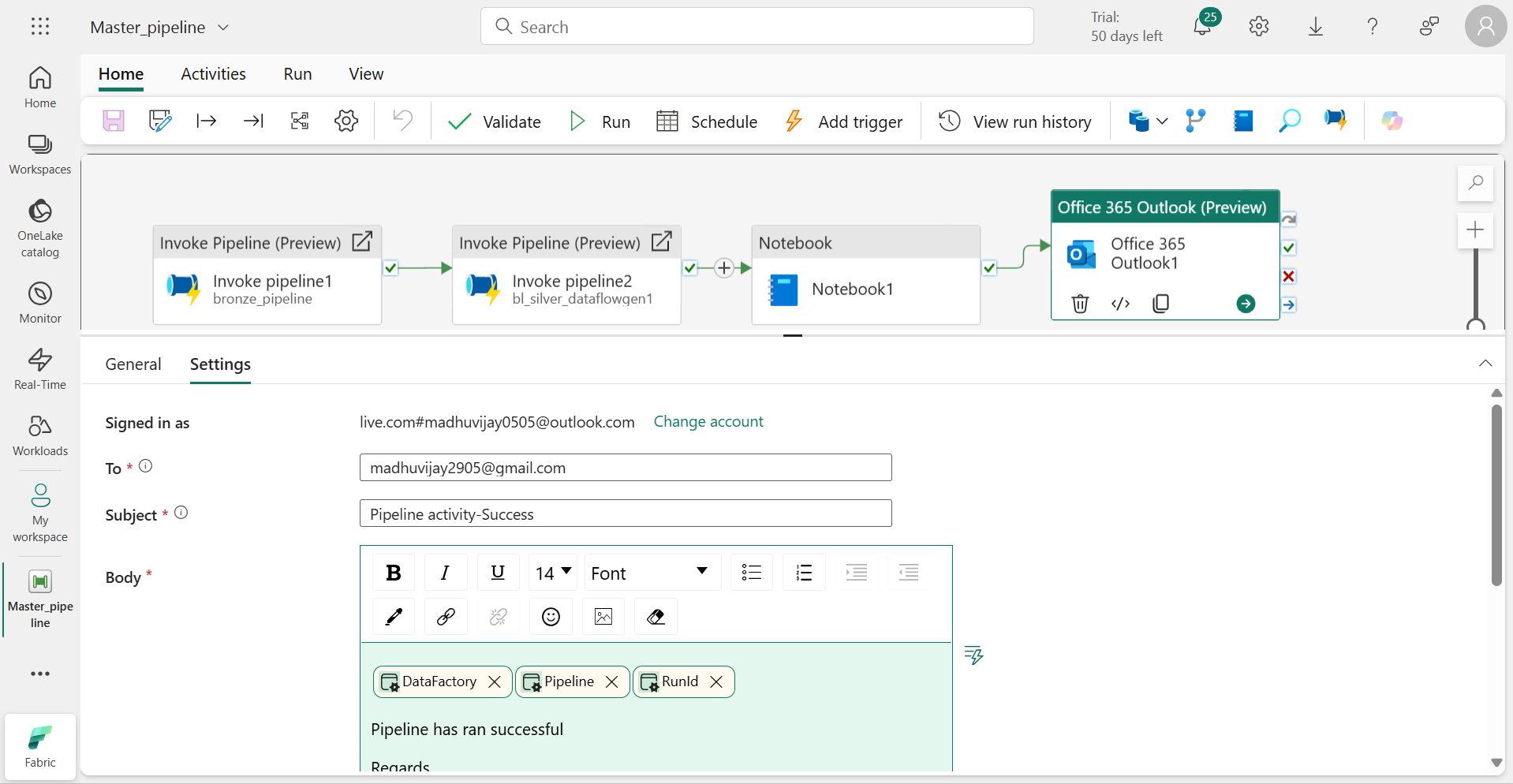


**Master Pipeline**

****

****

****

****