**Vacuum a Delta table.**

**Why use VACUUM on Delta Lake?**

VACUUM is used to clean up unused and stale data files that are taking up unnecessary storage space. Removing these files can help reduce storage costs.

When you run VACUUM on a Delta table it removes the following files from the underlying file system:

Any data files that are not maintained by Delta Lake

Removes stale data files (files that are no longer referenced by a Delta table) and are older than 7 days.

**Note: VACUUM does NOT remove directories that begin with an underscore, such as \_delta\_log.**

**When should you run VACUUM?**

When you run VACUUM, it removes stale data files. This does not impact regular work, but it can limit your ability to time travel.

The default configuration for a Delta table allows you to time travel 30 days into the past. However, to do this, the underlying data files must be present.

The default configuration for VACUUM deletes stale data files that are older than seven days. As a result, if you run VACUUM with the default settings, you will only be able to time travel seven days into the past, from the time you run VACUUM.

If you do not need to time travel more than seven days into the past, you can VACUUM on a daily basis.

Running VACUUM daily helps keep storage costs in check, especially for larger tables. You can also run VACUUM on-demand if you notice a sudden surge in the storage costs for a specific Delta table.

**Issues you may face with VACUUM.**

* No progress update: You may not know how far the VACUUM has completed, especially when VACUUM has run for a long time. You may not know how many files have been successfully removed and how many files remain.
* Poor run performance: VACUUM runs for a long time, especially when tables are huge and/or when tables are a source for high frequency input streams.

**Mitigate issues with VACUUM.**

**No progress update.**

If VACUUM completes within an hour or two, there is no need to troubleshoot. However, if VACUUM runs for longer than two hours (this can happen on large tables when VACUUM hasn’t been run recently), you may want to check the progress. In this case you can run VACUUM with the DRY RUN option before and after the actual VACUUM run to monitor the performance of a specific VACUUM run and to identify the number of files deleted.

Run VACUUM DRY RUN to determine the number of files eligible for deletion. Replace <table-path> with the actual table path location.

**%python**

**spark.sql("VACUUM delta.`<table-path>` DRY RUN")**

The DRY RUN option tells VACUUM it should not delete any files. Instead, DRY RUN prints the number of files and directories that are safe to be deleted. The intention in this step is not to delete the files but know the number of files eligible for deletion.

The example DRY RUN command returns an output which tells us that there are x files and directories that are safe to be deleted.

Found x files and directories in a total of y directories that are safe to delete.

You should record the number of files identified as safe to delete.

Run VACUUM.

Cancel VACUUM after one hour.

Run VACUUM with DRY RUN again.

The second DRY RUN command identifies the number of outstanding files that can be safely deleted.

Subtract the outstanding number of files (second DRY RUN) from the original number of files to get the number of files that were deleted.

**Note:**

**You can also review your storage bucket information in your cloud portal to identify the remaining number of files existing in the bucket, or the number of deletion requests issued, to determine how far the deletion has progressed.**

**Poor run performance**

This can be mitigated by following VACUUM best practices.

Avoid actions that hamper performance.

Avoid over-partitioned data folders.

Over-partitioned data can result in a lot of small files. You should avoid partitioning on a high cardinality column. When you over-partition data, even running OPTIMIZE can have issues compacting small files, as compaction does not happen across partition directories.

File deletion speed is directly dependent on the number of files. Over-partitioning data can hamper the performance of VACUUM.

Info

You should partition on a low cardinality column and z-order on a high cardinality column.

**The following is the end to end automation of the vacuum implementation using Synapse Pipeline, data bricks script & SQL:**

**Prerequisite:**

* Synapse workspace
* Databricks Notebook
* SQL Server
* Github Action Pipeline

Let’s see the example below:

**Step1: To create schema and table use the below query.**

--To create schema and table use the below query

CREATE TABLE [monitoring].[Housekeeping\_Vacuum](

[HID] [int] IDENTITY(1,1) NOT NULL,

[TargetFolderPath] [nvarchar](max) NULL,

[LastVaccumDateTime] [datetime] NULL,

[LastZOrderDateTime] [datetime] NULL,

[Comments] [varchar](255) NULL,

PRIMARY KEY CLUSTERED

(

[HID] ASC

)WITH (STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, OPTIMIZE\_FOR\_SEQUENTIAL\_KEY = OFF) ON [PRIMARY]

) ON [PRIMARY] TEXTIMAGE\_ON [PRIMARY]

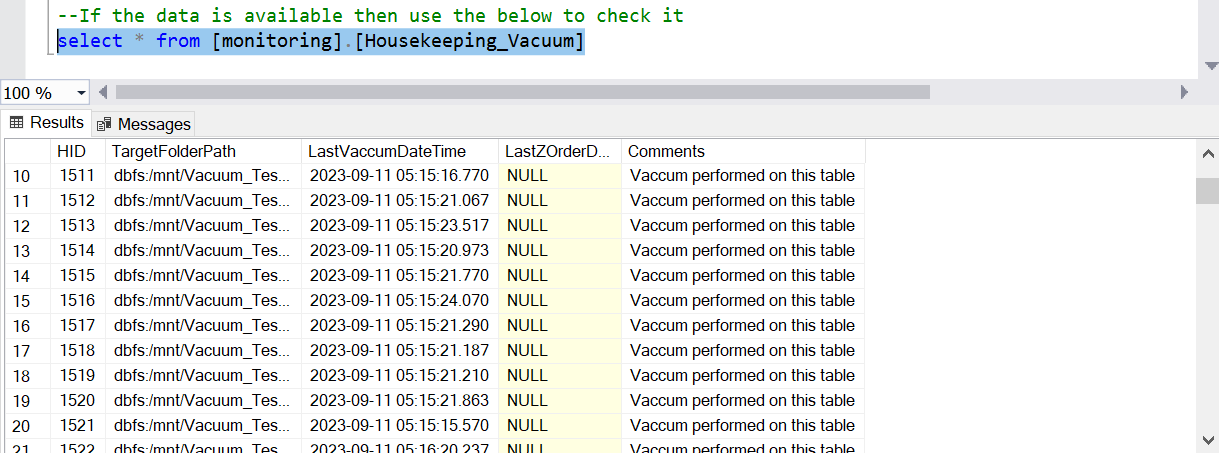
GO

--To create schema if it does not exists

CREATE SCHEMA monitoring

--If the data is available then use the below to check it

select \* from [monitoring].[Housekeeping\_Vacuum]



**Step2: Create Stored Procedure for reusability:**

/\*\*\*\*\*\* Object: StoredProcedure [DATACATALOGUE].[ADLSHouseKeepingLoad] Script Date: 8/30/2023 12:15:21 PM \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

-- =============================================

-- Author: <Author, , Name>

-- Create Date: <Create Date, , >

-- Description: <Description, , >

-- =============================================

CREATE PROCEDURE [monitoring].[ADLSHouseKeepingLoad]

(

-- Add the parameters for the stored procedure here

@TargetFolderPath NVARCHAR(max),

@IsVaccumDone BIT,

@IsZOrderDone BIT,

@Comments NVARCHAR(max) = Null

)

AS

BEGIN

-- SET NOCOUNT ON added to prevent extra result sets from

DECLARE @VaccumDatetime DATETIME;

DECLARE @ZorderDatetime DATETIME;

DECLARE @RecordCheck BIT;

SET NOCOUNT ON

-- SET vaccumdatetime if vaccum is performed

IF @IsVaccumDone = 1

SET @VaccumDatetime = GETUTCDATE()

Else

SET @VaccumDatetime = (SELECT DISTINCT LastVaccumDateTime FROM [monitoring].[Housekeeping\_Vacuum]

WHERE TargetFolderPath = @TargetFolderPath)

-- SET zorderdatetime if zordering is performed

IF @IsZOrderDone = 1

SET @ZorderDatetime = GETUTCDATE()

Else

SET @ZorderDatetime = (SELECT DISTINCT LastZOrderDateTime FROM [monitoring].[Housekeeping\_Vacuum]

WHERE TargetFolderPath = @TargetFolderPath)

-- SET @RecordCheck = 1 if record already exists else 0

IF EXISTS

(

SELECT DISTINCT

TargetFolderPath

FROM [monitoring].[Housekeeping\_Vacuum]

WHERE TargetFolderPath = @TargetFolderPath

)

SET @RecordCheck = 1

ELSE

SET @RecordCheck = 0

IF @RecordCheck = 0

BEGIN

INSERT INTO [monitoring].[Housekeeping\_Vacuum]

([TargetFolderPath]

,[LastVaccumDateTime]

,[LastZOrderDateTime]

,[Comments])

VALUES

(@TargetFolderPath

,@VaccumDatetime

,@ZorderDatetime

,@Comments)

END

ELSE IF @RecordCheck = 1

BEGIN

UPDATE [monitoring].[Housekeeping\_Vacuum]

SET [LastVaccumDateTime] = @VaccumDatetime,

[LastZOrderDateTime] = @ZorderDatetime,

[Comments] = @Comments

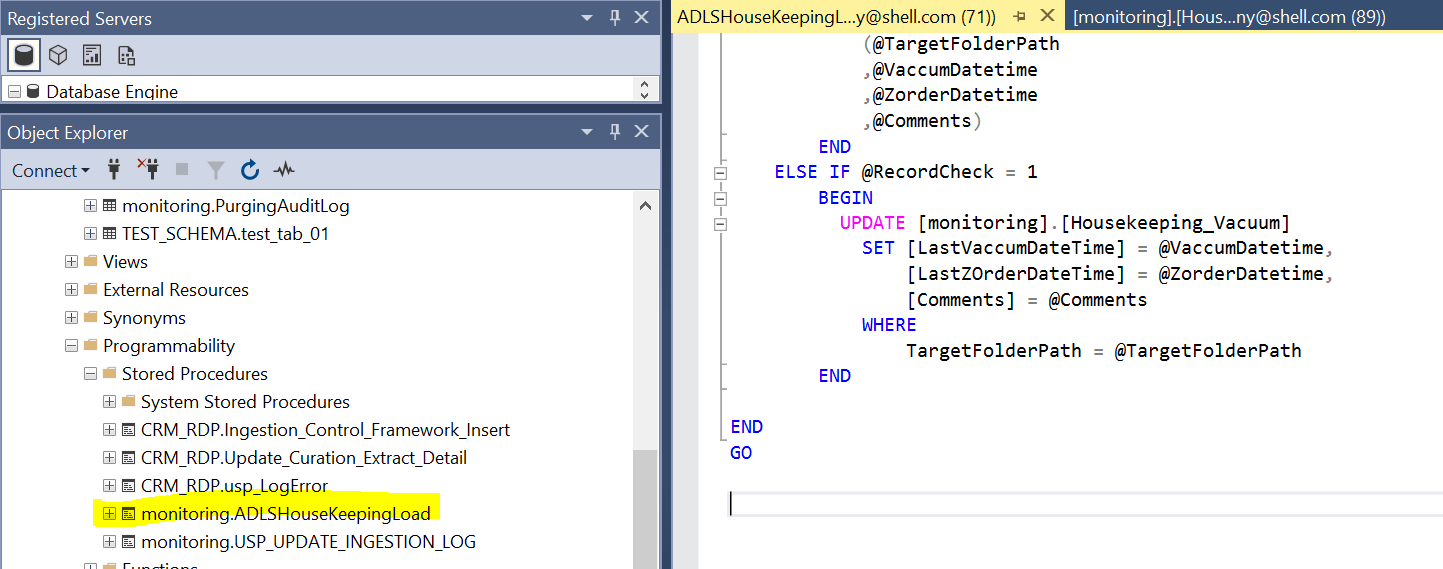
WHERE

TargetFolderPath = @TargetFolderPath

END

END

GO



**Step3: Create Databricks script.**

To achieve the vacuum implementation, two scripts needed to be created.

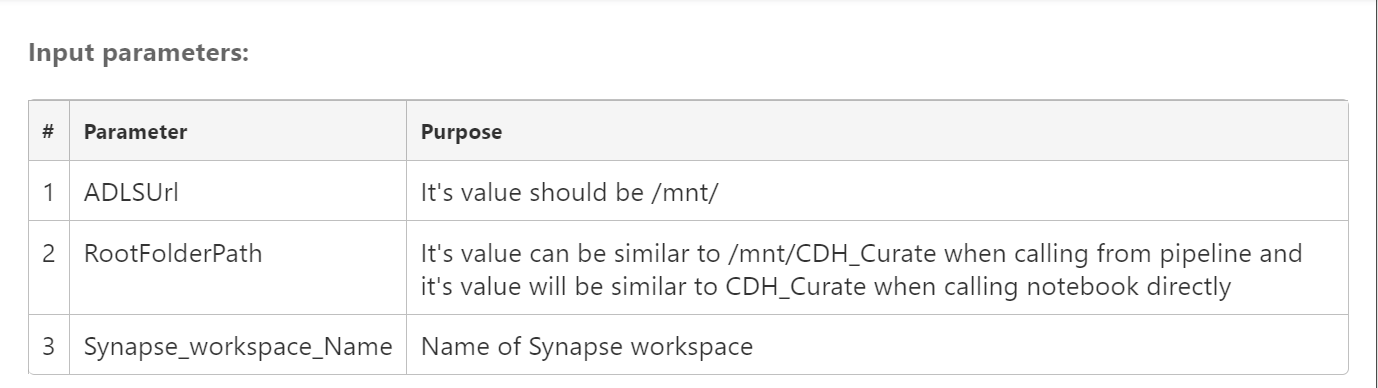
**NOTEBOOK: 01\_ADLS\_LIST\_DELTA\_UPDATE\_SQL**

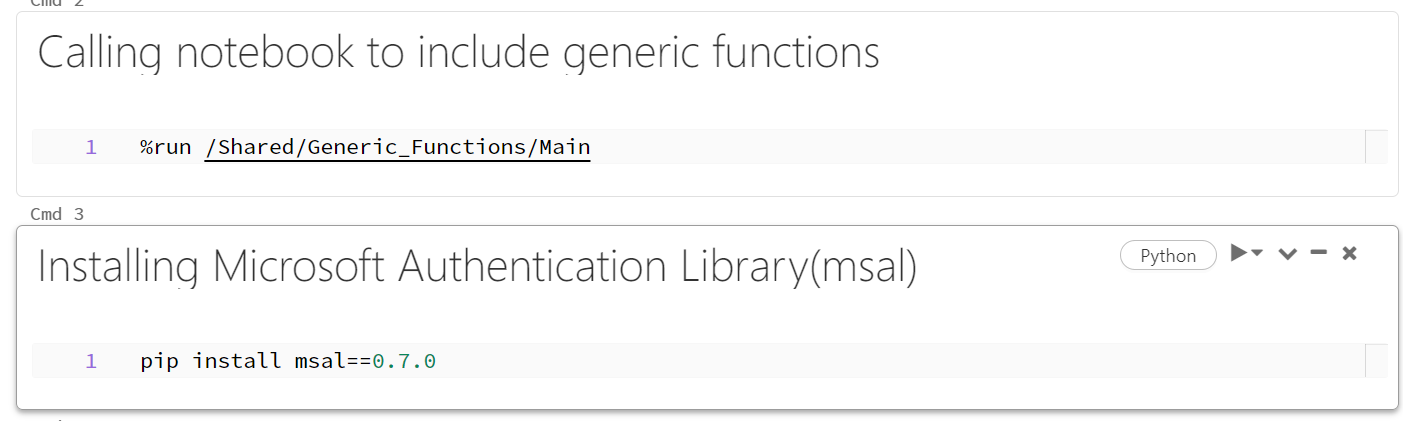
Notebook to scan the given root folder path and find all delta directories and log these directories into the sql log table [monitoring].[Housekeeping\_Vacuum]

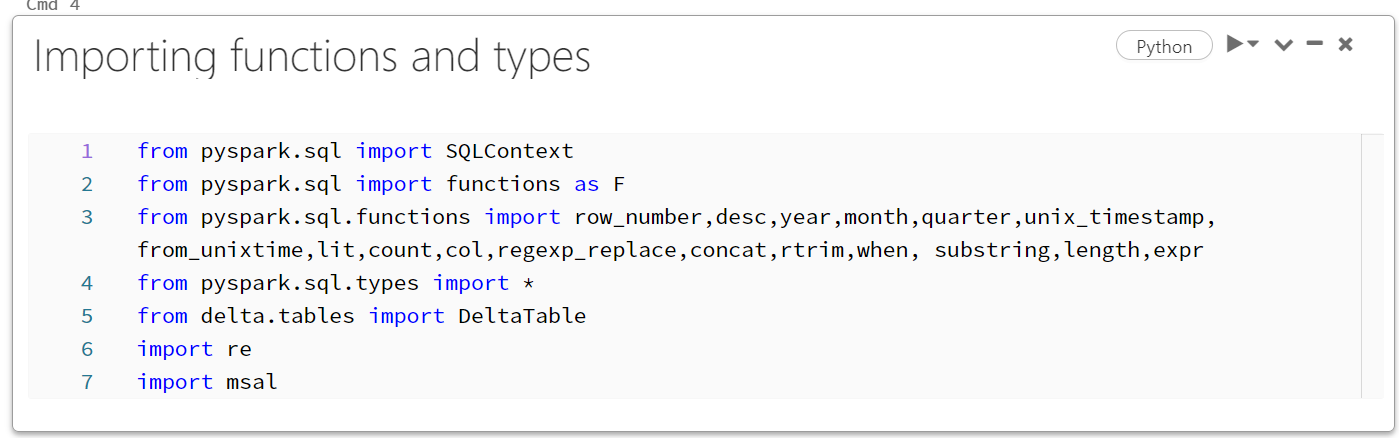
**NOTEBOOK: 02\_ADLS\_DELTA\_VACCUM\_NTBK**

Notebook to perform the VACUUM operation (for 7 days = 24 hrs X 7 days= 168 hrs) on given delta target path

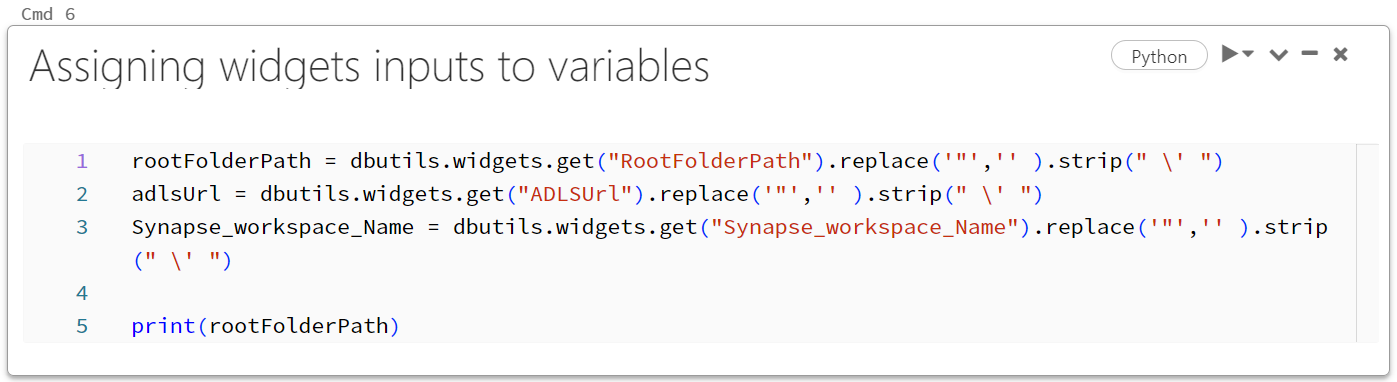
**NOTEBOOK: 01\_ADLS\_LIST\_DELTA\_UPDATE\_SQL**

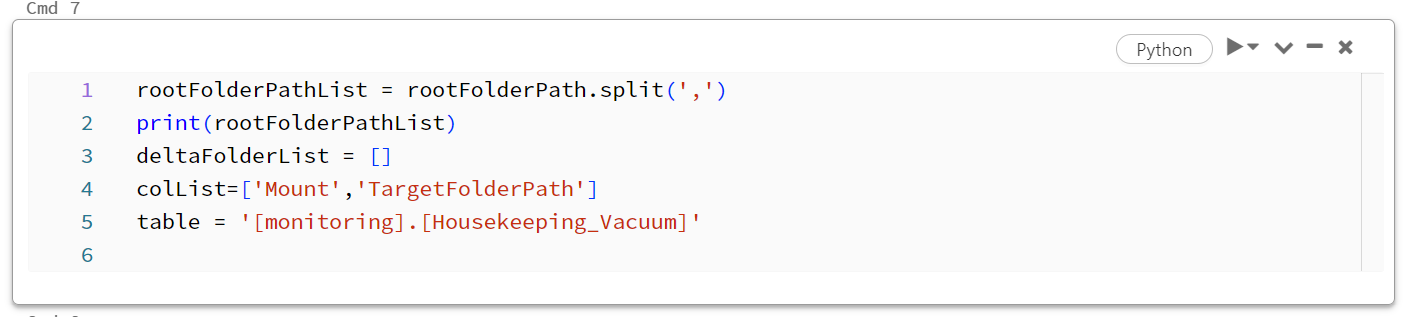


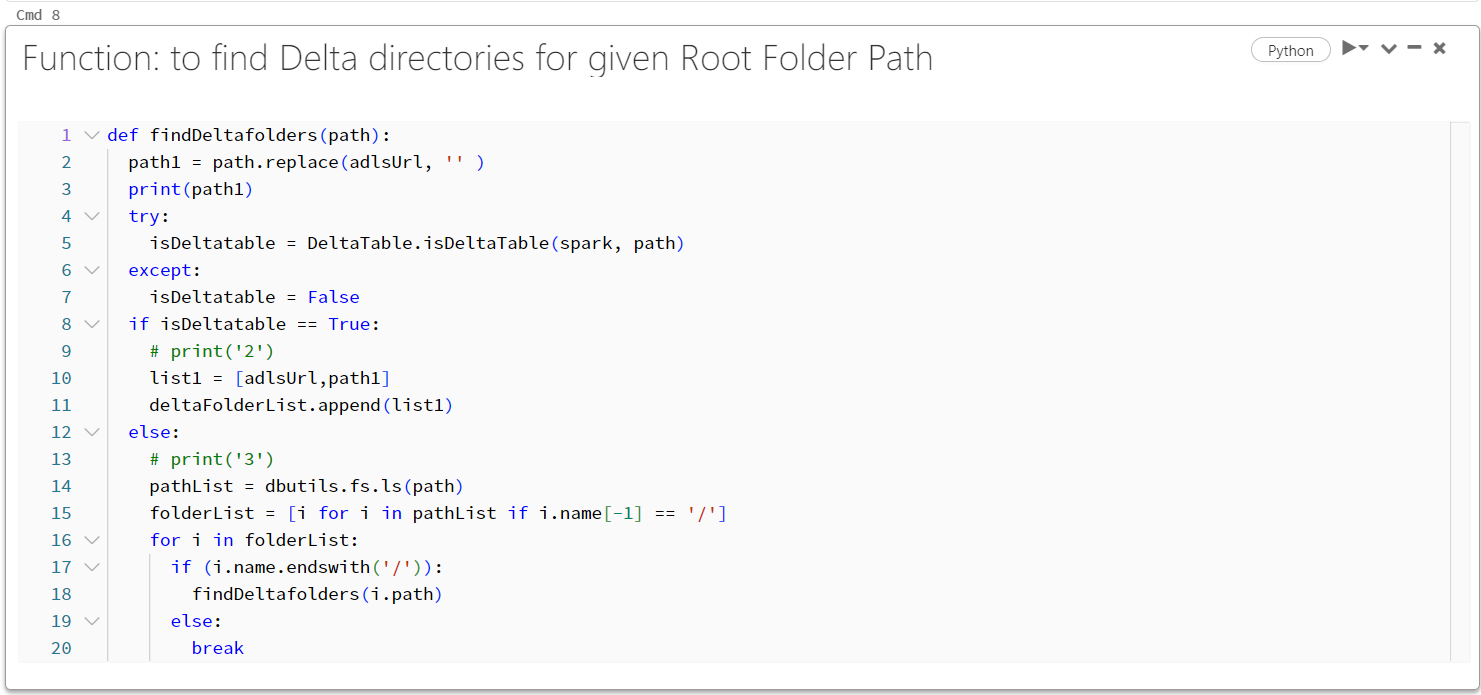


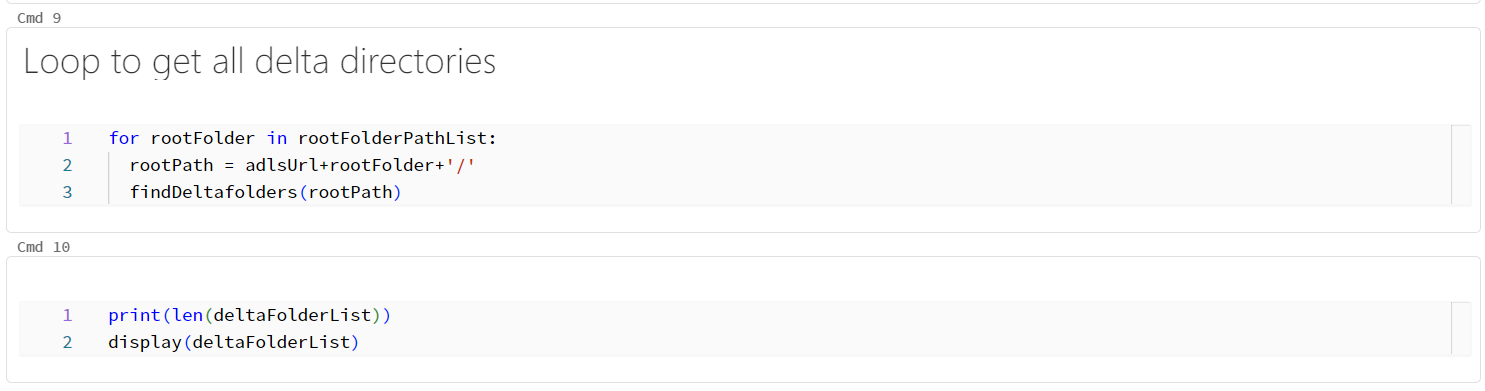


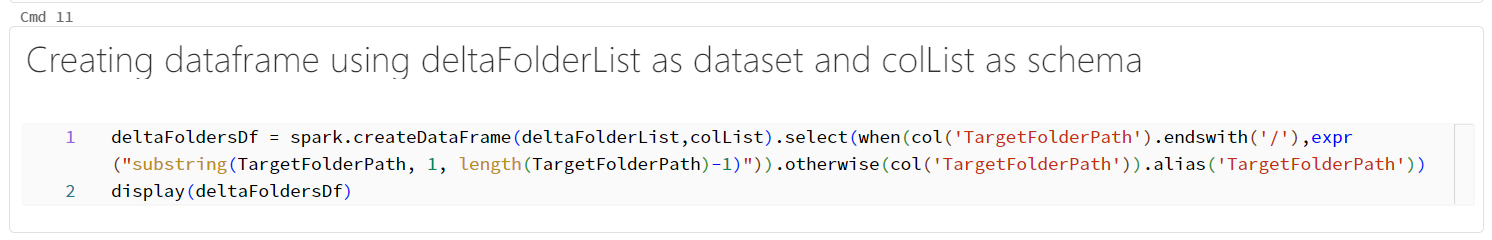


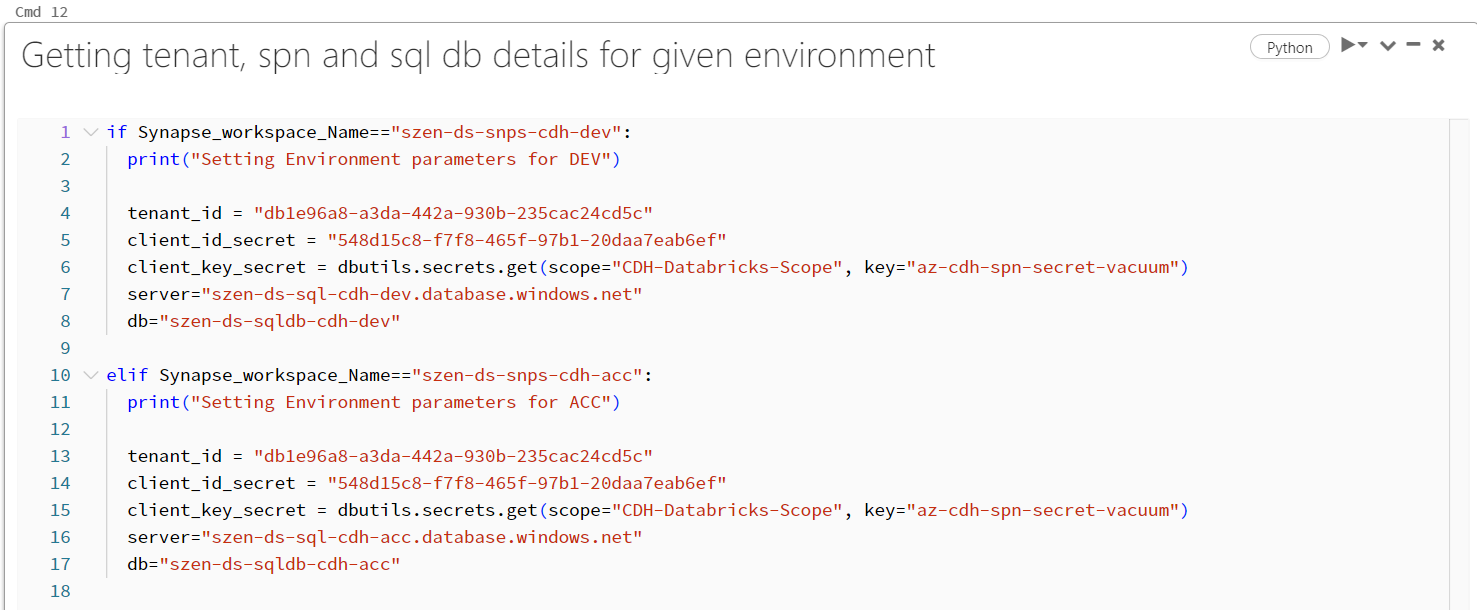


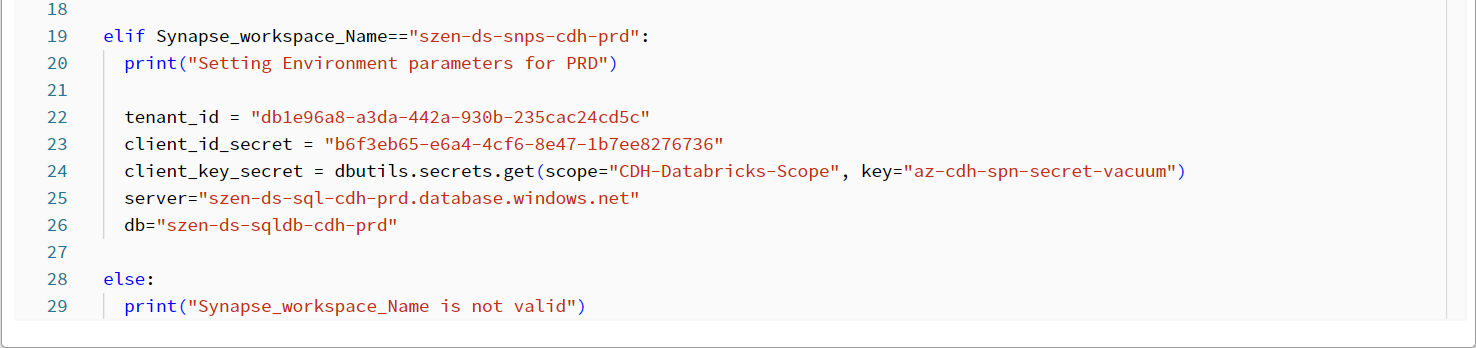


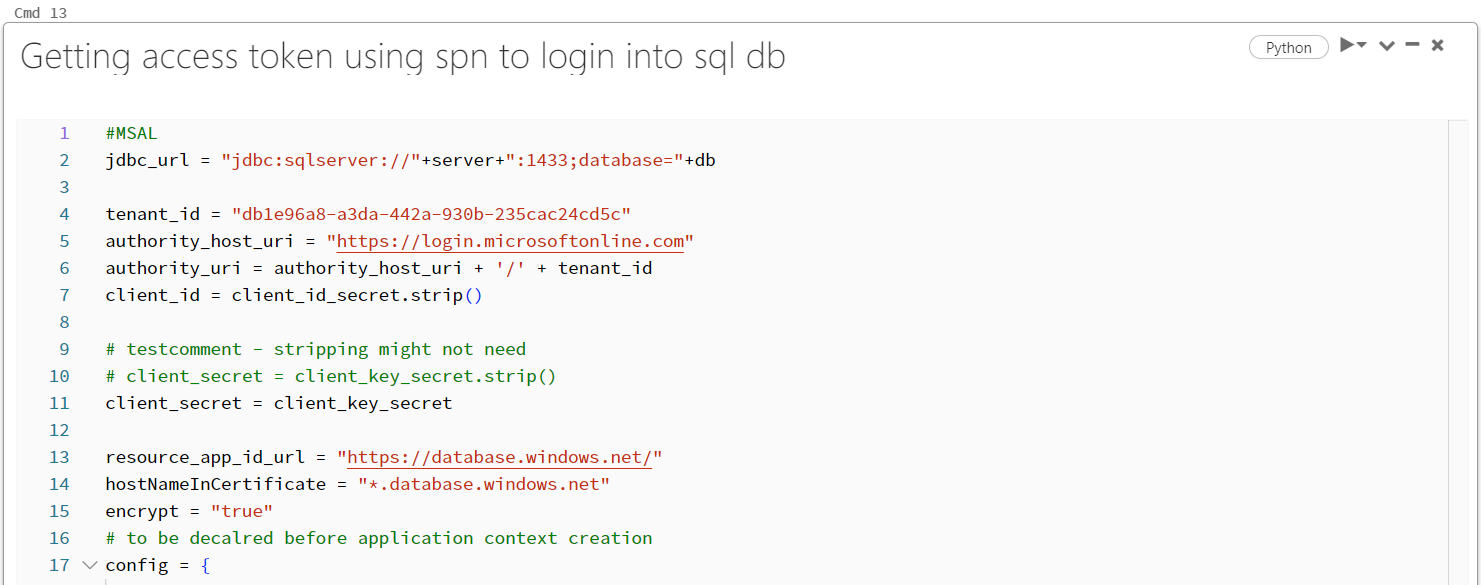


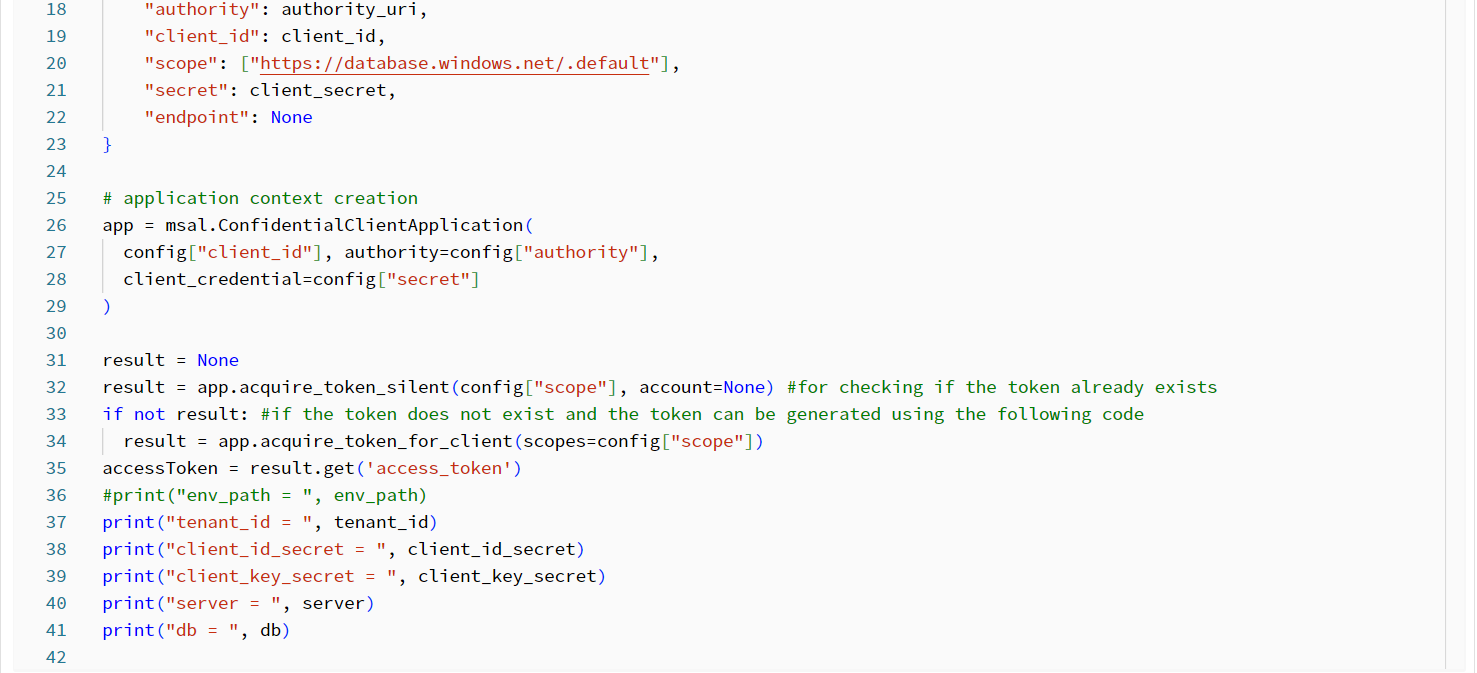




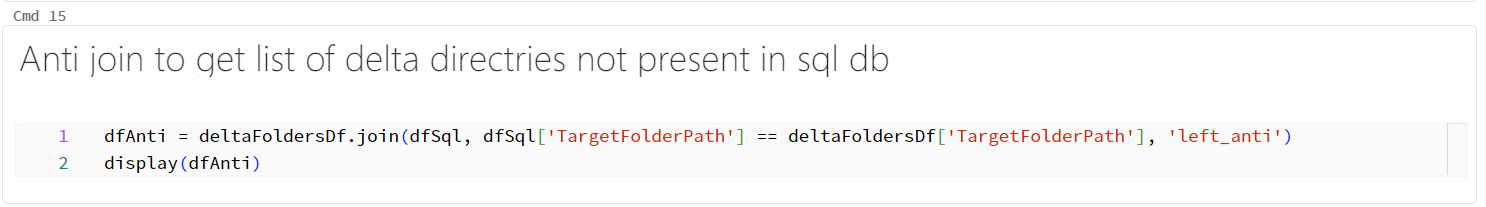








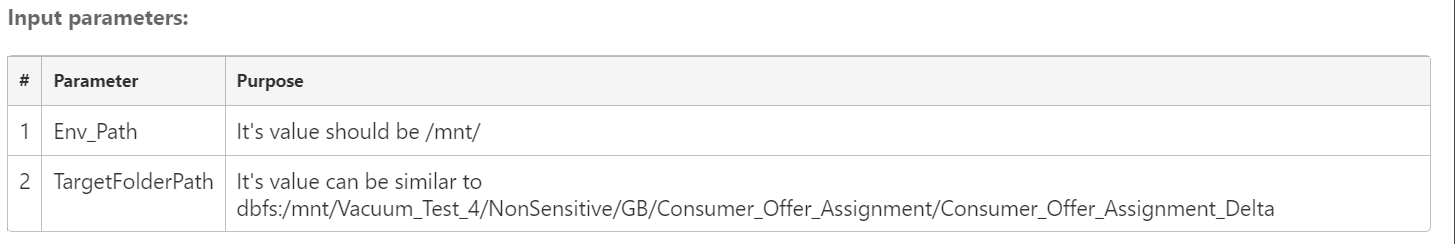


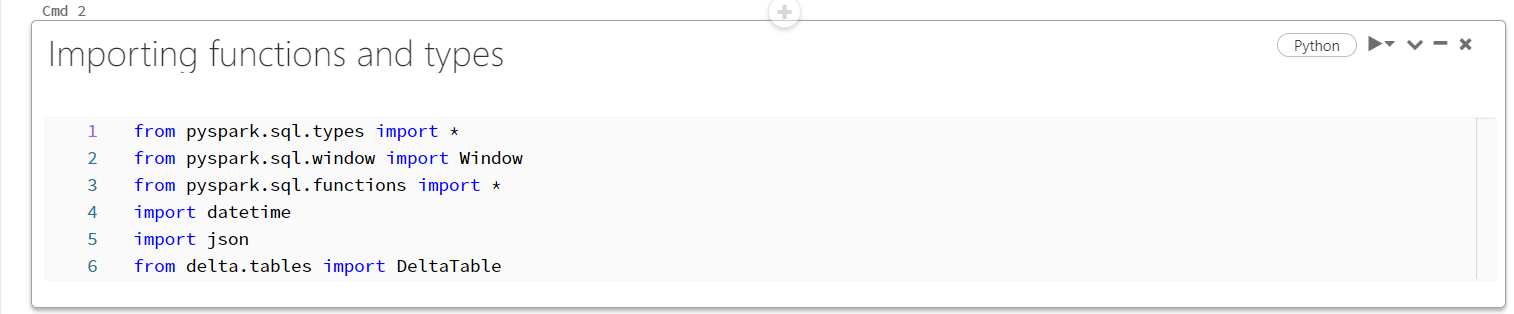


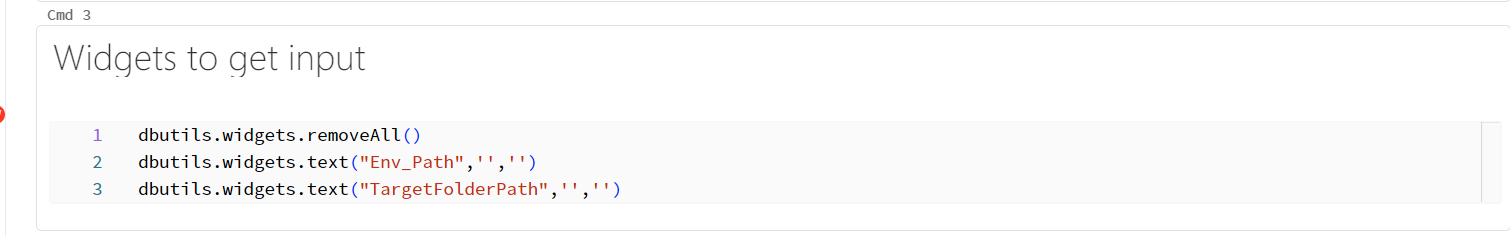




**NOTEBOOK: 02\_ADLS\_DELTA\_VACCUM\_NTBK**



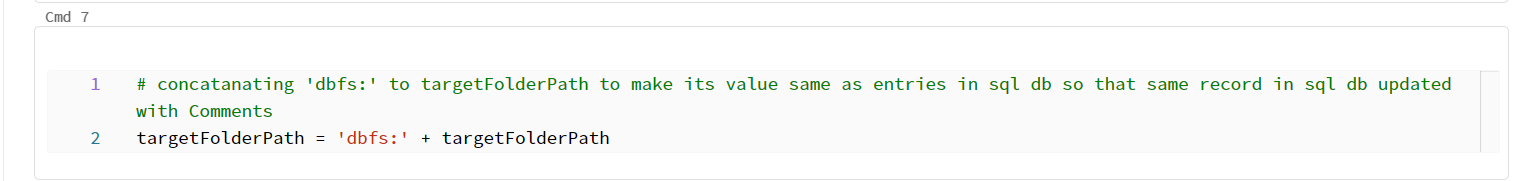


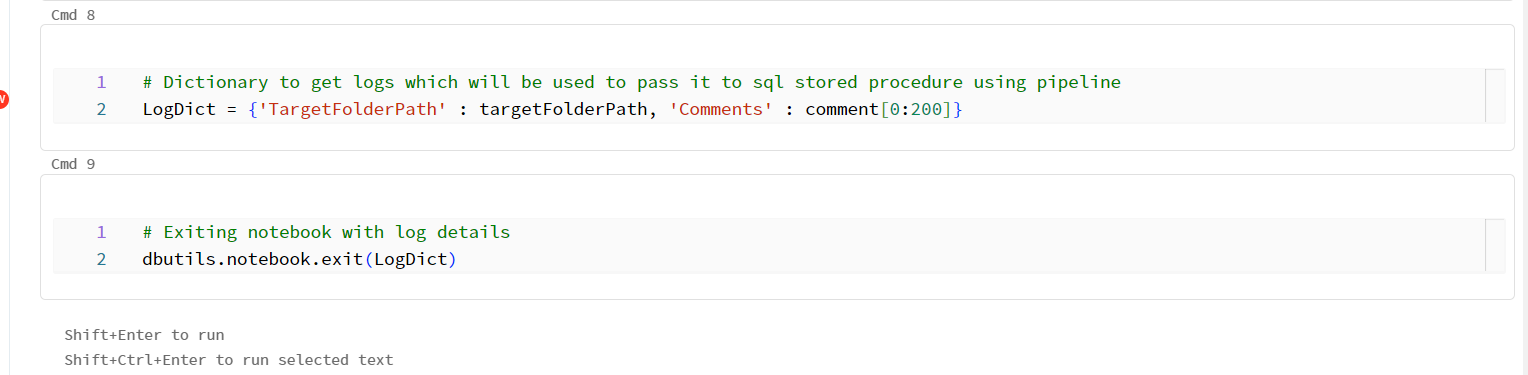






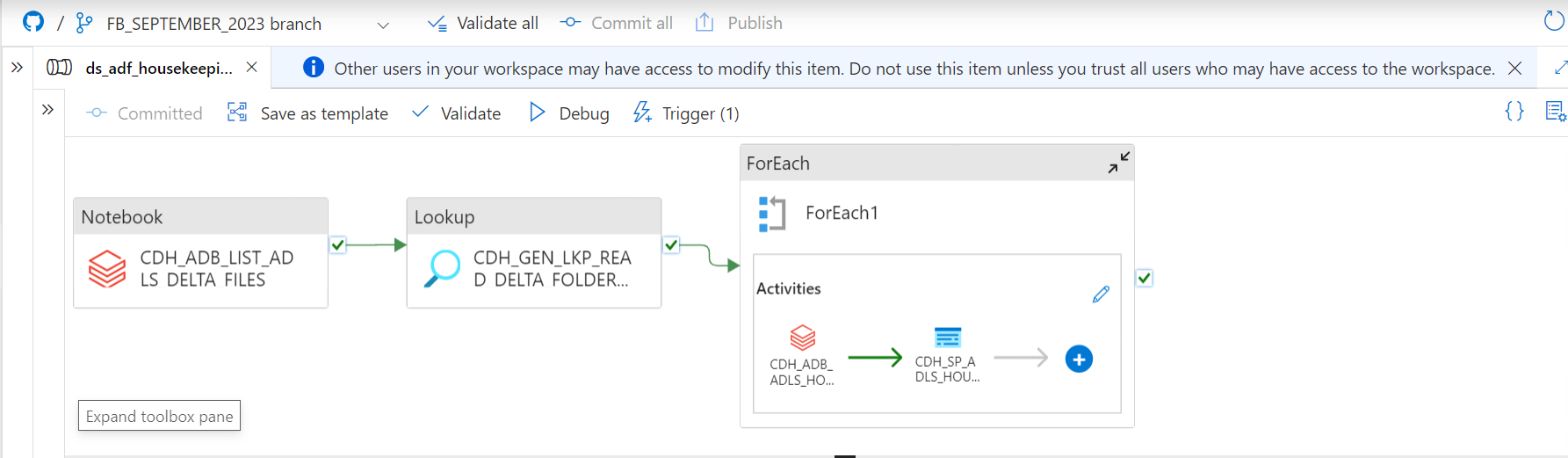






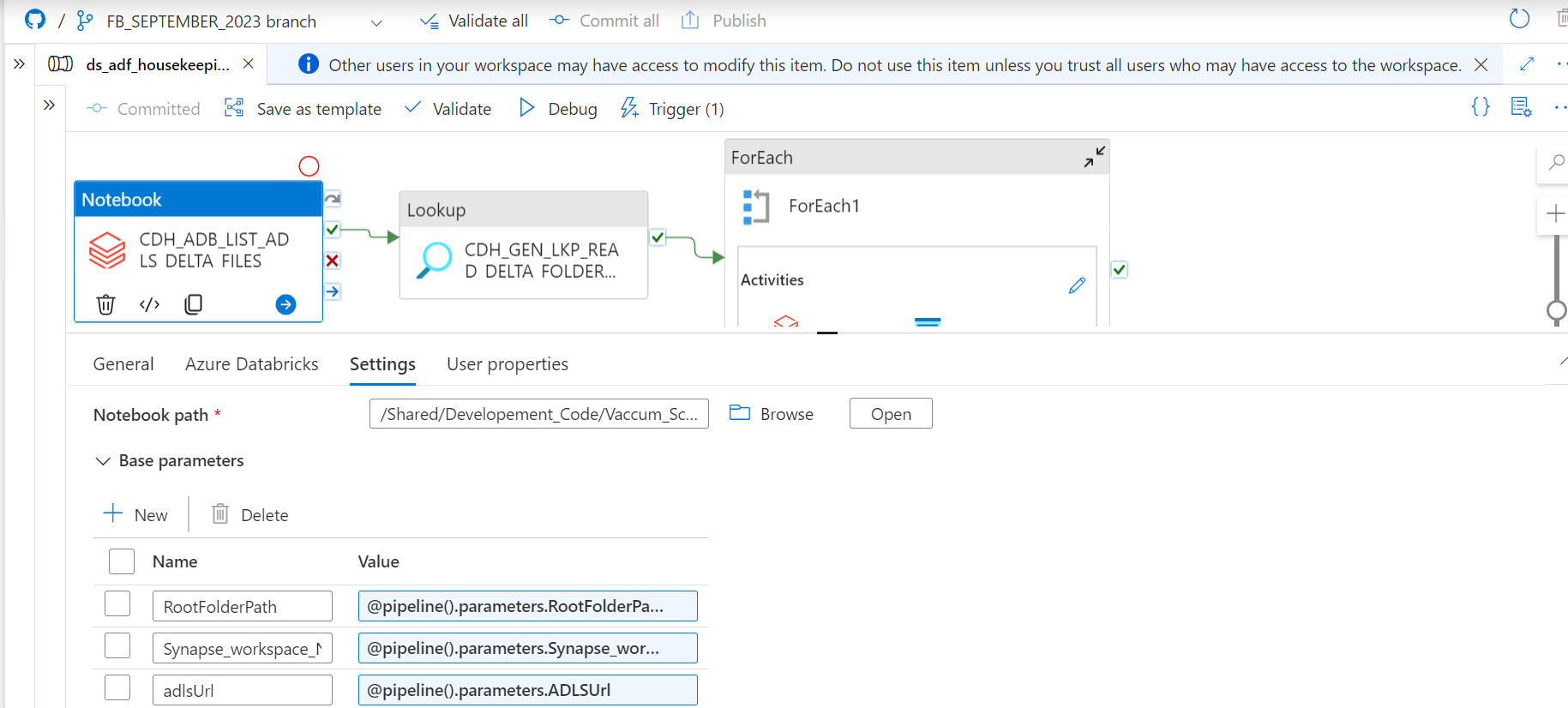
**Step4: Create Synapse pipeline:**

**The synapse pipeline activity looks like the below,**

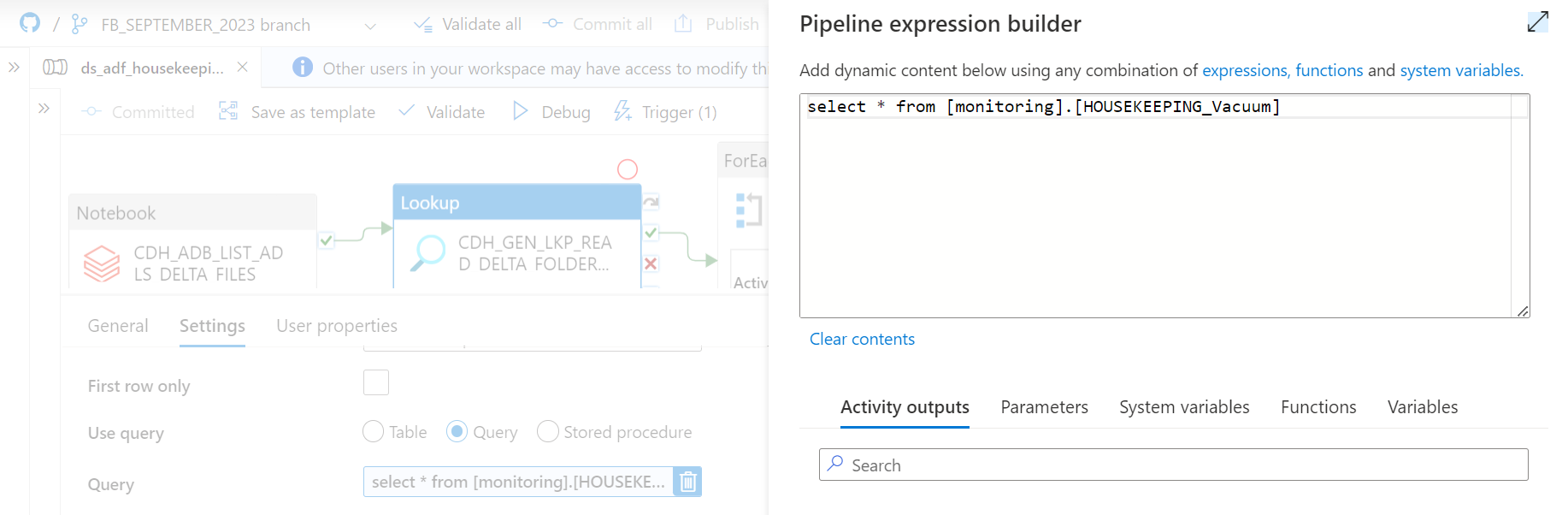


**There are three activities involved in it,**

Activity1: Notebook Activity: Call the NOTEBOOK: 01\_ADLS\_LIST\_DELTA\_UPDATE\_SQL and then execute the script and then perform Notebook to scan the given root folder path and find all delta directories and log these directories into the sql log table [monitoring].[Housekeeping\_Vacuum]



Activity2: Upon success of the activity1 and then Lookup activity will run the query into the SQL Server

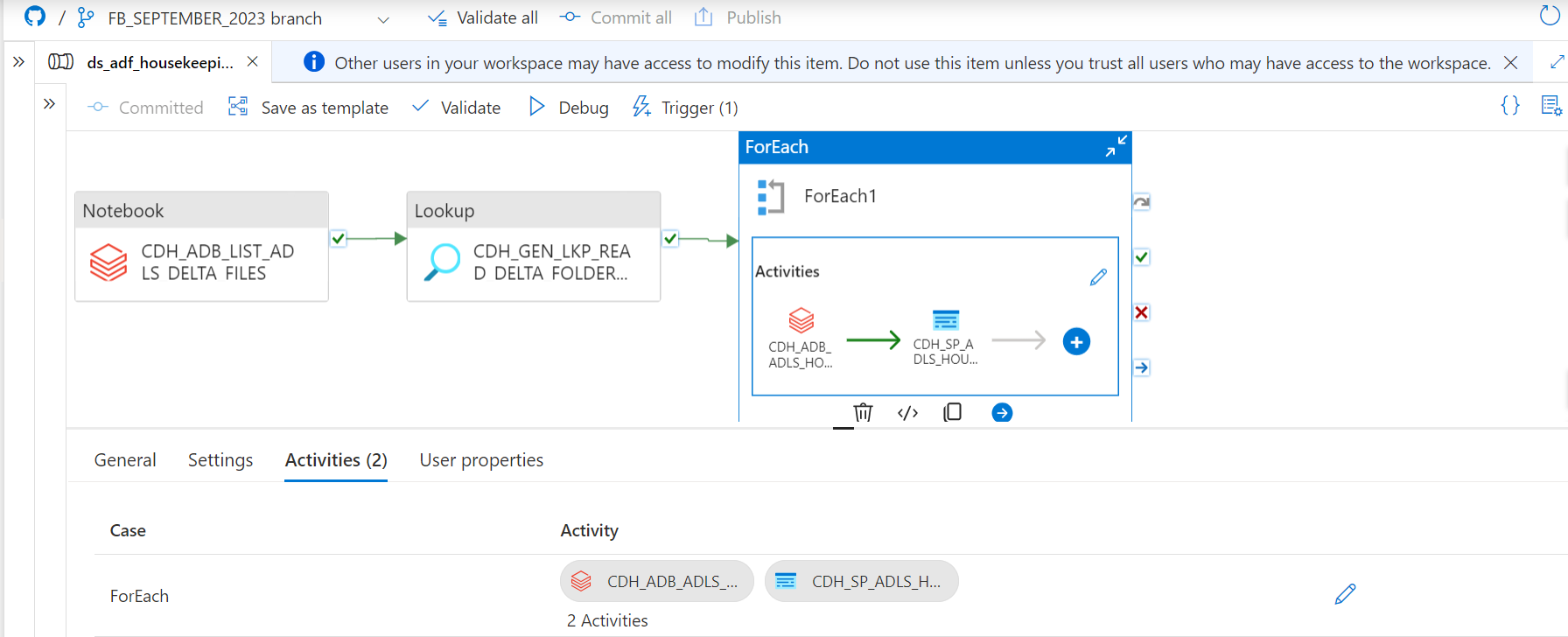


Activity 3: ForEach will include two sub activities.

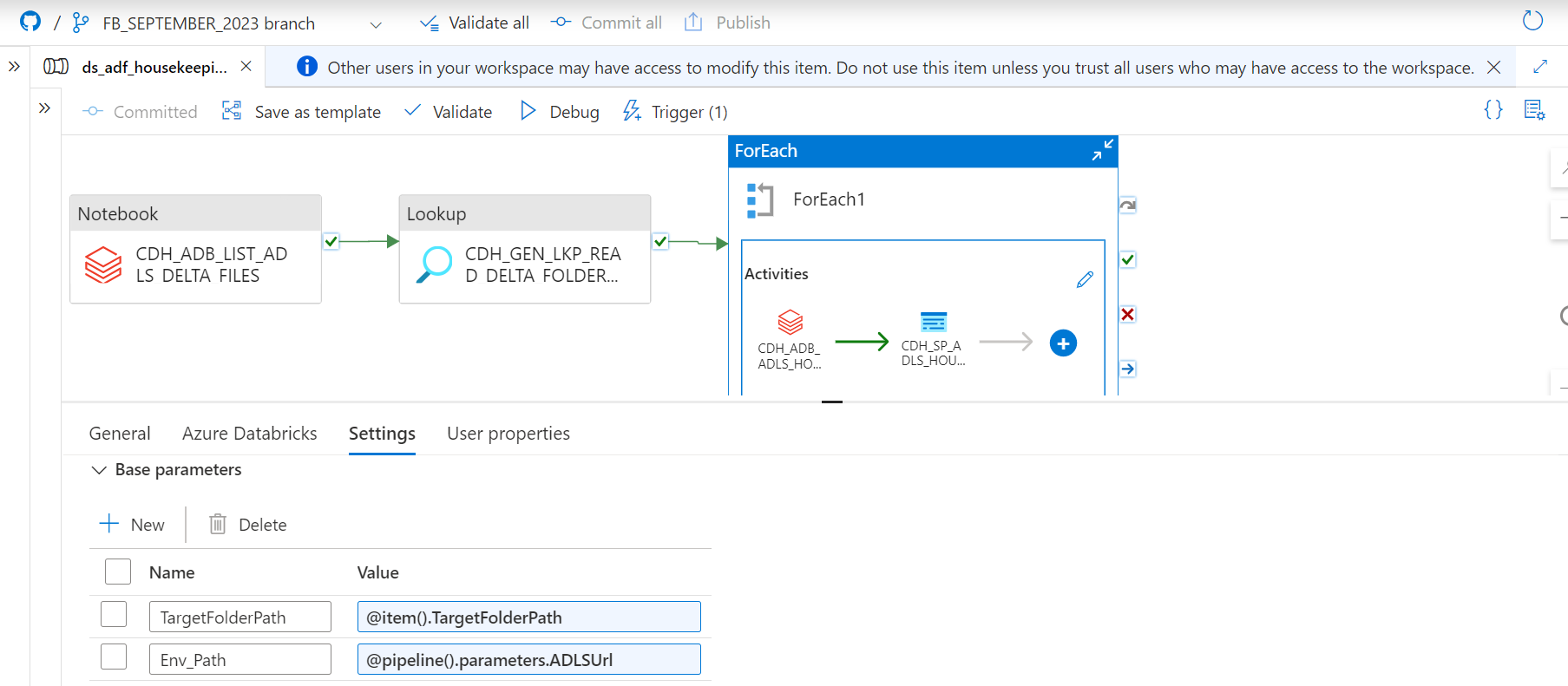
Sub activity 1: **NOTEBOOK: 02\_ADLS\_DELTA\_VACCUM\_NTBK**

Notebook to perform the VACUUM operation (for 7 days = 24 hrs X 7 days= 168 hrs) on given delta target path

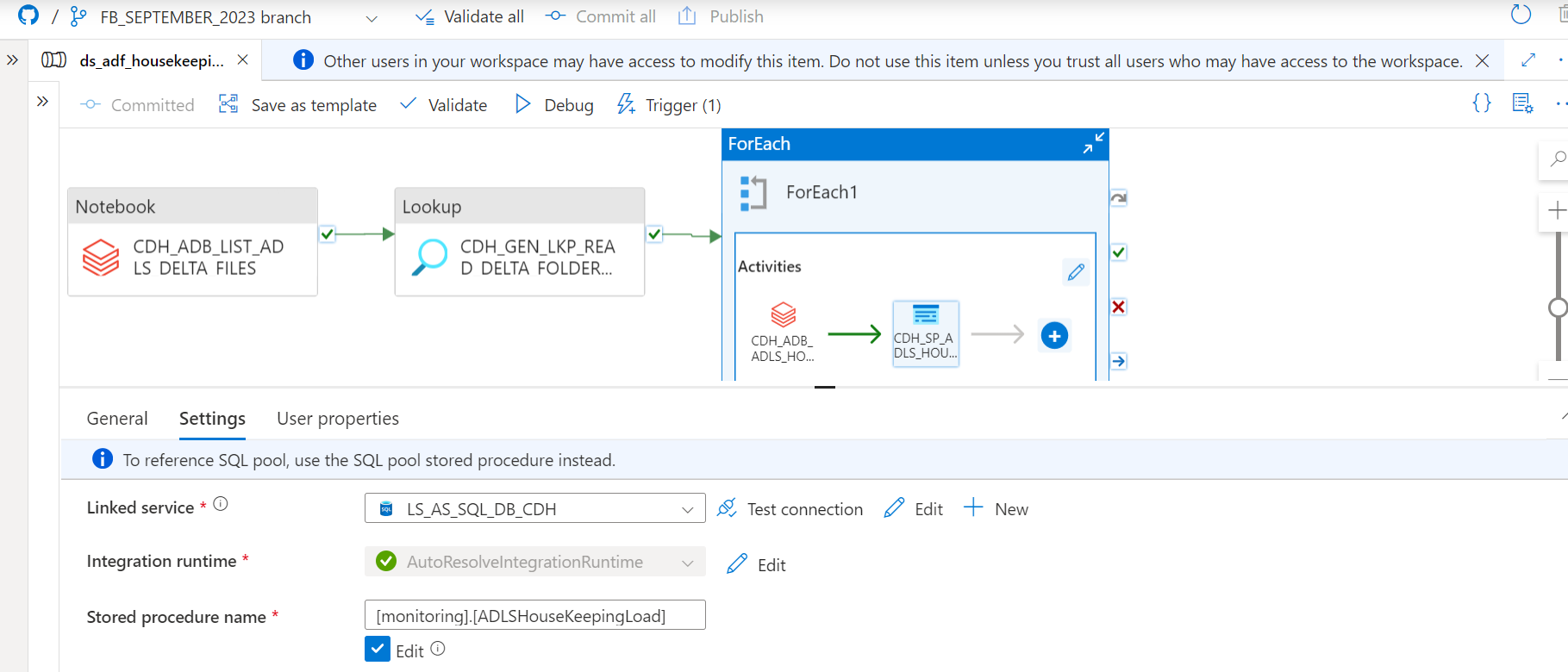
Sub activity2: Stored Procedure for replication purpose



Sub activity1: Vacuum Databricks notebook script



Sub activity2: Stored procedure

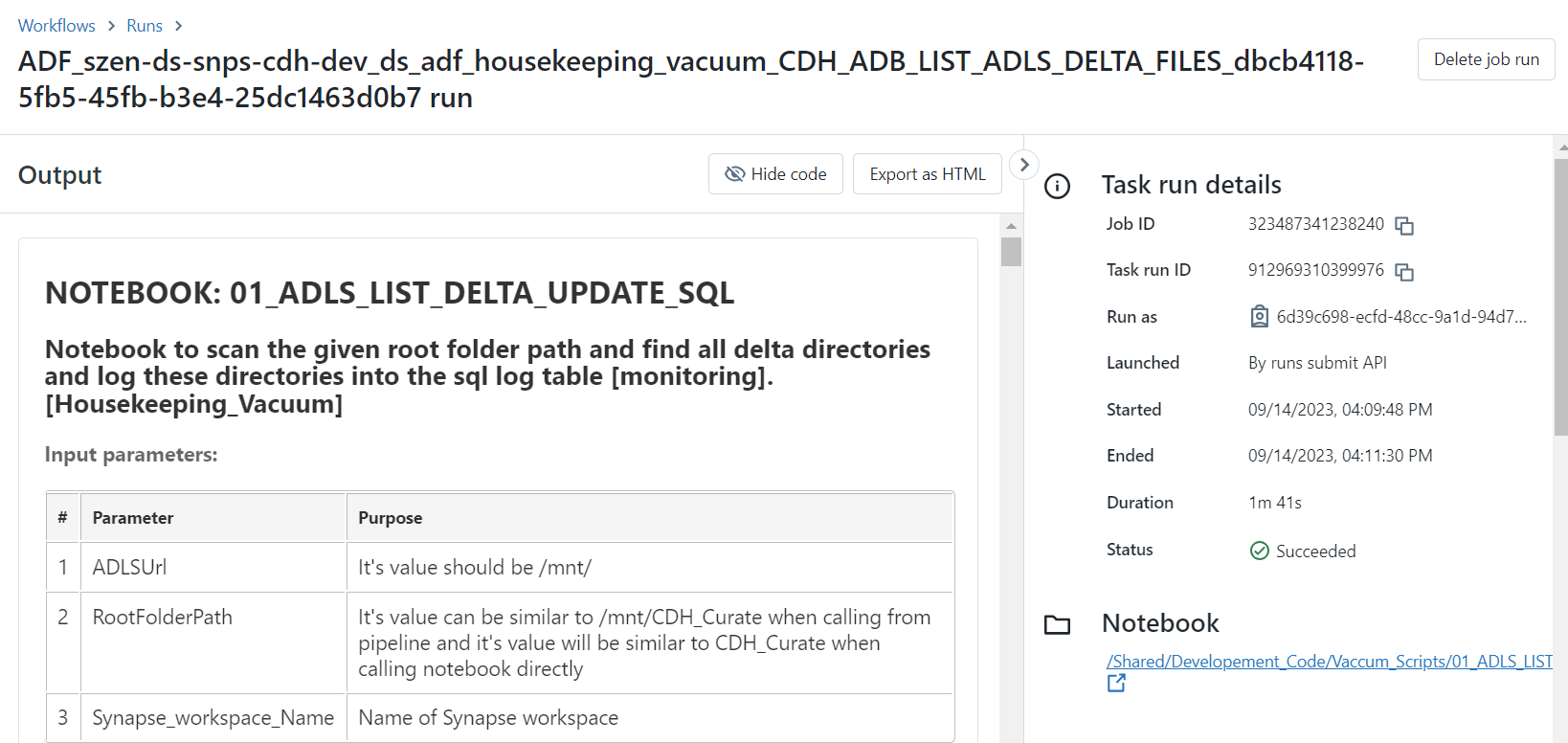


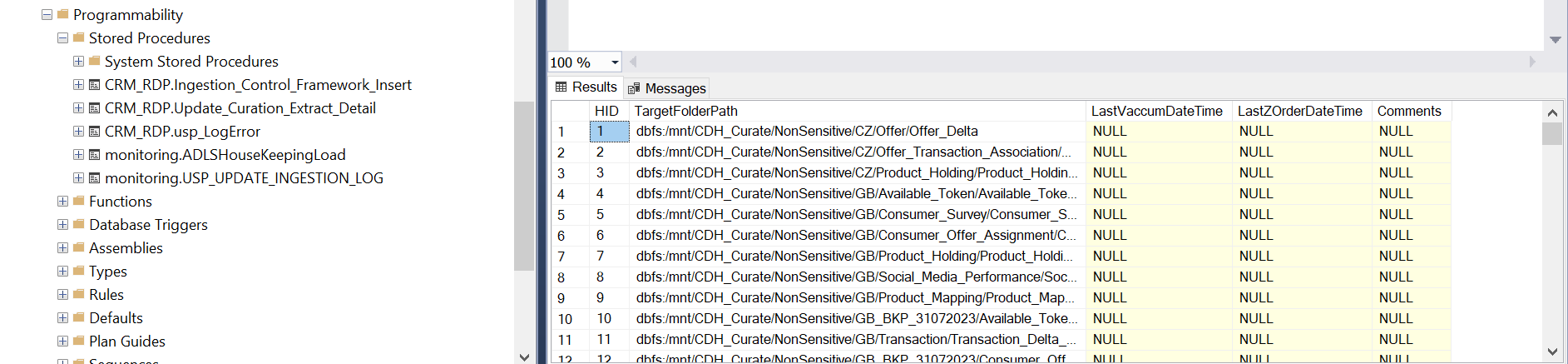
**Testing & Validation of the Synapse Pipeline and Vacuum Script:**

Go to the Synapse pipeline and then run it as a Debug mode.

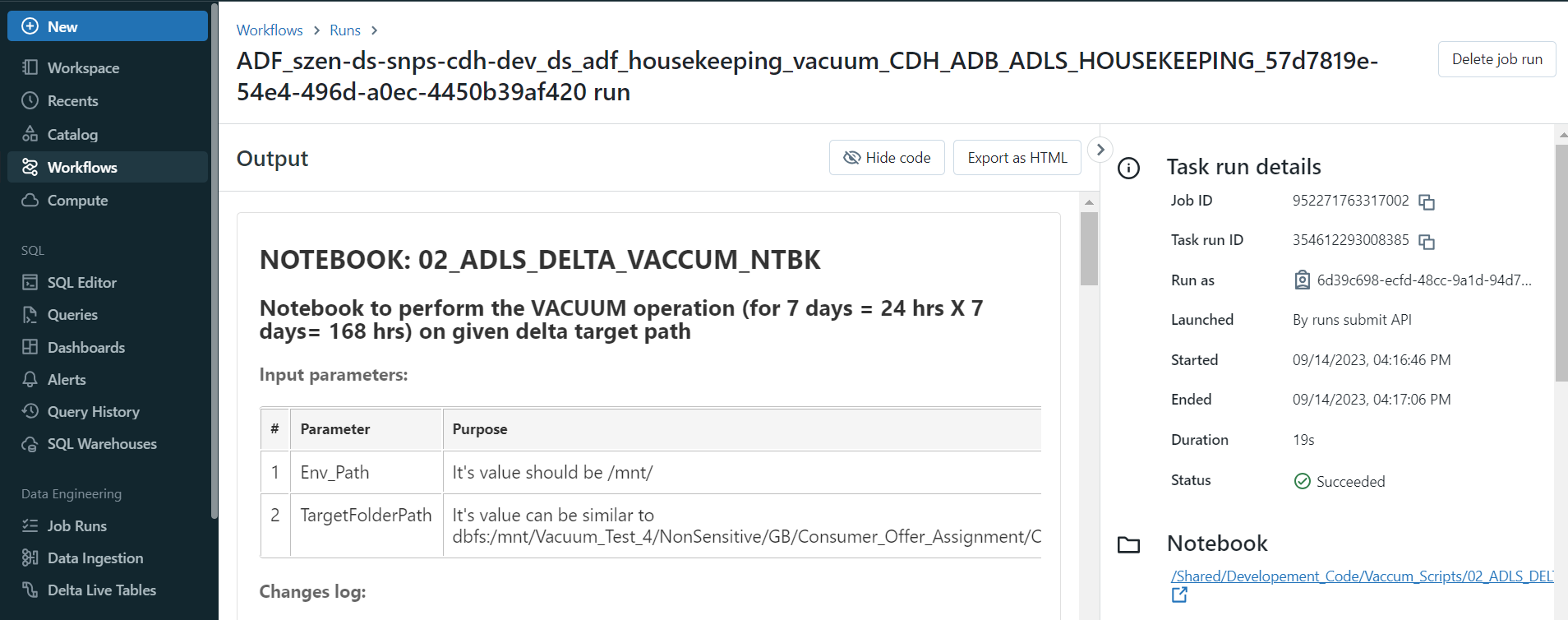
Once pipeline starts to run then the activity will happen one after another

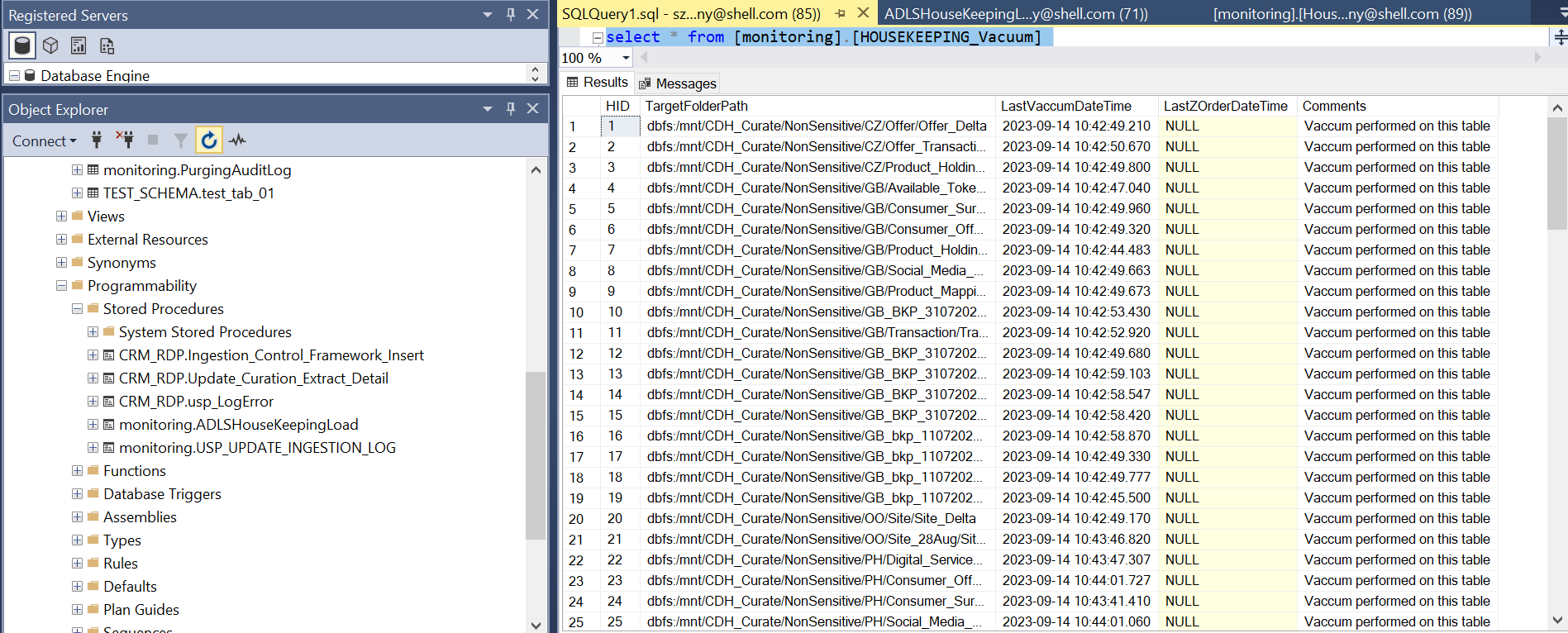
After the successful completion of the first script



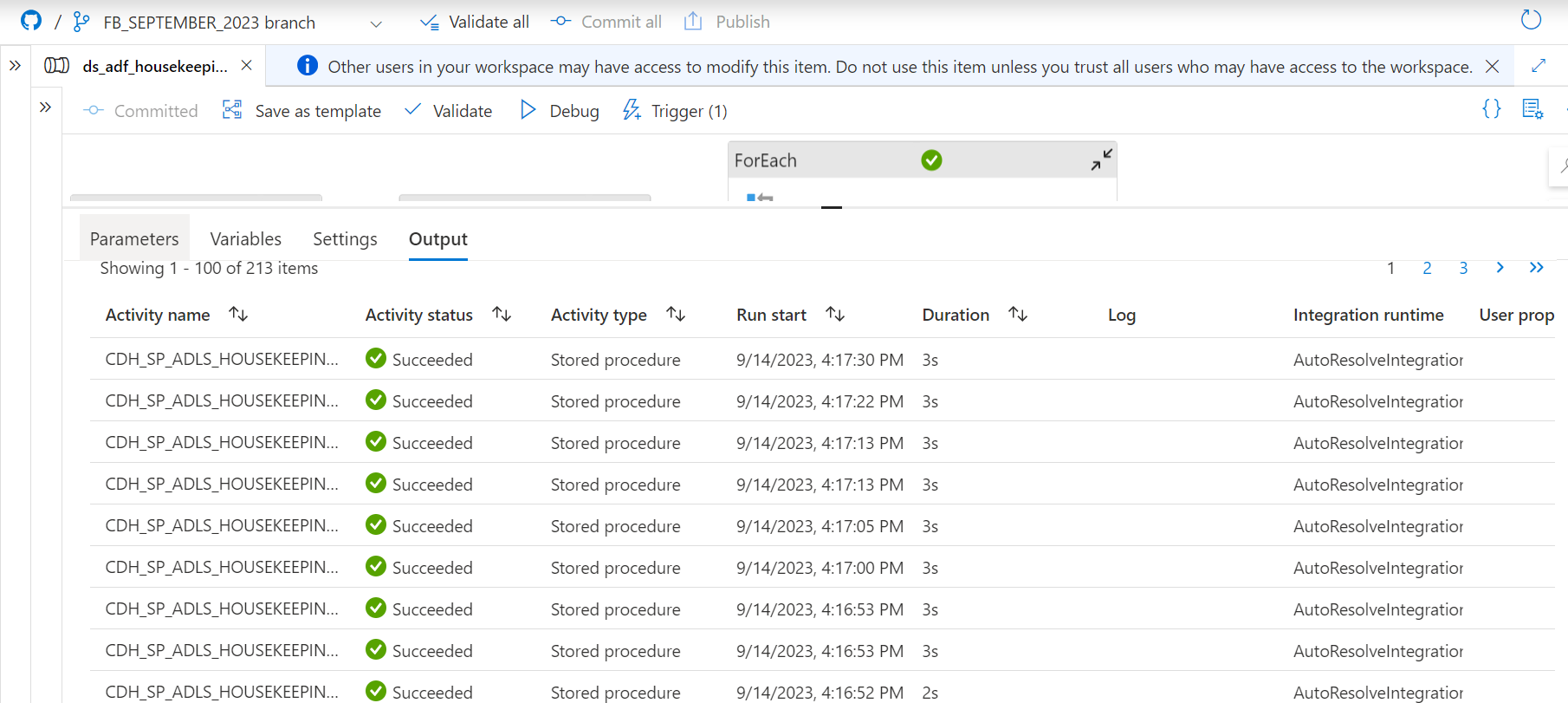


After successful completion of the second script :

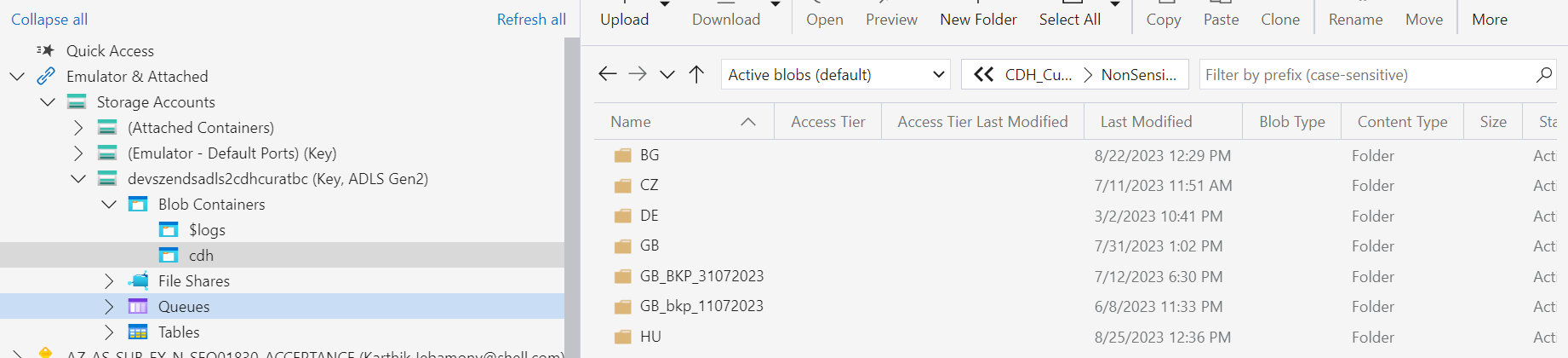




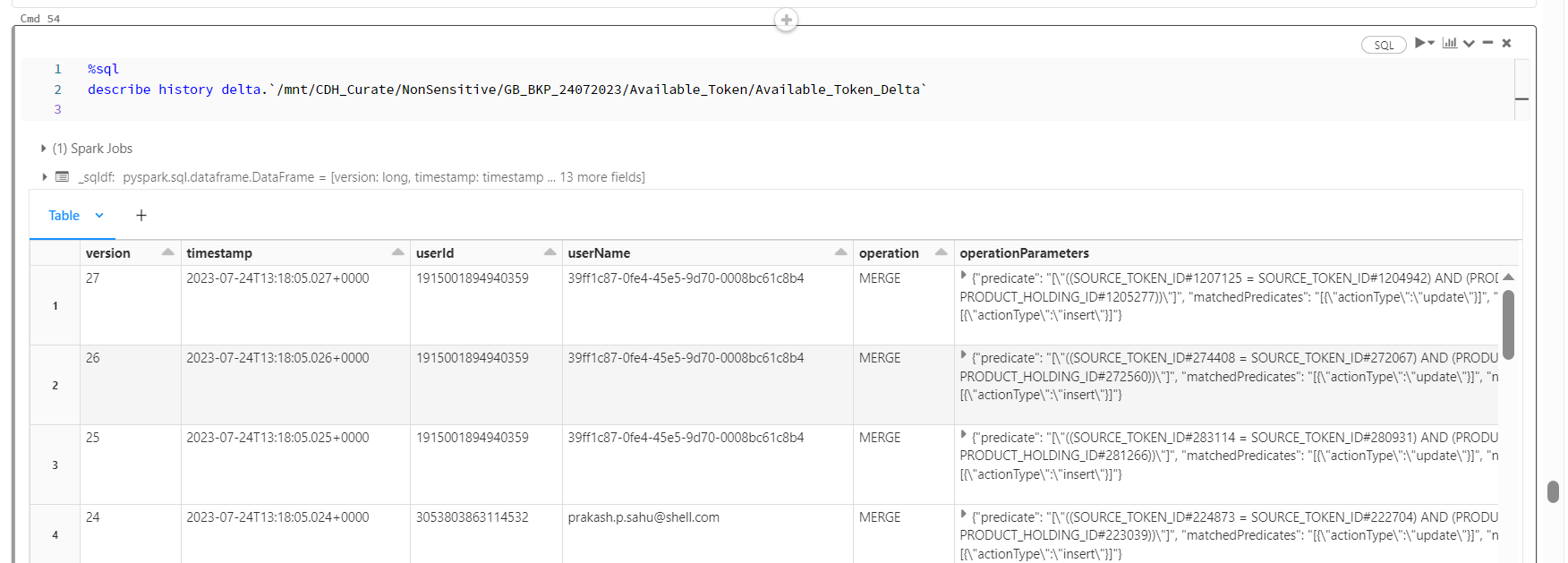
The pipeline successful completion status:

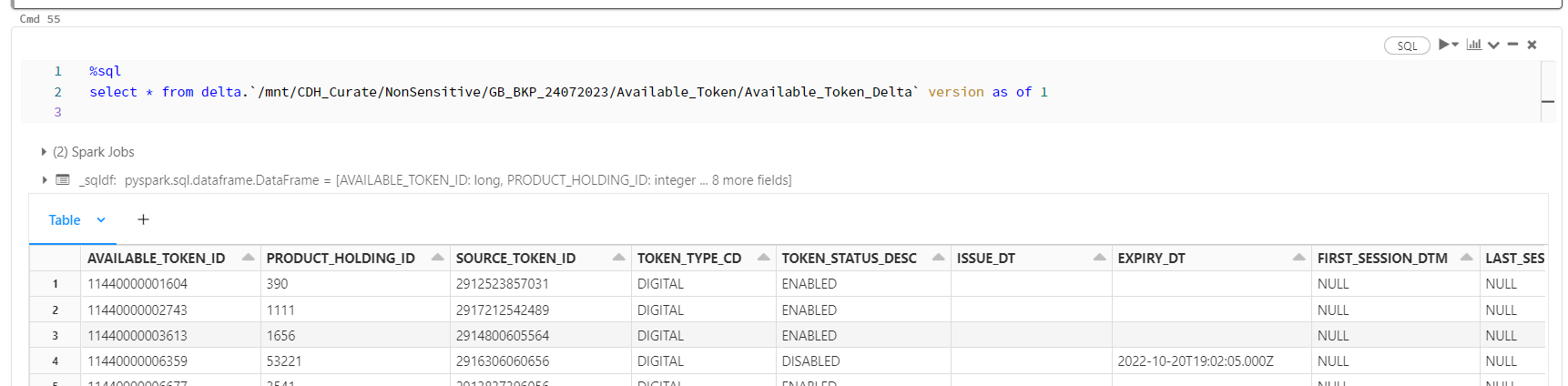


**Validation Process:**

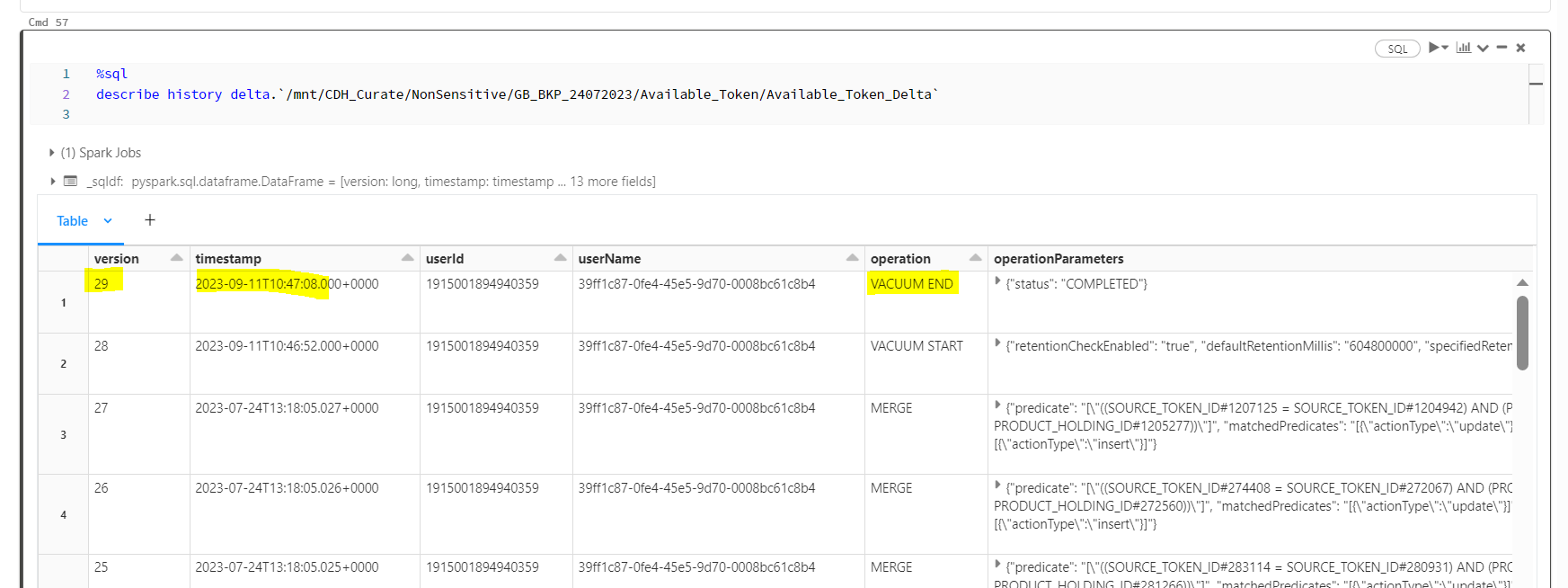


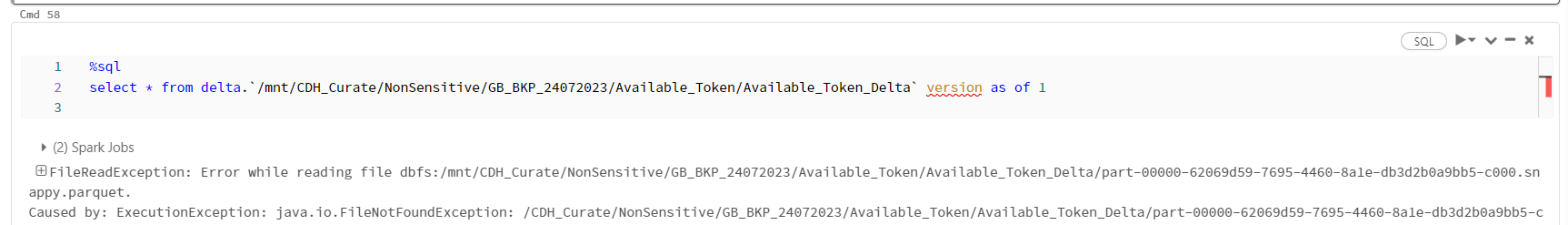
Before Vacuum operation





After Vacuum Operation:





**Conclusion:**

Thus, the vacuum is performed using Databricks script and Synapse pipeline and then SQL Server.

The vacuum process for 7 days once will significantly reduce the storage cost of the ADLS.

**Note:**

It is recommended that you set a retention interval to be at least 7 days, because old snapshots and uncommitted files can still be in use by concurrent readers or writers to the table. If VACUUM cleans up active files, concurrent readers can fail or, worse, tables can be corrupted when VACUUM deletes files that have not yet been committed. You must choose an interval that is longer than the longest running concurrent transaction and the longest period that any stream can lag behind the most recent update to the table.