

Lab 2 - Run interactive queries using serverless SQL pools

In this lab, you will learn how to work with files stored in the data lake and external file sources, through T-SQL statements executed by a serverless SQL pool in Azure Synapse Analytics. You will query Parquet files stored in a data lake, as well as CSV files stored in an external data store. Next, you will create Azure Active Directory security groups and enforce access to files in the data lake through Role-Based Access Control (RBAC) and Access Control Lists (ACLs).

After completing this lab, you will be able to:

- Query Parquet data with serverless SQL pools
- Create external tables for Parquet and CSV files
- Create views with serverless SQL pools
- Secure access to data in a data lake when using serverless SQL pools
- Configure data lake security using Role-Based Access Control (RBAC) and Access Control Lists (ACLs)

Lab setup and pre-requisites

Before starting this lab, ensure you have successfully completed the setup steps to create your lab environment.

Exercise 1: Querying a Data Lake Store using serverless SQL pools in Azure Synapse Analytics

Understanding data through data exploration is one of the core challenges faced today by data engineers and data scientists as well. Depending on the underlying structure of the data as well as the specific requirements of the exploration process, different data processing engines will offer varying degrees of performance, complexity, and flexibility.

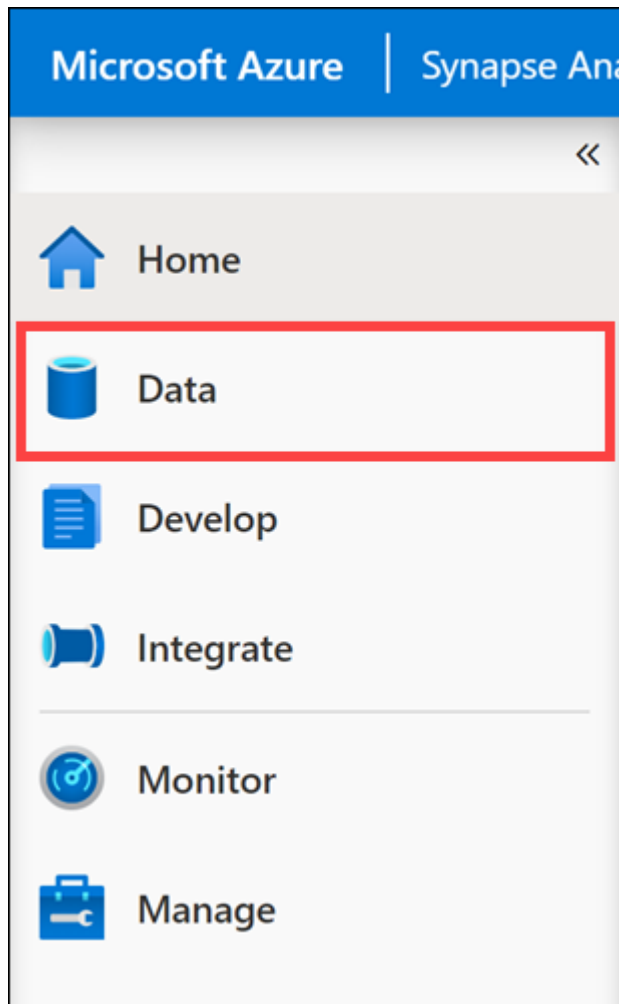
In Azure Synapse Analytics, you can use either SQL, Apache Spark for Synapse, or both. Which service you use mostly depends on your personal preference and expertise. When conducting data engineering tasks, both options can be equally valid in many cases. However, there are certain situations where harnessing the power of Apache Spark can help you overcome problems with the source data. This is because in a Synapse Notebook, you can import from a large number of free libraries that add functionality to your environment when working with data. There are other situations where it is much more convenient and faster using serverless SQL pool to explore the data, or to expose data in the data lake through a SQL view that can be accessed from external tools, like Power BI.

In this exercise, you will explore the data lake using both options.

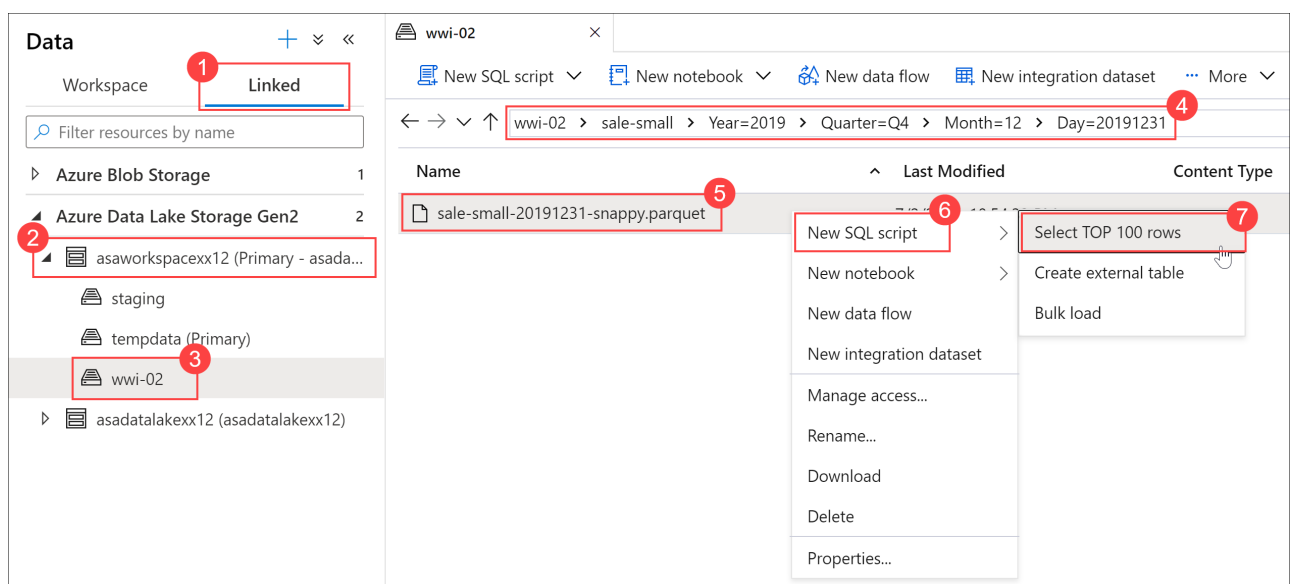
Task 1: Query sales Parquet data with serverless SQL pools

When you query Parquet files using serverless SQL pools, you can explore the data with T-SQL syntax.

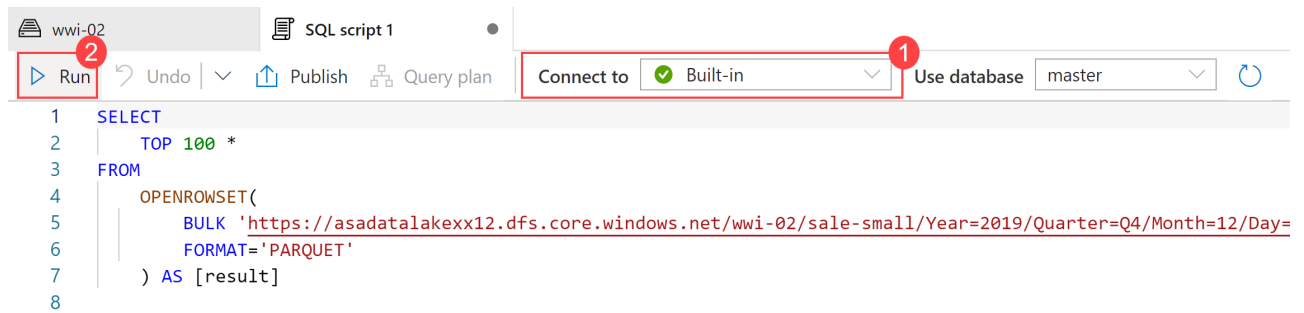
1. Open Synapse Studio (<https://web.azuresynapse.net/>), and then navigate to the **Data** hub.



- In the pane on the left, on the **Linked** tab, expand **Azure Data Lake Storage Gen2** and the **asaworkspacexxxxxx** primary ADLS Gen2 account, and select the **wwi-02** container
- In the **sale-small/Year=2019/Quarter=Q4/Month=12/Day=20191231** folder, right-click the **sale-small-20191231-snappy.parquet** file, select **New SQL script**, and then select **Select TOP 100 rows**.



- Ensure **Built-in** is selected in the **Connect to** dropdown list above the query window, and then run the query. Data is loaded by the serverless SQL endpoint and processed as if it was coming from any regular relational database.



The cell output shows the query results from the Parquet file.

1	SELECT
2	TOP 100 *
3	FROM
4	OPENROWSET(
5	BULK 'https://asadatalakex1.dfs.core.windows.net/wwi-02/sale-sma
6	FORMAT='PARQUET'
7) AS [result]
8	

TransactionId	CustomerId	ProductId	Quantity	Price
3e280434-9d5f...	102449	3173	2	26.4700000000..
3e280434-9d5f...	102449	2686	1	21.4600000000..
3e280434-9d5f...	102449	2335	3	16.6500000000..
3e280434-9d5f...	102449	337	3	24.1800000000..

5. Modify the SQL query to perform aggregates and grouping operations to better understand the data. Replace the query with the following, replacing *SUFFIX* with the unique suffix for your Azure Data Lake store and making sure that the file path in the OPENROWSET function matches the current file path:

```

SELECT
    TransactionDate, ProductId,
    CAST(SUM(ProfitAmount) AS decimal(18,2)) AS [(sum) Profit],
    CAST(AVG(ProfitAmount) AS decimal(18,2)) AS [(avg) Profit],
    SUM(Quantity) AS [(sum) Quantity]
FROM
    OPENROWSET(
        BULK 'https://asadatalakeSUFFIX.dfs.core.windows.net/wwi-02/sale-
small/Year=2019/Quarter=Q4/Month=12/Day=20191231/sale-small-20191231-
snappy.parquet',

```

```

    FORMAT='PARQUET'
  ) AS [r] GROUP BY r.TransactionDate, r.ProductId;

```

1

I

2

SELECT

3

TransactionDate, ProductId,

4

CAST(SUM(ProfitAmount) AS decimal(18,2)) AS [(sum) Profit],

5

CAST(AVG(ProfitAmount) AS decimal(18,2)) AS [(avg) Profit],

6

SUM(Quantity) AS [(sum) Quantity]

7

FROM

8

OPENROWSET(

9

BULK 'https://asadatalakexx12.dfs.core.windows.net/wwi-02/sale-small/Year=2019/Quarter=Q4/Month=12/Day=20191231/sale

10

FORMAT='PARQUET'

11

) AS [r] GROUP BY r.TransactionDate, r.ProductId;

12

results

Messages

view

Table

Chart

Export results

Search

TransactionDate	ProductId	(sum) Profit	(avg) Profit	(sum) Quantity
20191231	1	64820.25	29.83	5335
20191231	2	71955.68	26.17	6866
20191231	4	49988.43	19.42	6417

6. Let's move on from this single file from 2019 and transition to a newer data set. We want to figure out how many records are contained within the Parquet files for all 2019 data. This information is important for planning how we optimize for importing the data into Azure Synapse Analytics. To do this, we'll replace the query with the following (be sure to update the suffix of your data lake in the BULK statement):

```

SELECT
    COUNT(*)
FROM
    OPENROWSET(
        BULK 'https://asadatalakeSUFFIX.dfs.core.windows.net/wwi-02/sale-
small/Year=2019/*/*/*/*',
        FORMAT='PARQUET'
    ) AS [r];

```

Notice how we updated the path to include all Parquet files in all subfolders of *sale-small/Year=2019*.

The output should be **4124857** records.

Task 2: Create an external table for 2019 sales data

Rather than creating a script with OPENROWSET and a path to the root 2019 folder every time we want to query the Parquet files, we can create an external table.

1. In Synapse Studio, return to the **wwi-02** tab, which should still be showing the contents of the *sale-small/Year=2019/Quarter=Q4/Month=12/Day=20191231* folder.

2. Right-click the **sale-small-20191231-snappy.parquet** file, select **New SQL script**, and then select **Create external table**. In the New external table dialog box, click **Continue**.
3. Make sure **Built-in** is selected for the **SQL pool**. Then, under **Select a database**, select **+ New** and create a database named **demo**, and click **Create**. For **External table name**, enter **All2019Sales**. Finally, under **Create external table**, ensure **Using SQL script** is selected, and then select **Open Script** to generate the SQL script.

New external table

Select target database
[Learn more](#)

Select SQL pool* ①

✓ Built-in

Select a database* ②

demo

External table name ③

All2019Sales

Create external table

☐ Automatically ☒ Using SQL script ④

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

⑤ Open script Back Cancel

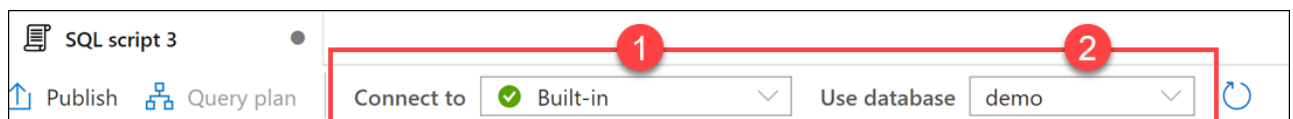
Note: The **Properties** pane for the script opens automatically. You can close it by using the **Properties** button above it on the right to make it easier to work with the script.

The generated script contains the following components:

- **1)** The script begins with creating the *SynapseParquetFormat* external file format with a *FORMAT_TYPE* of *PARQUET*.
 - **2)** Next, the external data source is created, pointing to the *wwi-02* container of the data lake storage account.
 - **3)** The CREATE EXTERNAL TABLE WITH statement specifies the file location and refers to the new external file format and data source created above.
 - **4)** Finally, we select the top 100 results from the *2019Sales* external table.
4. In the CREATE EXTERNAL TABLE statement, in the **[TransactionId] varchar(8000)** line, change 8000 to 4000 and add *COLLATE Latin1_General_100_BIN2_UTF8*; and replace the *LOCATION* value with *sale-small/Year=2019/**/*.parquet* so that the statement becomes similar to the following (except with your unique resource SUFFIX):

```
CREATE EXTERNAL TABLE All2019Sales (
  [TransactionId] varchar(4000) COLLATE Latin1_General_100_BIN2_UTF8,
  [CustomerId] int,
  [ProductId] smallint,
  [Quantity] smallint,
  [Price] numeric(38,18),
  [TotalAmount] numeric(38,18),
  [TransactionDate] int,
  [ProfitAmount] numeric(38,18),
  [Hour] smallint,
  [Minute] smallint,
  [StoreId] smallint
)
WITH (
  LOCATION = 'sale-small/Year=2019/**/*.parquet',
  DATA_SOURCE = [wwi-02_asadatalakeSUFFIX_dfs_core_windows_net],
  FILE_FORMAT = [SynapseParquetFormat]
)
GO
```

5. Make sure the script is connected to the serverless SQL pool (**Built-in**) and that the **demo** database is selected in the **Use database** list (use the ... button to see the list if the pane is too small to display it, and then use the ↺ button to refresh the list if needed).



6. Run the modified script.

After running the script, we can see the output of the SELECT query against the **All2019Sales** external table. This displays the first 100 records from the Parquet files located in the *YEAR=2019* folder.

```

27     LOCATION = 'sale-small/Year=2019/**/**/*.parquet',
28     DATA_SOURCE = [wwi-02_asadatalakeinaday84_dfs_core_windows_net],
29     FILE_FORMAT = [SynapseParquetFormat]
30 )
31 GO
32
33 SELECT TOP 100 * FROM All2019Sales
34 GO
35
36

```

Results Messages

Select Query 4 View Table Chart Export results

Search

TransactionId	CustomerId	ProductId	Quantity	Price	TotalAmount	TransactionDate	ProfitAmount	Hour
5fc634ab-e910...	1	2965	2	35.4200000000...	70.8400000000...	20190103	20.7000000000...	9
5fc634ab-e910...	1	3986	3	34.6400000000...	103.9200000000...	20190103	31.5900000000...	9
5fc634ab-e910...	1	4648	1	23.6000000000...	23.6000000000...	20190103	8.2500000000...	9
5fc634ab-e910...	1	1359	3	29.8200000000...	89.4600000000...	20190103	29.6700000000...	9
5fc634ab-e910...	1	2726	4	33.2200000000...	132.8800000000...	20190103	38.2000000000...	9
5fc634ab-e910...	1	384	4	31.6100000000...	126.4400000000...	20190103	29.3200000000...	9
5fc634ab-e910...	1	413	4	31.5900000000...	126.3600000000...	20190103	40.0000000000...	9
5fc634ab-e910...	1	2615	1	39.4000000000...	39.4000000000...	20190103	11.3700000000...	9

Tip: If an error occurs because of a mistake in your code, you should delete any resources that were successfully created before trying again. You can do this by running the appropriate DROP statements, or by switching to the **Workspace** tab, refreshing the list of **Databases**, and deleting the objects in the **demo** database.

Task 3: Create an external table for CSV files

Tailwind Traders found an open data source for country population data that they want to use. They do not want to merely copy the data since it is regularly updated with projected populations in future years.

You decide to create an external table that connects to the external data source.

1. Replace the SQL script you ran in the previous task with the following code:

```

IF NOT EXISTS (SELECT * FROM sys.symmetric_keys) BEGIN
    declare @password nvarchar(400) = CAST(newid() as VARCHAR(400));
    EXEC('CREATE MASTER KEY ENCRYPTION BY PASSWORD = ''' + @password + ''')
END

CREATE DATABASE SCOPED CREDENTIAL [sqlondemand]
WITH IDENTITY='SHARED ACCESS SIGNATURE',
SECRET = 'sv=2018-03-28&ss=bf&srt=sco&sp=r1&st=2019-10-
14T12%3A10%3A25Z&se=2061-12-
31T12%3A10%3A00Z&sig=KlSU2u1lCscyTS0An0nozEpo4t05JAgGBvw%2FJX2lguw%3D'
GO

-- Create external data source secured using credential
CREATE EXTERNAL DATA SOURCE SqlOnDemandDemo WITH (
    LOCATION = 'https://sqlondemandstorage.blob.core.windows.net',
    CREDENTIAL = sqlondemand
);

```

```

GO

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

CREATE EXTERNAL TABLE [population]
(
    [country_code] VARCHAR (5) COLLATE Latin1_General_BIN2,
    [country_name] VARCHAR (100) COLLATE Latin1_General_BIN2,
    [year] smallint,
    [population] bigint
)
WITH (
    LOCATION = 'csv/population/population.csv',
    DATA_SOURCE = SqlOnDemandDemo,
    FILE_FORMAT = QuotedCsvWithHeader
);
GO

```

At the top of the script, we create a MASTER KEY with a random password. Next, we create a database-scoped credential for the containers in the external storage account using a shared access signature (SAS) for delegated access. This credential is used when we create the **SqlOnDemandDemo** external data source that points to the location of the external storage account that contains the population data:

```

1  IF NOT EXISTS (SELECT * FROM sys.symmetric_keys) BEGIN
2      declare @password nvarchar(400) = CAST(newid() as VARCHAR(400));
3      EXEC('CREATE MASTER KEY ENCRYPTION BY PASSWORD = ''' + @password + ''')
4  END
5
6  CREATE DATABASE SCOPED CREDENTIAL [sqlondemand]
7  WITH IDENTITY='SHARED ACCESS SIGNATURE'
8  SECRET = 'sv=2018-03-28&ss=bf&srt=sco&sp=r&st=2019-10-14T12%3A10%3A25Z&se=2061-12-31T12%3A10%3A00Z&sig=K1SU2u1lCscyTS0An0nc'
9  GO
10
11  -- Create external data source secured using credential
12  CREATE EXTERNAL DATA SOURCE SqlOnDemandDemo WITH (
13      LOCATION = 'https://sqlondemandstorage.blob.core.windows.net'
14      CREDENTIAL = sqlondemand
15  );
16  GO

```

Database-scoped credentials are used when any principal calls the OPENROWSET function with a DATA_SOURCE or selects data from an external table that doesn't access public files. The database scoped credential doesn't need to match the name of storage account because it will be explicitly used in the DATA SOURCE that defines the storage location.

In the next part of the script, we create an external file format called **QuotedCsvWithHeader**. Creating an external file format is a prerequisite for creating an External Table. By creating an External File

Format, you specify the actual layout of the data referenced by an external table. Here we specify the CSV field terminator, string delimiter, and set the FIRST_ROW value to 2 since the file contains a header row:

```
17
18 CREATE EXTERNAL FILE FORMAT QuotedCsvWithHeader
19 WITH (
20     FORMAT_TYPE = DELIMITEDTEXT,
21     FORMAT_OPTIONS (
22         FIELD_TERMINATOR = ',',
23         STRING_DELIMITER = '"',
24         FIRST_ROW = 2
25     )
26 );
27 GO
```

Finally, at the bottom of the script, we create an external table named **population**. The WITH clause specifies the relative location of the CSV file, points to the data source created above, as well as the *QuotedCsvWithHeader* file format:

```
28
29 CREATE EXTERNAL TABLE [population]
30 (
31     [country_code] VARCHAR (5) COLLATE Latin1_General_BIN2,
32     [country_name] VARCHAR (100) COLLATE Latin1_General_BIN2,
33     [year] smallint,
34     [population] bigint
35 )
36 WITH (
37     LOCATION = 'csv/population/population.csv',
38     DATA_SOURCE = SqlOnDemandDemo,
39     FILE_FORMAT = QuotedCsvWithHeader
40 );
41 GO
```

2. Run the script.

Note that there are no data results for this query.

3. Replace the SQL script with the following to select from the population external table, filtered by 2019 data where the population is greater than 100 million:

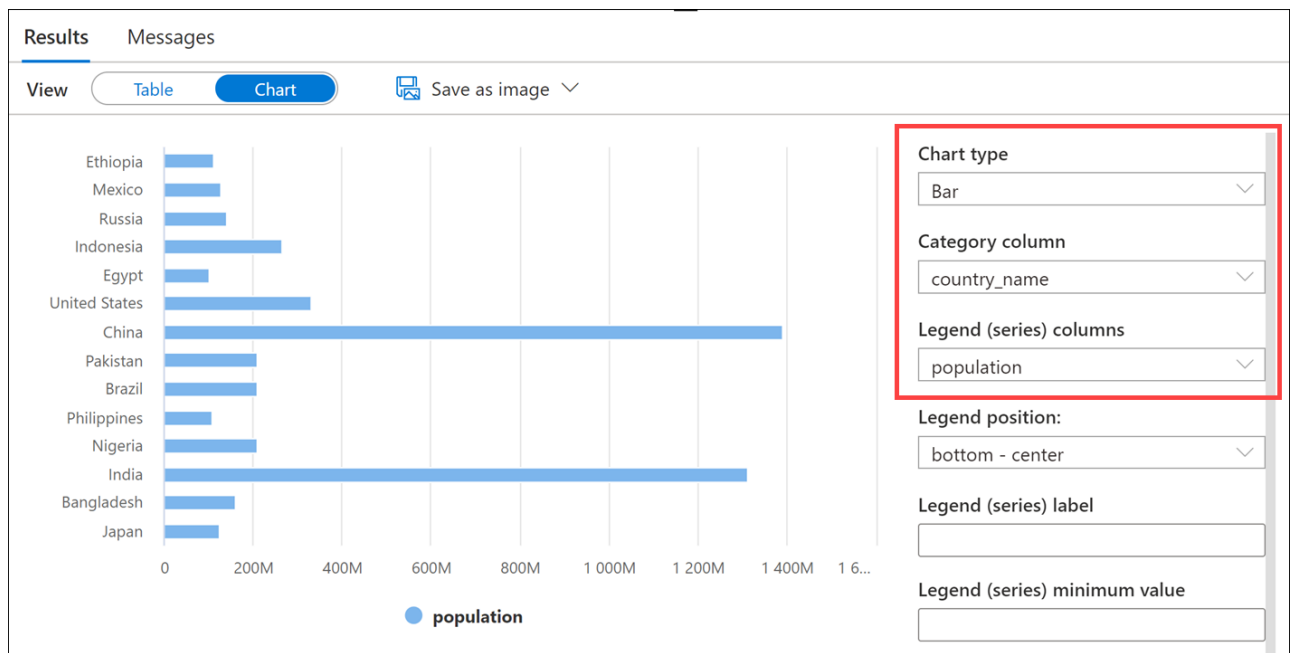
```
SELECT [country_code]
      ,[country_name]
      ,[year]
      ,[population]
```

```
FROM [dbo].[population]
WHERE [year] = 2019 and population > 100000000
```

4. Run the script.

5. In the query results, select the **Chart** view, then configure it as follows:

- **Chart type:** Bar
- **Category column:** country_name`
- **Legend (series) columns:** population
- **Legend position:** bottom - center



Task 4: Create a view with a serverless SQL pool

Let's create a view to wrap a SQL query. Views allow you to reuse queries and are needed if you want to use tools, such as Power BI, in conjunction with serverless SQL pools.

1. In the **Data** hub, on the **Linked** tab, in the **Azure Data Lake Storage Gen2/asaworkspacexxxxxx/wwi-02** container, navigate to the **customer-info** folder. Then right-click the **customerinfo.csv** file, select **New SQL script**, and then **Select TOP 100 rows**.

The screenshot shows the Azure Data Studio interface. On the left, the 'Data' pane shows a workspace with a 'Linked' connection. A red box labeled '1' highlights the 'Linked' tab. A red box labeled '2' highlights the 'asaworkspaceinaday84 (Primary ...)' connection. A red box labeled '3' highlights the 'wwi-02' connection. On the right, the 'wwi-02' connection is selected, and a red box labeled '4' highlights the 'customer-info' folder. A red box labeled '5' highlights the 'customerinfo.csv' file. A red box labeled '6' highlights the 'New SQL script' button. A red box labeled '7' highlights the 'Select TOP 100 rows' button.

2. Select **Run** to execute the script, and notice that the first row of the CSV file is the column header row. The columns in the resultset are named **C1**, **C2**, and so on.

The screenshot shows the Azure Data Studio interface with a SQL script being executed. A red box labeled '1' highlights the 'Run' button. The script is as follows:

```

1 This is auto-generated code
2 SELECT
3 TOP 100 *
4 FROM
5 OPENROWSET(
6 BULK 'https://asdatalakeinaday84.dfs.core.windows.net/wwi-02/customer-info/customerinfo.csv',
7 FORMAT = 'CSV',
8 PARSER_VERSION='2.0'
9 ) AS [result]
10

```

The 'Results' pane shows the execution results in a table view. A red box labeled '2' highlights the first row of the table, which contains the column headers: C1, C2, C3, C4, and C5. The table data is as follows:

C1	C2	C3	C4	C5
UserName	Gender	Phone	Email	CreditCard
Jason.Green	Male	1-364-410-6690 x08562	Jason.Green@adventureworks.co...	9777-4287-8862-7386
Ben_Dickens	Male	551-530-7354 x0506	Ben_Dickens@adventureworks.co...	2992-5468-4860-3593
Alton_Vandervort	Male	811-714-3148	Alton_Vandervort@adventurewo...	2235-1309-9262-3979
Theresa16	Female	826-960-1591 x58508	Theresa16@adventureworks.com	4128-2405-5087-4621
Robert_Jerde95	Male	(888) 258-7303 x20068	Robert_Jerde95@adventurework...	4347-5516-7396-9369
Jana_Schmeler	Female	386-275-8812 x4018	Jana_Schmeler@adventureworks	2146-3328-9686-9169

3. Update the script with the following code and **make sure you replace SUFFIX** in the OPENROWSET BULK path with your unique resource suffix.

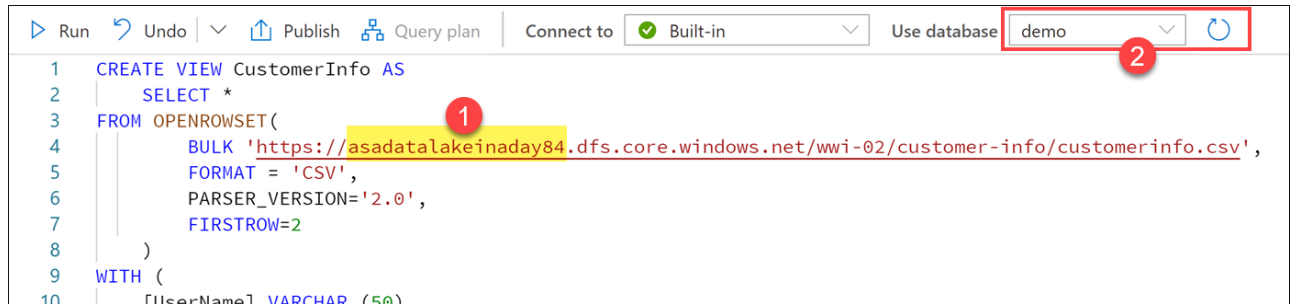
```

CREATE VIEW CustomerInfo AS
SELECT *
FROM OPENROWSET(
    BULK 'https://asdatalakeSUFFIX.dfs.core.windows.net/wwi-
02/customer-info/customerinfo.csv',
    FORMAT = 'CSV',
    PARSER_VERSION='2.0',
    FIRSTROW=2
)
WITH (
    [UserName] NVARCHAR (50),

```

```
[Gender] NVARCHAR (10),
[Phone] NVARCHAR (50),
[Email] NVARCHAR (100),
[CreditCard] NVARCHAR (50)
) AS [r];
GO
```

```
SELECT * FROM CustomerInfo;
GO
```



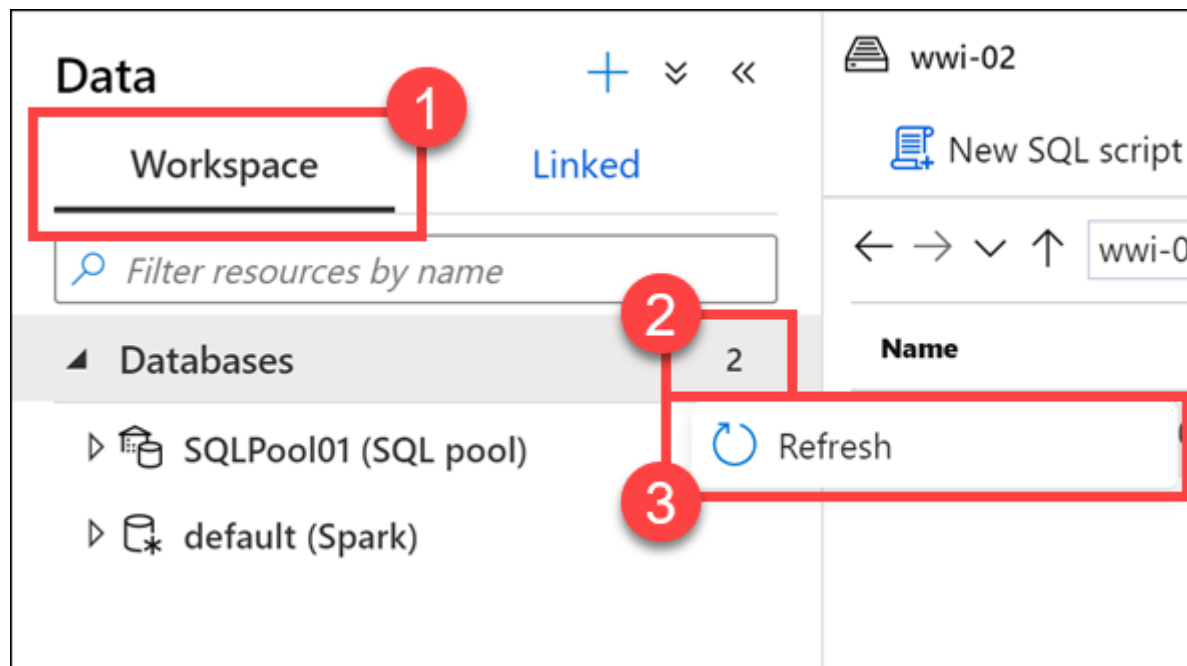
4. In the **Use database** list, ensure **demo** is still selected, and then run the script.

We just created the view to wrap the SQL query that selects data from the CSV file, then selected rows from the view:

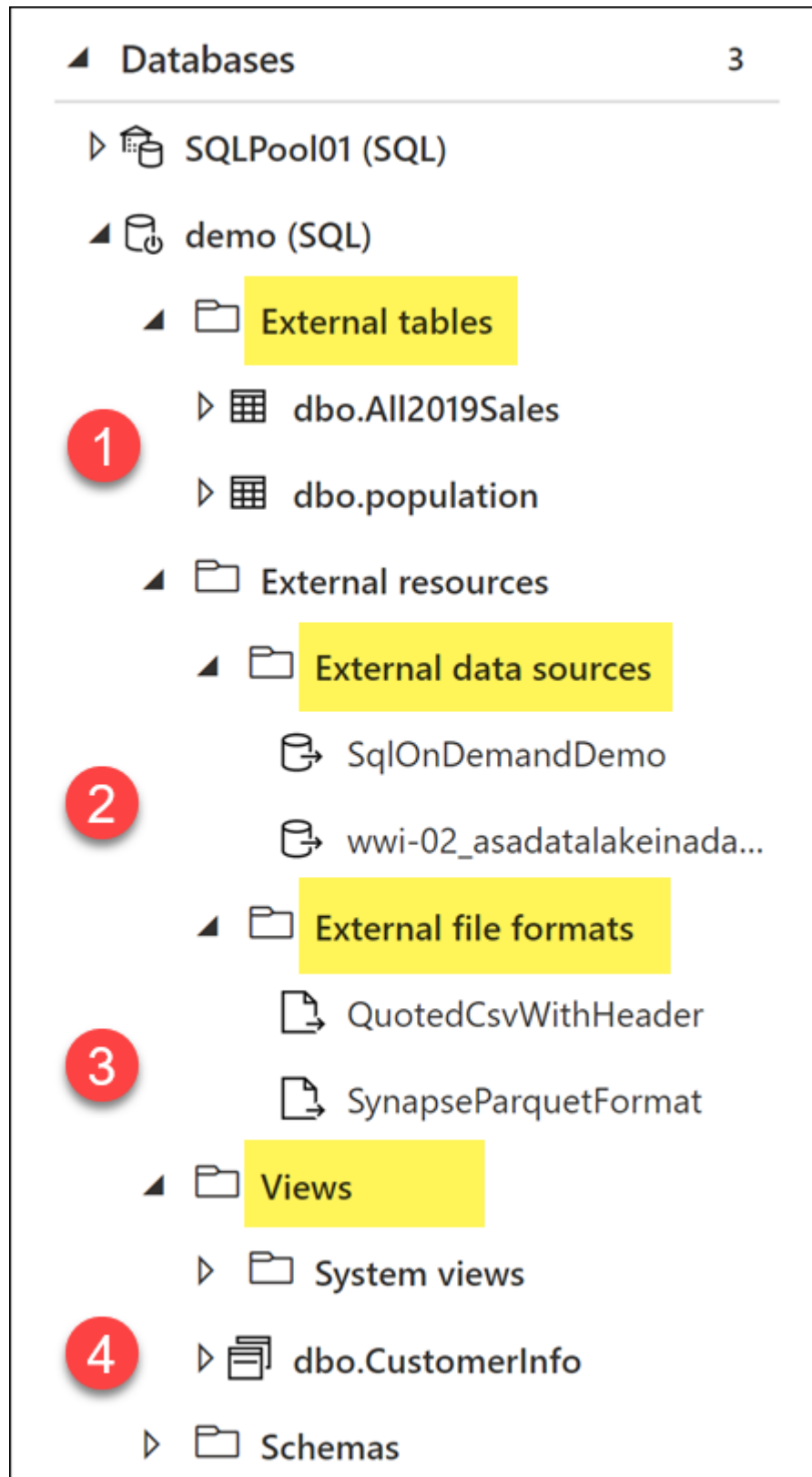
Results Messages				
Select Query 2 View Table Chart Export results				
Search				
UserName	Gender	Phone	Email	CreditCard
Jason.Green	Male	1-364-410-6690 x08562	Jason.Green@adventureworks.co...	9777-4287-8862-7386
Ben_Dickens	Male	551-530-7354 x0506	Ben_Dickens@adventureworks.c...	2992-5468-4860-3593
Alton_Vandervort	Male	811-714-3148	Alton_Vandervort@adventurewo...	2235-1309-9262-3979
Theresa16	Female	826-960-1591 x58508	Theresa16@adventureworks.com	4128-2405-5087-4621
Robert_Jerde95	Male	(888) 258-7303 x20068	Robert_Jerde95@adventurework...	4347-5516-7396-9369
Jana.Schmeler	Female	386-275-8812 x4018	Jana.Schmeler@adventureworks....	2146-3328-9686-9169
Nathaniel_Botsford51	Male	(502) 878-3300	Nathaniel_Botsford51@adventur...	3779-7612-8624-1689

Notice that the first row no longer contains the column headers. This is because we used the **FIRSTROW=2** setting in the **OPENROWSET** statement when we created the view.

5. In the **Data** hub, select the **Workspace** tab. Then select the actions ellipses (...) to the right of the Databases group and select **Refresh**. If the workspace is blank, then refresh the browser page.



6. Expand the **demo** SQL database.



The database contains the following objects that we created in our earlier steps:

- **1) External tables:** *All2019Sales* and *population*.
- **2) External data sources:** *SqlOnDemandDemo* and *wwi-02_asadatalakeinadayXXX_dfs_core_windows_net*.
- **3) External file formats:** *QuotedCsvWithHeader* and *SynapseParquetFormat*.
- **4) Views:** *CustomerInfo*

Exercise 2 - Securing access to data using a serverless SQL pool in Azure Synapse Analytics

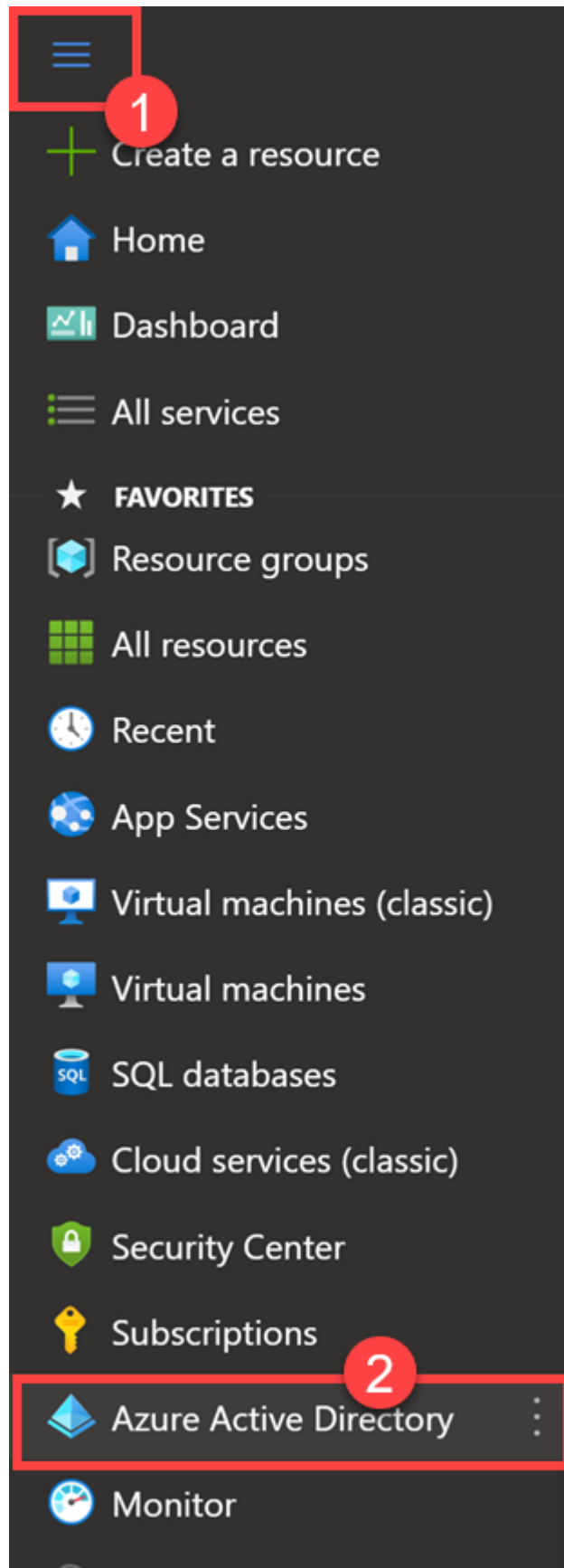
Tailwind Traders wants to enforce that any kind of modifications to sales data can happen in the current year only, while allowing all authorized users to query the entirety of data. They have a small group of admins who can modify historic data if needed.

- Tailwind Traders should create a security group in AAD, for example called **tailwind-history-owners**, with the intent that all users who belong to this group will have permissions to modify data from previous years.
- The **tailwind-history-owners** security group needs to be assigned to the Azure Storage built-in RBAC role **Storage Blob Data Owner** for the Azure Storage account containing the data lake. This allows AAD user and service principals that are added to this role to have the ability to modify all data from previous years.
- They need to add the user security principals who will have permissions to modify all historical data to the **tailwind-history-owners** security group.
- Tailwind Traders should create another security group in AAD, for example called **tailwind-readers**, with the intent that all users who belong to this group will have permissions to read all contents of the file system (**prod** in this case), including all historical data.
- The **tailwind-readers** security group needs to be assigned to the Azure Storage built-in RBAC role **Storage Blob Data Reader** for the Azure Storage account containing the data lake. This allows AAD user and service principals that are added to this security group to have the ability to read all data in the file system, but not to modify it.
- Tailwind Traders should create another security group in AAD, for example called **tailwind-2020-writers**, with the intent that all users who belong to this group will have permissions to modify data only from the year 2020.
- They would create a another security group, for example called **tailwind-current-writers**, with the intent that only security groups would be added to this group. This group will have permissions to modify data only from the current year, set using ACLs.
- They need to add the **tailwind-readers** security group to the **tailwind-current-writers** security group.
- At the start of the year 2020, Tailwind Traders would add **tailwind-current-writers** to the **tailwind-2020-writers** security group.
- At the start of the year 2020, on the **2020** folder, Tailwind Traders would set the read, write and execute ACL permissions for the **tailwind-2020-writers** security group.
- At the start of the year 2021, to revoke write access to the 2020 data they would remove the **tailwind-current-writers** security group from the **tailwind-2020-writers** group. Members of **tailwind-readers** would continue to be able to read the contents of the file system because they have been granted read and execute (list) permissions not by the ACLs but by the RBAC built in role at the level of the file system.
- This approach takes into account that current changes to ACLs do not inherit permissions, so removing the write permission would require writing code that traverses all of its content removing the permission at each folder and file object.
- This approach is relatively fast. RBAC role assignments may take up to five minutes to propagate, regardless of the volume of data being secured.

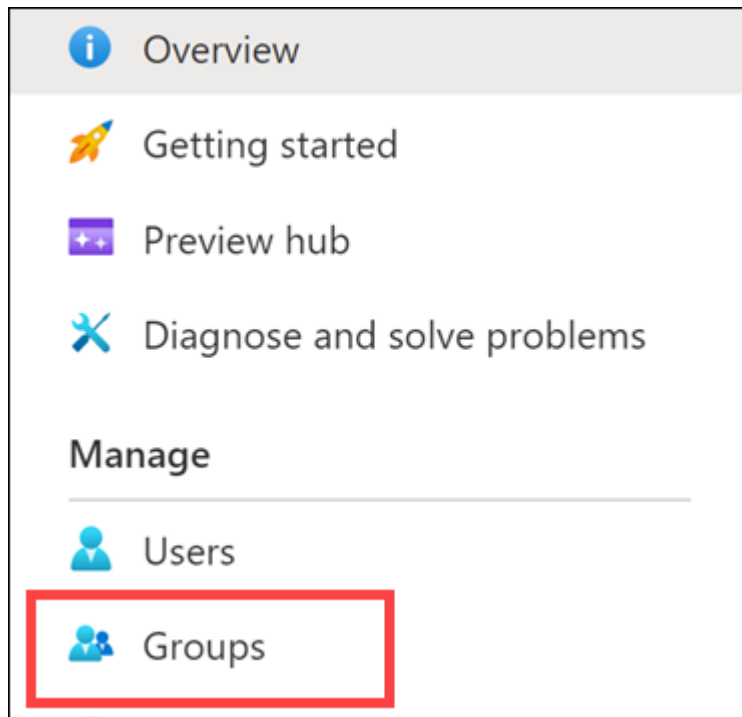
Task 1: Create Azure Active Directory security groups

In this segment, we will create security groups as described above. However, our data set ends in 2019, so we will create a **tailwind-2019-writers** group instead of 2021.

1. Switch back to the Azure portal (<https://portal.azure.com>) in a different browser tab, leaving Synapse Studio open.
2. On the **Home** page, expand the portal menu if it is not already expanded, then select **Azure Active Directory**.



3. Select **Groups** in the left-hand menu.



4. Select **+ New group**.
5. Ensure that the **Security** group type is selected, and enter **tailwind-history-owners-SUFFIX** (where *suffix* is your unique resource suffix) for the **Group name**, and then select **Create**.

The screenshot shows the 'New Group' form in the Azure portal. The form has the following fields and values:

- Group type ***: Security (selected from a dropdown menu)
- Group name ***: tailwind-history-owners-jdh42 (with a green checkmark indicating it is valid)
- Group description**: Enter a description for the group (text input field)
- Membership type ***: Assigned (selected from a dropdown menu)
- Owners**: No owners selected
- Members**: No members selected

A blue 'Create' button is located at the bottom of the form.

6. Add a second new security group named **tailwind-readers-SUFFIX** (where *SUFFIX* is your unique resource suffix).

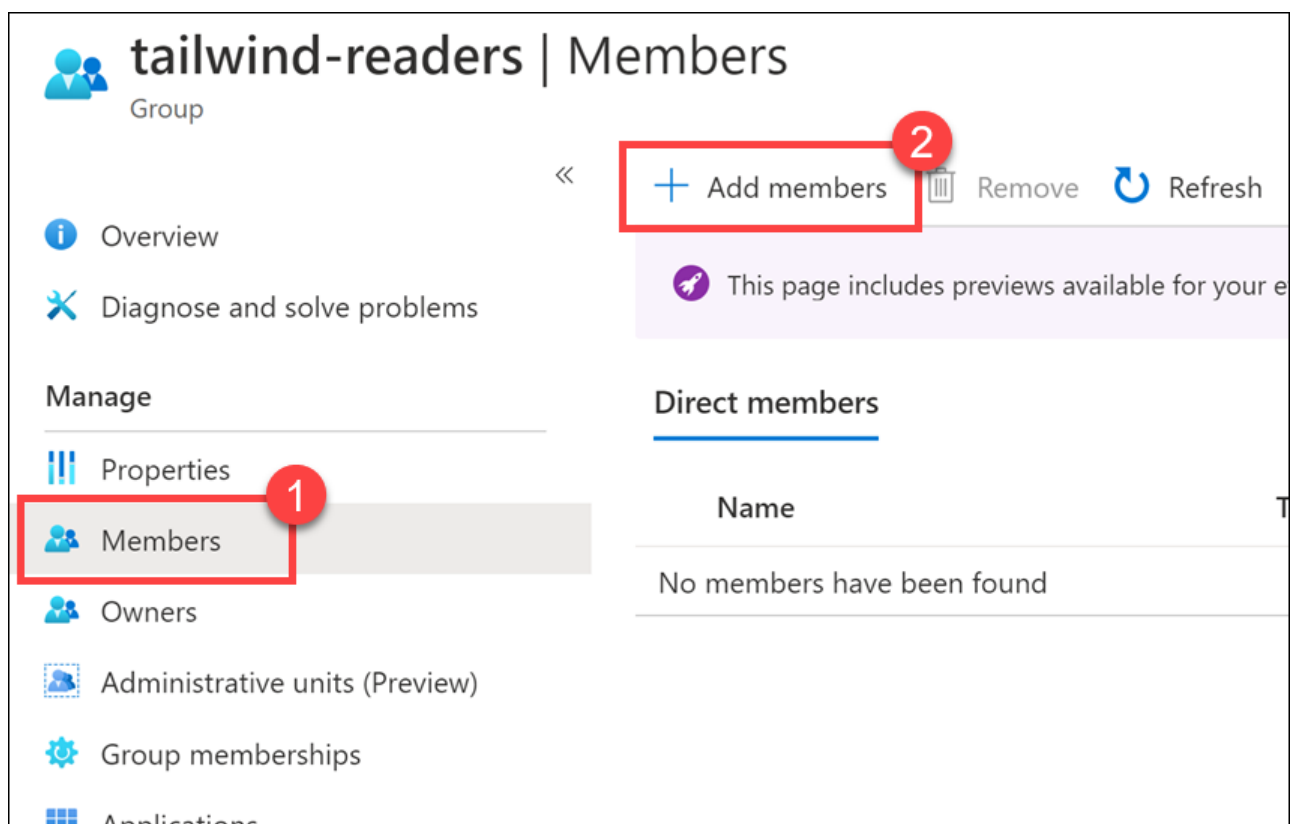
7. Add a third security group named **tailwind-current-writers-SUFFIX** (where *SUFFIX* is your unique resource suffix).
8. Add a fourth security group named **tailwind-2019-writers-SUFFIX** (where *SUFFIX* is your unique resource suffix).

Note: In the remaining instructions in this exercise, we'll omit the *-SUFFIX* part of the role names for clarity. You should work with your uniquely identified role names based on your specific resource suffix.

Task 2: Add group members

To test out the permissions, we will add your own account to the groups.

1. Open your newly created **tailwind-readers** group.
2. Select **Members** on the left, then select **+ Add members**.



3. Search for your user account that you are signed into for the lab, then select **Select**.
4. Open your **tailwind-2019-writers** group.
5. Select **Members** on the left, then select **+ Add members**.
6. Search for **tailwind**, select your **tailwind-current-writers** group, then select **Select**.

Add members ✕

Search ⓘ

tailwind ✕

TA

tailwind-2019-writers

TA

tailwind-current-writers
Selected

TA

tailwind-history-owners

TA

tailwind-readers

Selected items

TA

tailwind-current-writers

Remove

Select

7. Select **Overview** in the left-hand menu, then **copy** the **Object Id**.

tailwind-2019-writers ⓘ

Group

Overview ⓘ

Diagnose and solve problems

Manage

Properties

Members

Owners

Administrative units (Preview)

Group memberships

Applications

Licenses

Azure role assignments

Activity

Access reviews

TA

tailwind-2019-writers

Membership type

Assigned

Source

Cloud

Type

Security

Object Id

6d1d9de1-adea-4a4c-8307-41b6ff685724

Creation date

9/13/2020, 6:52:13 PM





Note: Save the **Object Id** value to Notepad or similar text editor. This will be used in a later step when you assign access control in the storage account.

20 / 29

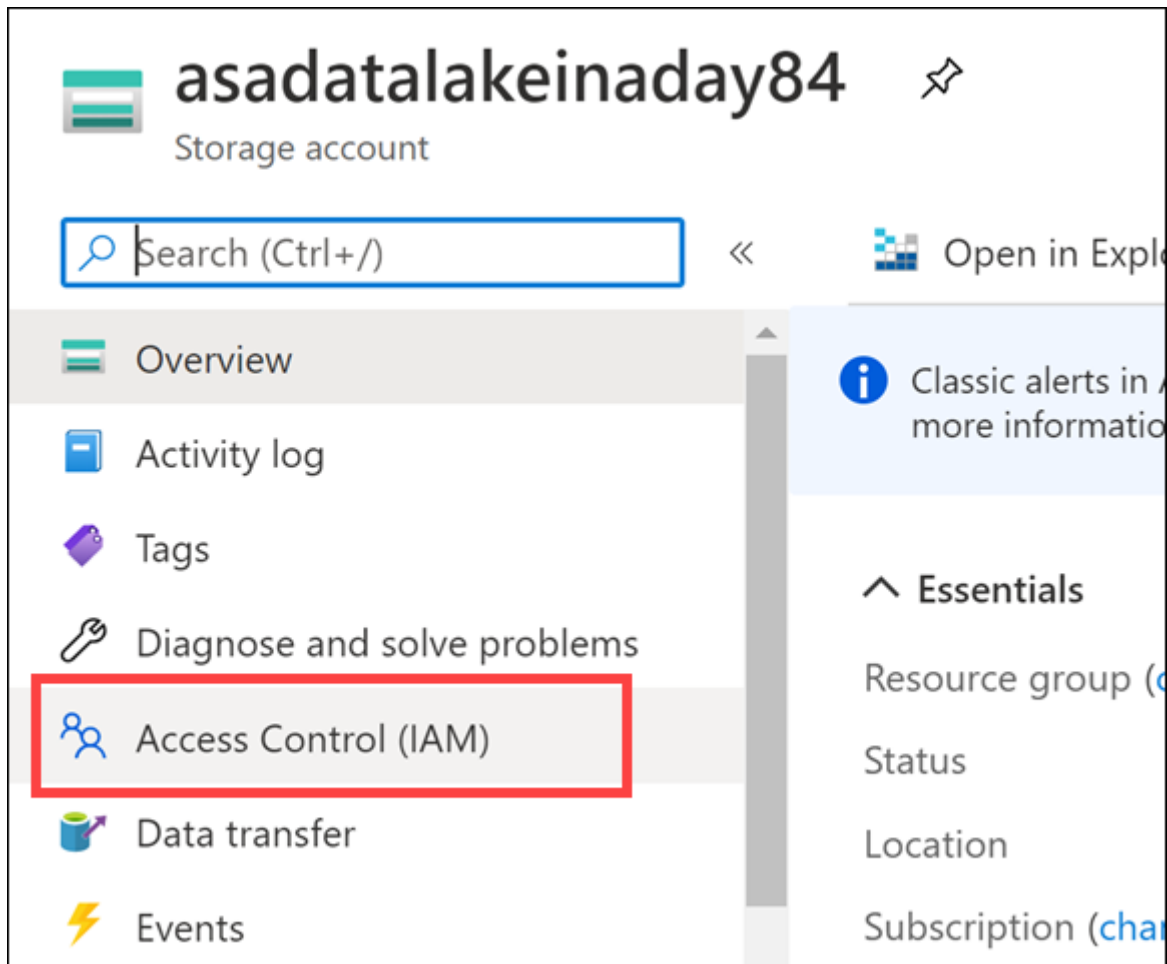
Task 3: Configure data lake security - Role-Based Access Control (RBAC)

1. In the Azure portal, open the **data-engineering-synapse-xxxxxxx** resource group.
2. Open the **asadatalakexxxxxxx** storage account.

Showing 1 to 25 of 25 records. ☐ Show hidden types ⓘ No g

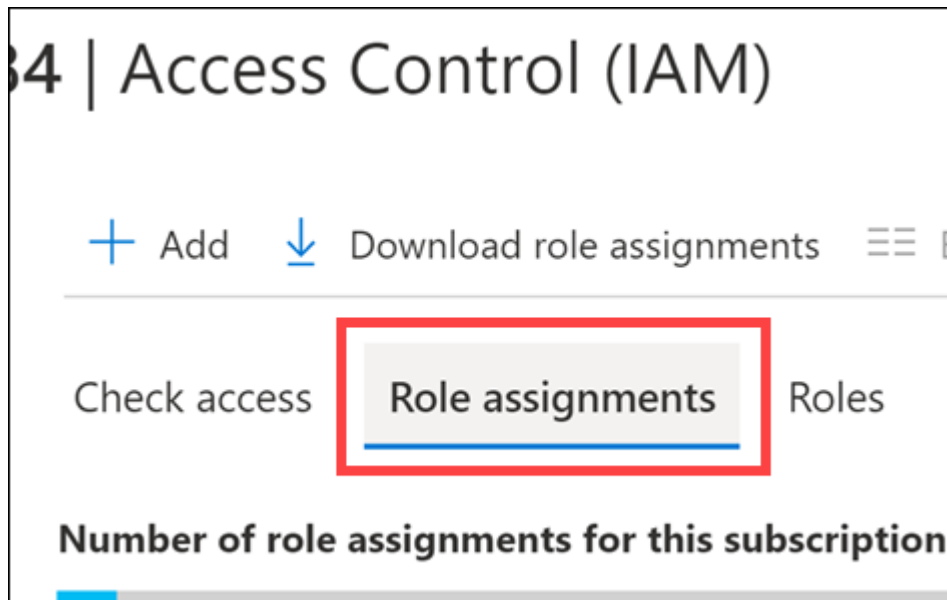
<input type="checkbox"/> Name ↑↓	Type ↑↓
<input type="checkbox"/>  amlworkspaceinaday84	Machine Learning
<input type="checkbox"/>  asaappinsightsinaday84	Application Insights
<input type="checkbox"/>  asacosmosdbinaday84	Azure Cosmos DB account
<input type="checkbox"/>  asadatalakeinaday84	Storage account

3. Select **Access Control (IAM)** in the left-hand menu.

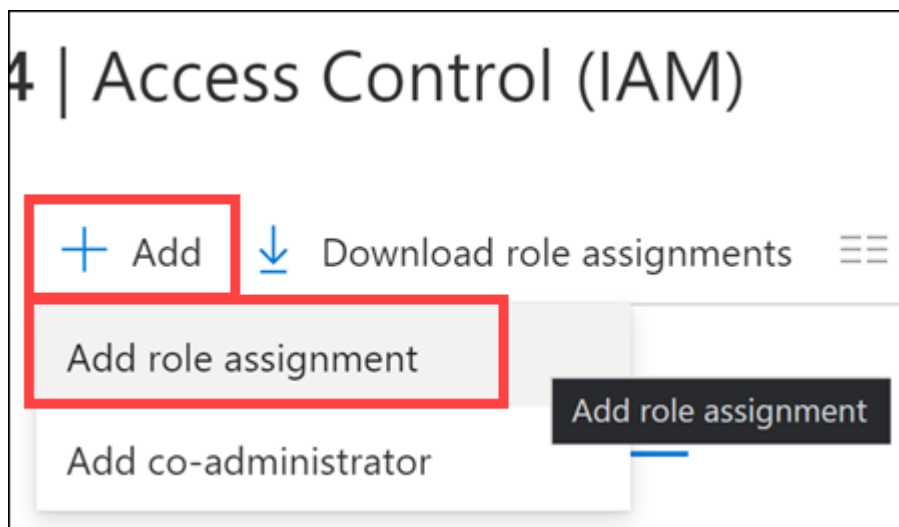


The screenshot displays the Azure portal interface for the storage account 'asadatalakeinaday84'. The left-hand navigation menu is expanded, and the 'Access Control (IAM)' option is highlighted with a red rectangular box. The right-hand pane shows the 'Essentials' section, which includes a search bar, a 'Search (Ctrl+ /)' button, and a list of links: 'Classic alerts in a', 'more information', 'Essentials', 'Resource group (c)', 'Status', 'Location', and 'Subscription (cha)'.

4. Select the **Role assignments** tab.



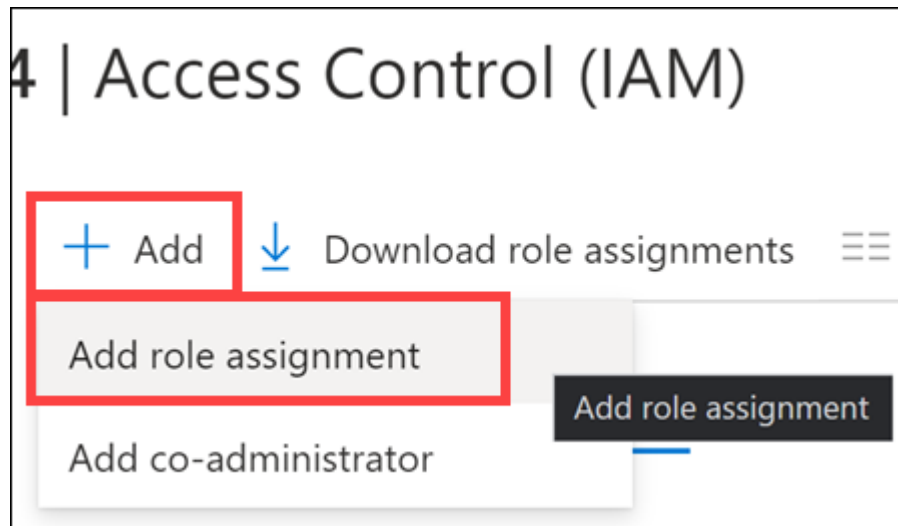
5. Select + **Add**, then **Add role assignment**.



6. In the **Role** screen, search and select **Storage Blob Data Reader** and then click on **Next**. In the **Members** screen, click on + **Select members** then search for **tailwind-readers** and select your **tailwind-readers** group in the results. Then click **Select**. Then click on **Review + assign**, and click on **Review + assign** a second time.

Because your user account is added to this group, you will have read access to all files in the blob containers of this account. Tailwind Traders would need to add all users to the **tailwind-readers** security group.

7. Select + **Add**, then **Add role assignment**.



8. For **Role**, search **Storage Blob Data Owner**, then select **Next**.

9. In the **Members** screen, click on **+ Select Members** and search for **tailwind** and select your **tailwind-history-owners** group in the results. Then click on **Review + Assign**, and click on **Review + Assign** again.

The **tailwind-history-owners** security group is now assigned to the Azure Storage built-in RBAC role **Storage Blob Data Owner** for the Azure Storage account containing the data lake. This allows Azure AD user and service principals that are added to this role to have the ability to modify all data.

Tailwind Traders needs to add the user security principals who will have have permissions to modify all historical data to the **tailwind-history-owners** security group.

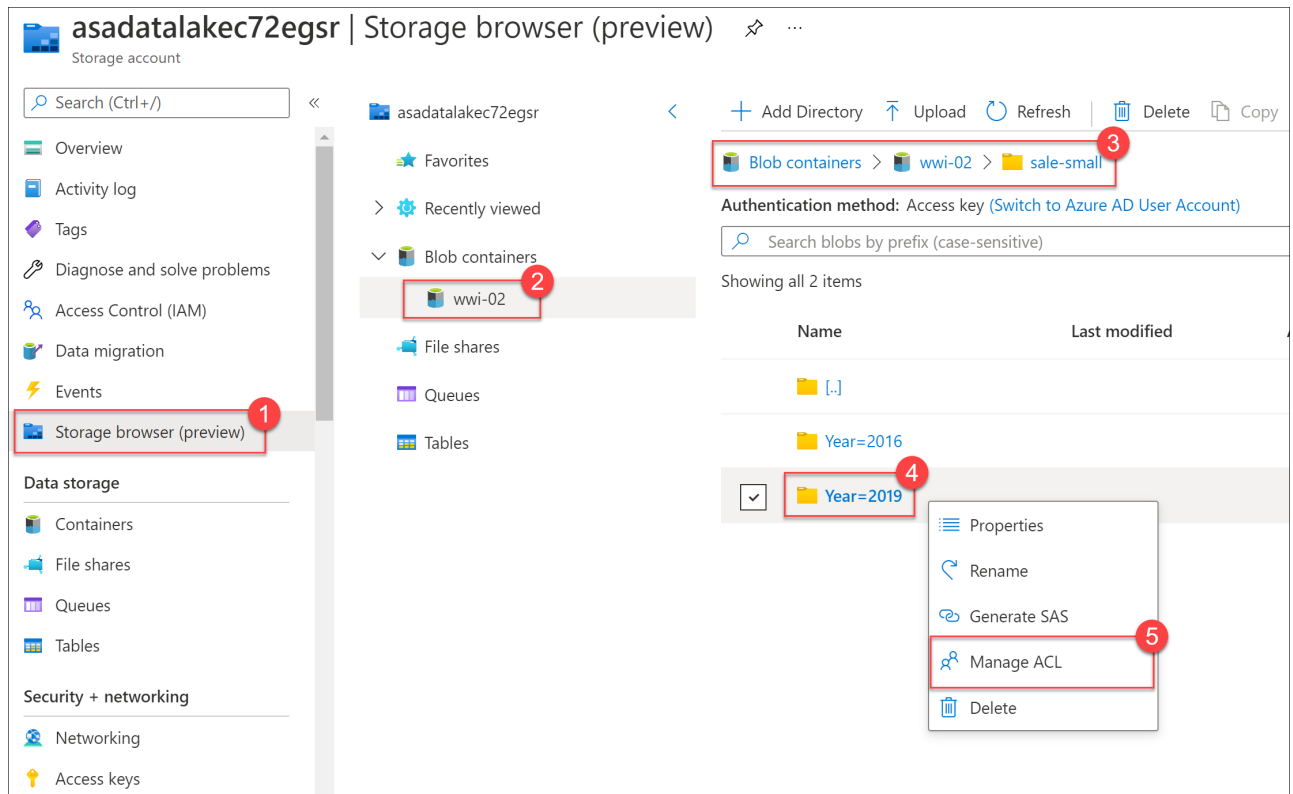
10. In the **Access Control (IAM)** list for the storage account, select your Azure user account under the **Storage Blob Data Owner** role, then select **Remove**.

Notice that the **tailwind-history-owners** group is assigned to the **Storage Blob Data Owner** group, and **tailwind-readers** is assigned to the **Storage Blob Data Reader** group.

Note: You may need to navigate back to the resource group, then come back to this screen to see all of the new role assignments.

Task 4: Configure data lake security - Access Control Lists (ACLs)

1. Select **Storage browser (preview)** on the left-hand menu. Expand **Blob containers** and select the **wwi-02** container. Open the **sale-small** folder, right-click the **Year=2019** folder, then select **Manage ACL**.



- In the Manage ACL screen, in the **Access permissions** screen, click on **+ Add principal**, paste the **object Id** value you copied from the **tailwind-2019-writers** security group into the **Add principal** search box, click on **tailwind-2019-writers-suffix**, then select **Select**.
- Now you should see that the **tailwind-2019-writers** group is selected in the Manage ACL dialog. Check **Read**, **Write**, and **Execute** checkboxes, then select **Save**.
- In the Manage ACL screen, select the **Default permissions** tab. Enable the checkbox for **Configure default permissions**, then click on **+ Add principal**, paste the **object Id** value you copied from the **tailwind-2019-writers** security group into the **Add principal** search box, click on **tailwind-2019-writers-suffix**, then select **Select**. Check **Read**, **Write**, and **Execute** checkboxes for the new object, then select **Save**.

Now the security ACLs have been set to allow any users added to the **tailwind-current** security group to write to the **Year=2019** folder, by way of the **tailwind-2019-writers** group. These users can only manage current (2019 in this case) sales files.

At the start of the following year, to revoke write access to the 2019 data they would remove the **tailwind-current-writers** security group from the **tailwind-2019-writers** group. Members of **tailwind-readers** would continue to be able to read the contents of the file system because they have been granted read and execute (list) permissions not by the ACLs but by the RBAC built in role at the level of the file system.

Notice that we configured both the *access* ACLs and *default* ACLs in this configuration.

Access ACLs control access to an object. Files and directories both have access ACLs.

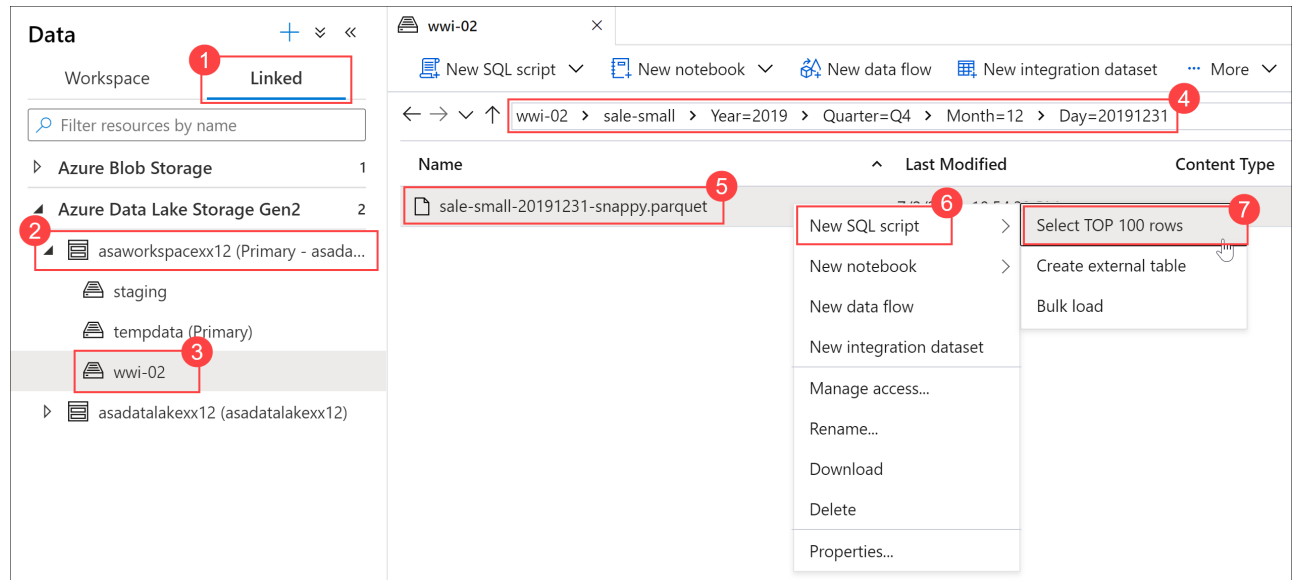
Default ACLs are templates of ACLs associated with a directory that determine the access ACLs for any child items that are created under that directory. Files do not have default ACLs.

Both access ACLs and default ACLs have the same structure.

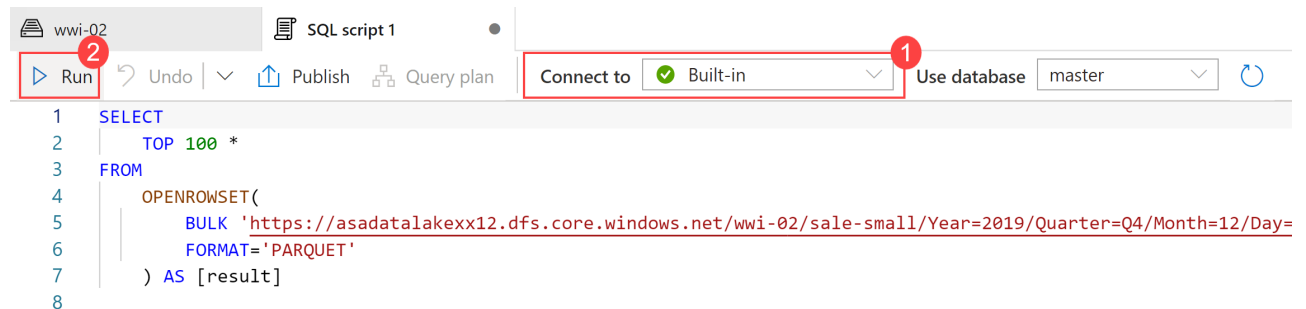
Note be sure to save after configuring permissions.

Task 5: Test permissions

1. In Synapse Studio, in the **Data** hub, on the **Linked** tab select the **Azure Data Lake Storage Gen2/asaworkspacexxxxxx/wwi02** container; and in the *sale-small/Year=2019/Quarter=Q4/Month=12/Day=20191231* folder, right-click the **sale-small-20191231-snappy.parquet** file, select **New SQL script**, and select **Select TOP 100 rows**.



2. Ensure **Built-in** is selected in the **Connect to** dropdown list above the query window, then run the query. Data is loaded by the serverless SQL pool endpoint and processed as if it was coming from any regular relational database.



The cell output shows the query results from the Parquet file.

```

1  SELECT
2      TOP 100 *
3  FROM
4      OPENROWSET(
5          BULK 'https://asadatalakex1.dfs.core.windows.net/wwi-02/sale-small-20191231-snappy.parquet',
6          FORMAT='PARQUET'
7      ) AS [result]
8

```

Results Messages

View **Table** Chart [Export results](#)

Search

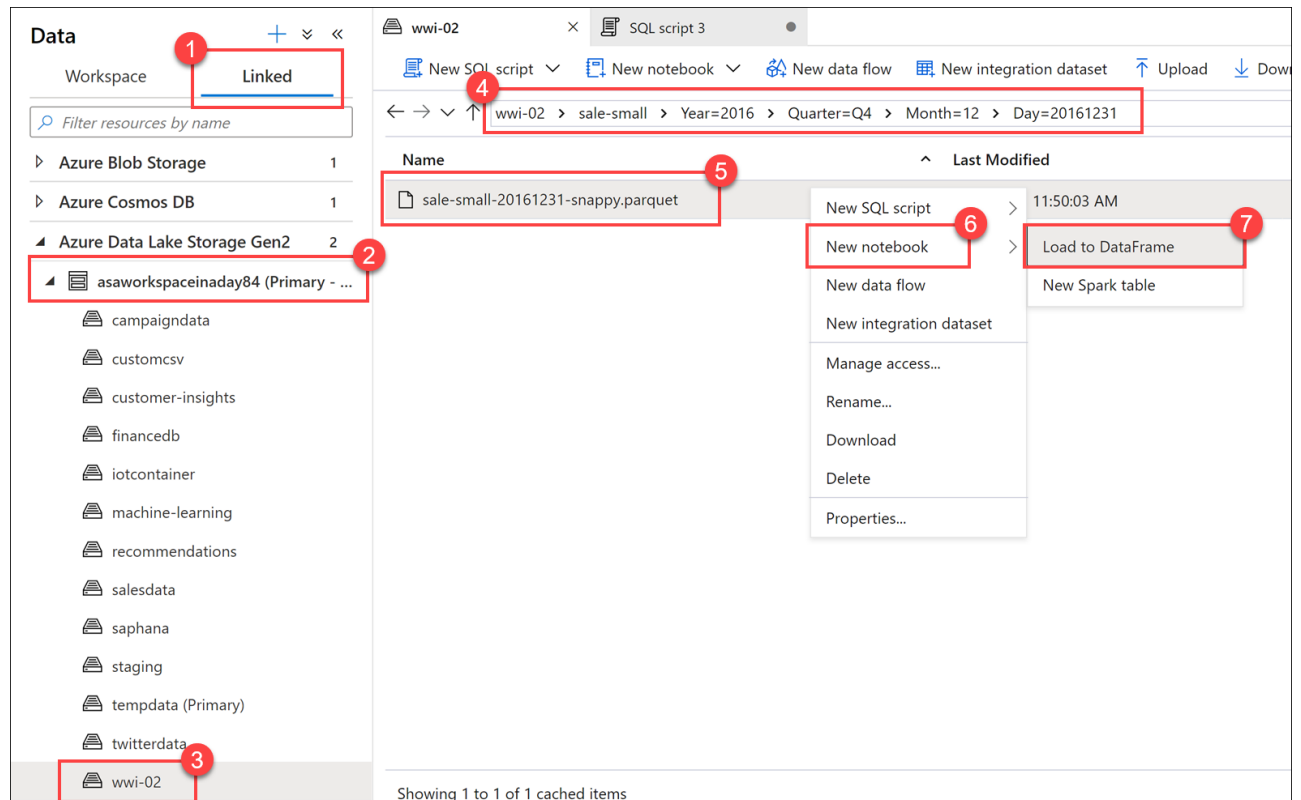
TransactionId	CustomerId	ProductId	Quantity	Price
3e280434-9d5f...	102449	3173	2	26.470000000000..
3e280434-9d5f...	102449	2686	1	21.460000000000..
3e280434-9d5f...	102449	2335	3	16.650000000000..
3e280434-9d5f...	102449	337	3	24.180000000000..

The read permissions to the Parquet file assigned to us through the **tailwind-readers** security group, which then is granted RBAC permissions on the storage account through the **Storage Blob Data Reader** role assignment, is what enabled us to view the file contents.

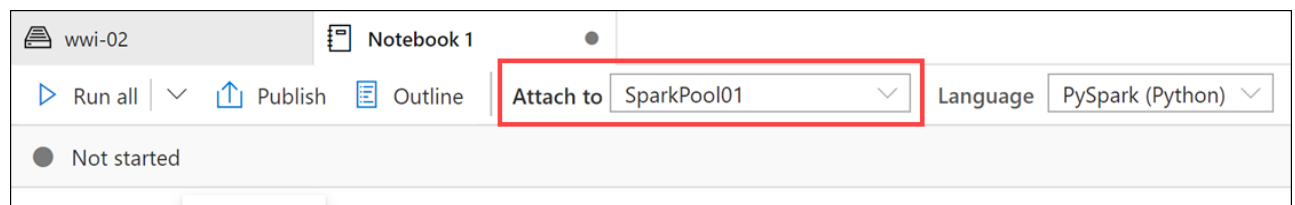
However, since we removed our account from the **Storage Blob Data Owner** role, and we did not add our account to the **tailwind-history-owners** security group, what if we try to write to this directory?

Let's give it a try.

3. In the **wwi-02** pane, right-click the **sale-small-20191231-snappy.parquet** file, select **New Notebook**, then select **Load to DataFrame**.



4. Attach your **SparkPool01** Spark pool to the notebook.



5. Run the cell in the notebook to load the data into a dataframe. This may take a while as the spark pool is started, but eventually it should display the first ten rows of the data - confirming once again that you have permission to read data in this location.

6. Under the results, select **+ Code** to add a code cell beneath the existing cell.

7. Enter the following code, replacing *SUFFIX* with the unique suffix for your data lake resource (you can copy this from cell 1 above):

```
df.write.parquet('abfss://wwi-02@asdatalakeSUFFIX.dfs.core.windows.net/sale-small/Year=2016/Quarter=Q4/Month=12/Day=20161231/sale-small-20161231-snappy-test.parquet')
```

8. Run the new cell you just added. You should see a **403 error** in the output.

bebd0650-b54f-...	3	2253	4	26.8	107.2	20161231
bebd0650-b54f-...	3	3498	4	25.08	100.32	20161231
bebd0650-b54f-...	3	2358	3	23.29	69.87	20161231
bebd0650-b54f-...	3	198	4	30.82	123.28	20161231
bebd0650-b54f-...	3	190	1	24.65	24.65	20161231

Cell 2

```
1 df.write.parquet('abfss://wwi-02@asadalakeinaday84.dfs.core.windows.net/sale-small/Year=2016, ...
```

Command executed in 2s 583ms by joel on 11-27-2020 10:43:03.494 -05:00

Py4JJavaError : An error occurred while calling o155.parquet.
 : Operation failed: "This request is not authorized to perform this operation using this permission.", 403, PUT, https://asadalakeinaday84.dfs.core.windows.net/wwi-02/sale-small/Year%3D2016/Quarter%3DQ4/Month%3D12/Day%3D20161231/sale-small-20161231-snappy-test.parquet/_temporary/0?resource=directory&timeout=90, AuthorizationPermissionMismatch, "This request is not authorized to perform this operation using this permission." RequestId:1ea8d457-401f-0078-27d3-c4ff84000000 Time:2020-11-27T15:43:02.4187420Z"
 at org.apache.hadoop.fs.azurebfs.services.AbfsRestOperation.execute(AbfsRestOperation.java:166)
 at org.apache.hadoop.fs.azurebfs.services.AbfsClient.createPath(AbfsClient.java:278)
 at org.apache.hadoop.fs.azurebfs.AzureBlobFileSystemStore.createDirectory(AzureBlobFileSystemStore.java:419)
 at org.apache.hadoop.fs.azurebfs.AzureBlobFileSystemStore.mkdirs(AzureBlobFileSystemStore.java:402)

As expected, you do not have write permissions. The error returned by cell 2 is, *This request is not authorized to perform this operation using this permission.*, with a status code of 403.

9. Publish the notebook and end the session. Then sign out of Azure Synapse Studio and close the browser tab, returning to the Azure portal tab (<https://portal.azure.com>).
10. On the **Home** page, in the portal menu, select **Azure Active Directory**.
11. Select **Groups** in the left-hand menu.
12. Type **tailwind** in the search box, then select your **tailwind-history-owners** group in the results.
13. Select **Members** on the left, then select **+ Add members**.
14. Add your user account that you are signed into for the lab, then select **Select**.
15. In a new tab, browse to Azure Synapse Studio (<https://web.azuresynapse.net/>). Then on the **Develop** tab, expand **Notebooks** and re-open the notebook you published previously.
16. Click **Run All** to re-run all of the cells in the notebook. After a while, the Spark session will start and the code will run. Cell 2 should return a status of **Succeeded**, indicating that a new file was written to the data lake store.

bebd0650-b54f-...	3	3498	4	25.08	100.32	20161231
bebd0650-b54f-...	3	2358	3	23.29	69.87	20161231
bebd0650-b54f-...	3	198	4	30.82	123.28	20161231
bebd0650-b54f-...	3	190	1	24.65	24.65	20161231

Cell 2

```
1 df.write.parquet('abfss://wwi-02@asadalakeinaday84.dfs.core.windows.net/sale-small/Year=2016, ...
```

Command executed in 4s 413ms by joel on 11-27-2020 10:47:31.987 -05:00

Job execution Succeeded Spark 2 executors 8 cores [View in monitoring](#) [Open Spark UI](#)

The cell succeeded this time because we added our account to the **tailwind-history-owners** group, which is assigned the **Storage Blob Data Owner** role.

Note: If you encounter the same error this time, **stop the Spark session** on the notebook, then select **Publish all**, then Publish. After publishing your changes, select your user profile on the top-right corner of the page and **log out**. **Close the browser tab** after logout is successful, then re-launch Synapse Studio (<https://web.azuresynapse.net/>), re-open the notebook, then re-run the cell. This may be needed because you must refresh the security token for the auth changes to take place.

17. At the top right of the notebook, use the **Stop Session** button to stop the notebook session.

18. Publish the notebook if you want to save the changes. Then close it.

Now let's verify that the file was written to the data lake.

19. In Synapse Studio, in the **Data** hub, on the **Linked** tab select the **Azure Data Lake Storage Gen2/asaworkspacexxxxxxx/wwi02** container; and browse to the *sale-small/Year=2019/Quarter=Q4/Month=12/Day=20191231* folder to verify that a new file has been added to this folder.

