Use Delta Tables in Apache Spark

Tables in a Microsoft Fabric Lakehouse are based on the open-source Delta Lake format. Delta Lake adds support for relational semantics for both batch and streaming data. In this exercise you will create Delta tables and explore the data using SQL queries.

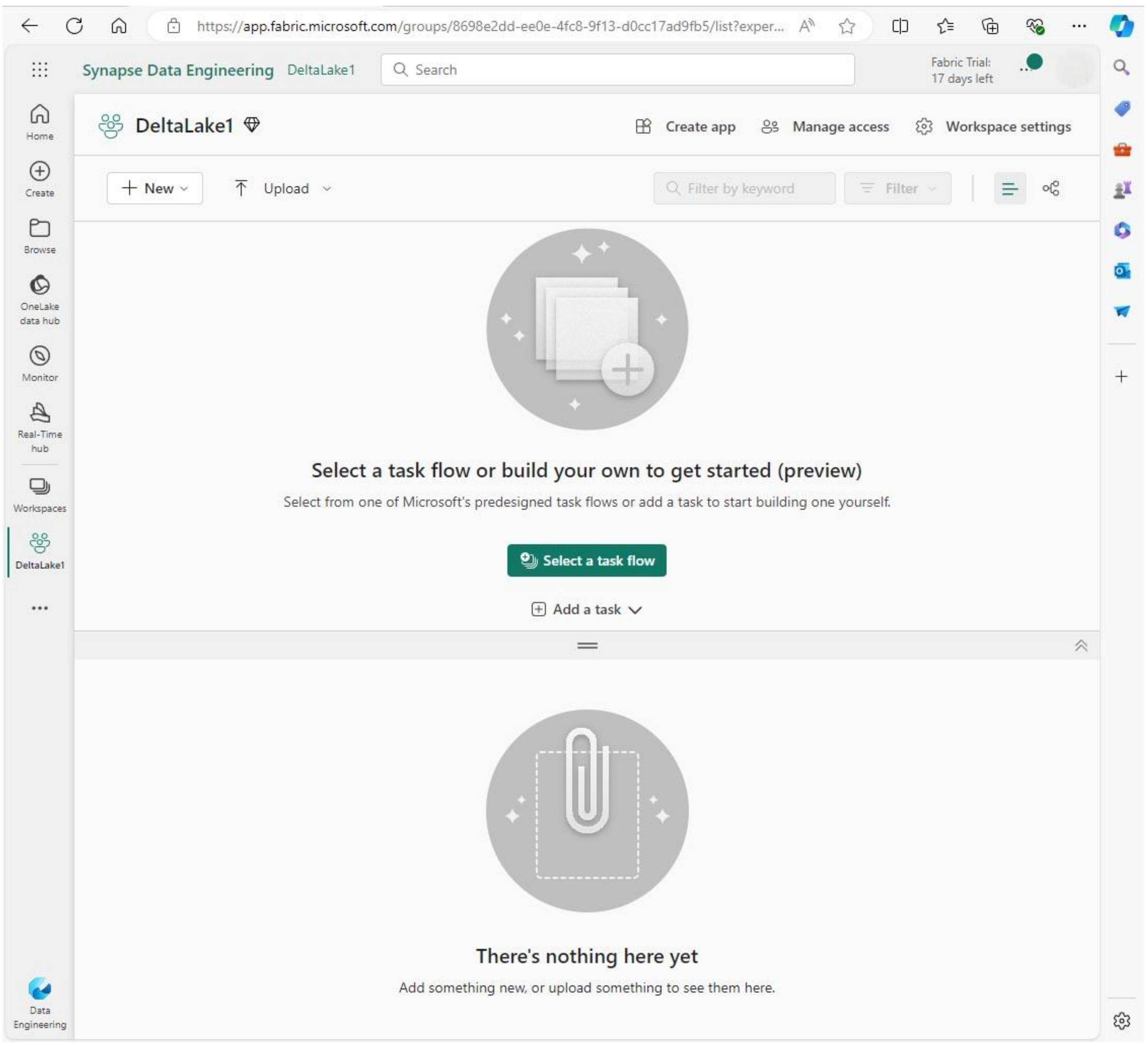
This exercise should take approximately **45** minutes to complete

[!NOTE] You need a Microsoft Fabric trial to complete this exercise.

Create a workspace

First, create a workspace with the *Fabric trial* enabled.

1. Navigate to the Microsoft Fabric home page at `https://app.fabric.microsoft.com/home?experience=fabric` in a browser, and sign in with your Fabric credentials.
2. In the menu bar on the left, select **Workspaces** (🗇).
3. Create a **new workspace** with a name of your choice, selecting a licensing mode that includes Fabric capacity (Trial, Premium, or Fabric).
4. When your new workspace opens, it should be empty.



Select a task flow or build your own to get started (preview)

Select from one of Microsoft's predesigned task flows or add a task to start building one yourself.

There's nothing here yet

Add something new, or upload something to see them here.

Create a lakehouse and upload data

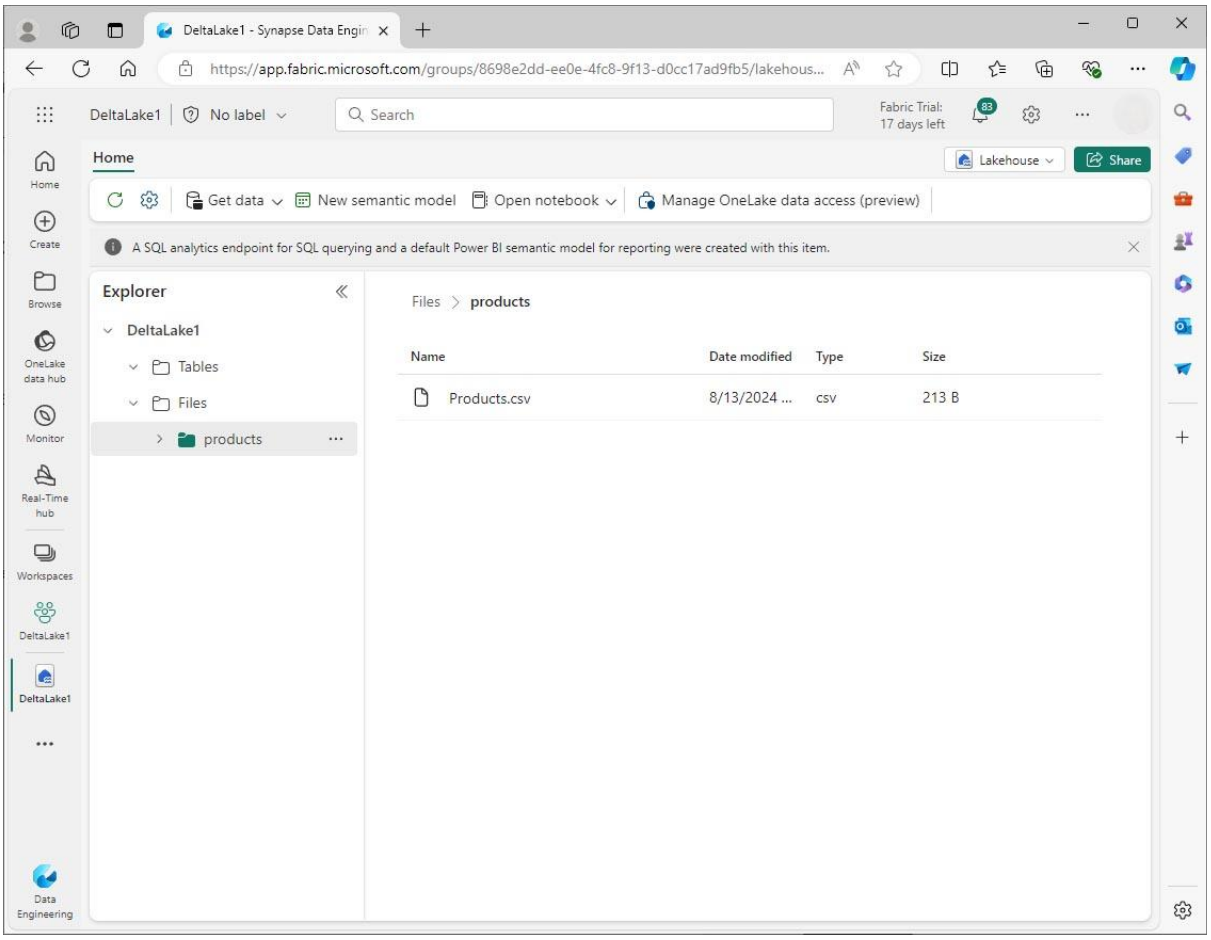
Now that you have a workspace, it's time to create a lakehouse and upload some data.

1. On the menu bar on the left, select **Create**. In the *New* page, under the *Data Engineering* section, select **Lakehouse**. Give it a unique name of your choice.

   Note: If the **Create** option is not pinned to the sidebar, you need to select the ellipsis (**...**) option first.

2. There are various ways to ingest data, but in this exercise you'll download a text file to your local computer (or lab VM if applicable) and then upload it to your lakehouse. Download the data file from `https://github.com/MicrosoftLearning/dp-data/raw/main/products.csv`, saving it as *products.csv*.

3. Return to the web browser tab containing your lakehouse, and in the Explorer pane, next to the **Files** folder, select the ... menu. Create a **New subfolder** called *products*.
4. In the ... menu for the products folder, **upload** the *products.csv* file from your local computer (or lab VM if applicable).
5. After the file has been uploaded, select the **products** folder to verify that the file has been uploaded, as shown here:



A SQL analytics endpoint for SQL querying and a default Power BI semantic model for reporting were created with this item.

Name	Date modified	Type	Size
Products.csv	8/13/2024 ...	csv	213 B

Explore data in a DataFrame

1. Create a **New notebook**. After a few seconds, a new notebook containing a single cell will open. Notebooks are made up of one or more cells that can contain code or markdown (formatted text).
2. Select the first cell (which is currently a code cell), and then in the top-right tool bar, use the **M↓** button to convert it to a markdown cell. The text contained in the cell will then be displayed as formatted text. Use markdown cells to provide explanatory information about your code.
3. Use the 🖉 (Edit) button to switch the cell to editing mode, then modify the markdown as follows:

codeCopy

```
# Delta Lake tables
Use this notebook to explore Delta Lake functionality
```

4. Click anywhere in the notebook outside of the cell to stop editing it and see the rendered markdown.
5. Add a new code cell, and add the following code to read the products data into a DataFrame using a defined schema:

codeCopy

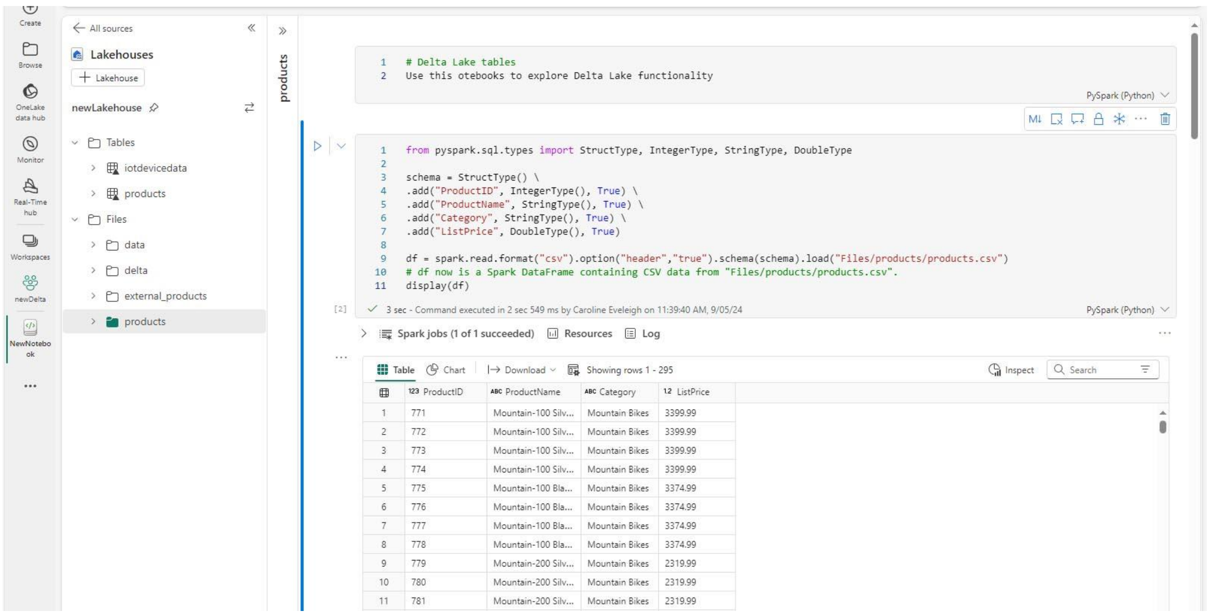
```
 from pyspark.sql.types import StructType, IntegerType, StringType,
DoubleType

 # define the schema
 schema = StructType() \
 .add("ProductID", IntegerType(), True) \
 .add("ProductName", StringType(), True) \
 .add("Category", StringType(), True) \
 .add("ListPrice", DoubleType(), True)

 df =
spark.read.format("csv").option("header","true").schema(schema).load("Files
/products/products.csv")
 # df now is a Spark DataFrame containing CSV data from
"Files/products/products.csv".
 display(df)
```

[!TIP] Hide or display the explorer panes by using the chevron « icon. This enables you to either focus on the notebook, or your files.

1. Use the **Run cell** (▷) button on the left of the cell to run it.

[!NOTE] Since this is the first time you've run any code in this notebook, a Spark session must be started. This means that the first run can take a minute or so to complete. Subsequent runs will be quicker.

1. When the cell code has completed, review the output below the cell, which should look similar to this:



```
# Delta Lake tables
Use this otebooks to explore Delta Lake functionality
```

PySpark (Python)

```
from pyspark.sql.types import StructType, IntegerType, StringType, DoubleType

schema = StructType() \
.add("ProductID", IntegerType(), True) \
.add("ProductName", StringType(), True) \
.add("Category", StringType(), True) \
.add("ListPrice", DoubleType(), True)

df = spark.read.format("csv").option("header","true").schema(schema).load("Files/products/products.csv")
# df now is a Spark DataFrame containing CSV data from "Files/products/products.csv".
display(df)
```

[2] 3 sec - Command executed in 2 sec 549 ms by Caroline Eveleigh on 11:39:40 AM, 9/05/24 PySpark (Python)

Spark jobs (1 of 1 succeeded) Resources Log

Table Chart Download Showing rows 1 - 295 Inspect Search

	ProductID	ProductName	Category	ListPrice
1	771	Mountain-100 Silv...	Mountain Bikes	3399.99
2	772	Mountain-100 Silv...	Mountain Bikes	3399.99
3	773	Mountain-100 Silv...	Mountain Bikes	3399.99
4	774	Mountain-100 Silv...	Mountain Bikes	3399.99
5	775	Mountain-100 Bla...	Mountain Bikes	3374.99
6	776	Mountain-100 Bla...	Mountain Bikes	3374.99
7	777	Mountain-100 Bla...	Mountain Bikes	3374.99
8	778	Mountain-100 Bla...	Mountain Bikes	3374.99
9	779	Mountain-200 Silv...	Mountain Bikes	2319.99
10	780	Mountain-200 Silv...	Mountain Bikes	2319.99
11	781	Mountain-200 Silv...	Mountain Bikes	2319.99

Create Delta tables

You can save the DataFrame as a Delta table by using the *saveAsTable* method. Delta Lake supports the creation of both managed and external tables:

- **Managed** Delta tables benefit from higher performance, as Fabric manages both the schema metadata and the data files.
- **External** tables allow you to store data externally, with the metadata managed by Fabric.

Create a managed table

The data files are created in the **Tables** folder.

1. Under the results returned by the first code cell, use the + Code icon to add a new code cell.

[!TIP] To see the + Code icon, move the mouse to just below and to the left of the output from the current cell. Alternatively, in the menu bar, on the Edit tab, select **+ Add code cell**.

1. To create a managed Delta table, add a new cell, enter the following code and then run the cell:

codeCopy

```
df.write.format("delta").saveAsTable("managed_products")
```

2. In the Lakehouse explorer pane, **Refresh** the Tables folder and expand the Tables node to verify that the **managed_products** table has been created.

[!NOTE] The triangle icon next to the file name indicates a Delta table.

The files for managed tables are stored in the **Tables** folder in the lakehouse. A folder named *managed_products* has been created which stores the Parquet files and delta_log folder for the table.

Create an external table

You can also create external tables, which may be stored somewhere other than the lakehouse, with the schema metadata stored in the lakehouse.

1. In the Lakehouse explorer pane, in the … menu for the **Files** folder, select **Copy ABFS path**. The ABFS path is the fully qualified path to the lakehouse Files folder.
2. In a new code cell, paste the ABFS path. Add the following code, using cut and paste to insert the abfs_path into the correct place in the code:

   codeCopy

```
df.write.format("delta").saveAsTable("external_products",
path="abfs_path/external_products")
```

3. The full path should look similar to this:

   codeCopy

```
 abfss://workspace@tenant-
onelake.dfs.fabric.microsoft.com/lakehousename.Lakehouse/Files/external_pro
ducts
```

4. **Run** the cell to save the DataFrame as an external table in the Files/external_products folder.
5. In the Lakehouse explorer pane, **Refresh** the Tables folder and expand the Tables node and verify that the external_products table has been created containing the schema metadata.
6. In the Lakehouse explorer pane, in the … menu for the Files folder, select **Refresh**. Then expand the Files node and verify that the external_products folder has been created for the table's data files.

Compare managed and external tables

Let's explore the differences between managed and external tables using the %%sql magic command.

1. In a new code cell and run the following code:

   codeCopy

   ```
   %%sql
   DESCRIBE FORMATTED managed_products;
   ```

2. In the results, view the Location property for the table. Click on the Location value in the Data type column to see the full path. Notice that the OneLake storage location ends with /Tables/managed_products.
3. Modify the DESCRIBE command to show the details of the external_products table as shown here:

   codeCopy

   ```
   %%sql
   DESCRIBE FORMATTED external_products;
   ```

4. Run the cell and in the results, view the Location property for the table. Widen the Data type column to see the full path and notice that the OneLake storage locations ends with /Files/external_products.
5. In a new code cell and run the following code:

   codeCopy

   ```
   %%sql
   DROP TABLE managed_products;
   DROP TABLE external_products;
   ```

6. In the Lakehouse explorer pane, **Refresh** the Tables folder to verify that no tables are listed in the Tables node.
7. In the Lakehouse explorer pane, **Refresh** the Files folder and verify that the external_products file has *not* been deleted. Select this folder to view the Parquet data files and _delta_log folder.

The metadata for the external table was deleted, but not the data file.

Use SQL to create a Delta table

You will now create a Delta table, using the %%sql magic command.

1. Add another code cell and run the following code:

   codeCopy

   ```
   %%sql
   CREATE TABLE products
   USING DELTA
   LOCATION 'Files/external_products';
   ```

2. In the Lakehouse explorer pane, in the ... menu for the **Tables** folder, select **Refresh**. Then expand the Tables node and verify that a new table named *products* is listed. Then expand the table to view the schema.
3. Add another code cell and run the following code:

   codeCopy

   ```
   %%sql
   SELECT * FROM products;
   ```

Explore table versioning

Transaction history for Delta tables is stored in JSON files in the delta_log folder. You can use this transaction log to manage data versioning.

1. Add a new code cell to the notebook and run the following code which implements a 10% reduction in the price for mountain bikes:

   codeCopy

   ```
   %%sql
   UPDATE products
   SET ListPrice = ListPrice * 0.9
   WHERE Category = 'Mountain Bikes';
   ```

2. Add another code cell and run the following code:

   codeCopy

```
%%sql
DESCRIBE HISTORY products;
```

The results show the history of transactions recorded for the table.

1. Add another code cell and run the following code:

   codeCopy

```
delta_table_path = 'Files/external_products'
# Get the current data
current_data = spark.read.format("delta").load(delta_table_path)
display(current_data)

# Get the version 0 data
original_data = spark.read.format("delta").option("versionAsOf",
0).load(delta_table_path)
display(original_data)
```

Two result sets are returned - one containing the data after the price reduction, and the other showing the original version of the data.

Analyze Delta table data with SQL queries

Using the SQL magic command you can use SQL syntax instead of Pyspark. Here you will create a temporary view from the products table using a `SELECT` statement.

1. Add a new code cell, and run the following code to create and display the temporary view:

   codeCopy

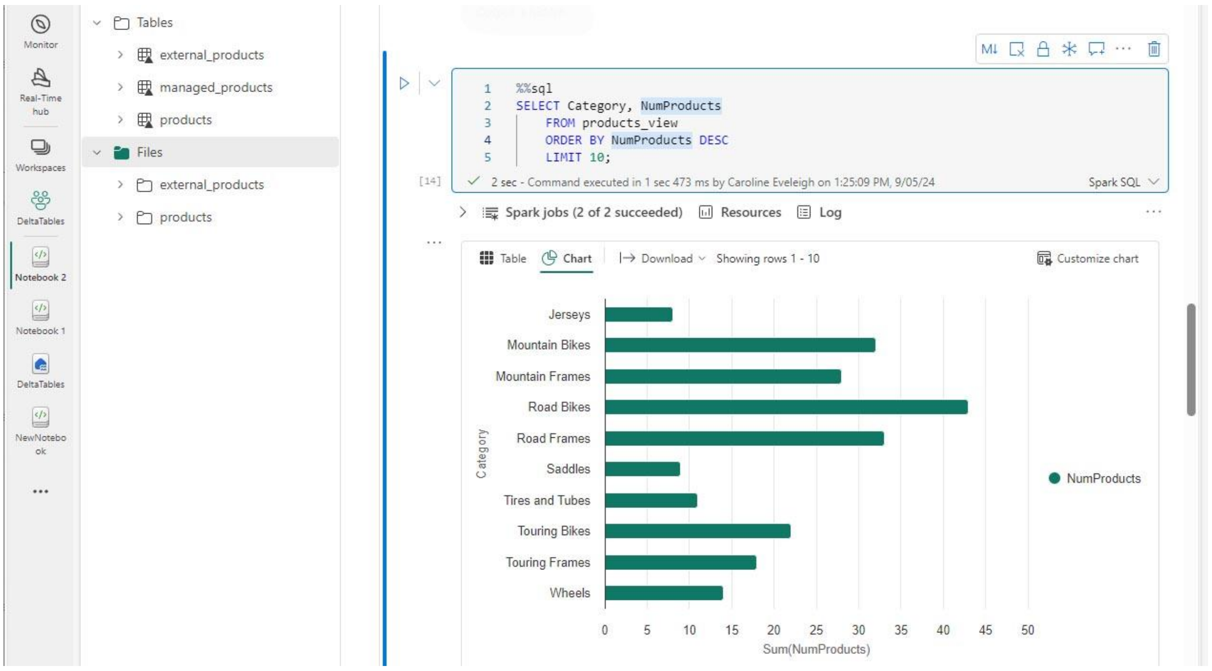
```
 %%sql
 -- Create a temporary view
 CREATE OR REPLACE TEMPORARY VIEW products_view
 AS
     SELECT Category, COUNT(*) AS NumProducts, MIN(ListPrice) AS MinPrice,
MAX(ListPrice) AS MaxPrice, AVG(ListPrice) AS AvgPrice
     FROM products
     GROUP BY Category;

 SELECT *
 FROM products_view
 ORDER BY Category;
```

2. Add a new code cell, and run the following code to return the top 10 categories by number of products:

codeCopy

```
%%sql
SELECT Category, NumProducts
FROM products_view
ORDER BY NumProducts DESC
LIMIT 10;
```

3. When the data is returned, select the **Chart** view to display a bar chart.



```
%%sql
SELECT Category, NumProducts
    FROM products_view
    ORDER BY NumProducts DESC
    LIMIT 10;
```

[14] ✓ 2 sec - Command executed in 1 sec 473 ms by Caroline Eveleigh on 1:25:09 PM, 9/05/24 Spark SQL

Alternatively, you can run a SQL query using PySpark.

1. Add a new code cell, and run the following code:

codeCopy

```
from pyspark.sql.functions import col, desc

df_products = spark.sql("SELECT Category, MinPrice, MaxPrice, AvgPrice
FROM products_view").orderBy(col("AvgPrice").desc())
display(df_products.limit(6))
```

Use Delta tables for streaming data

Delta Lake supports streaming data. Delta tables can be a sink or a source for data streams created using the Spark Structured Streaming API. In this example, you'll use a Delta table as a sink for some streaming data in a simulated internet of things (IoT) scenario.

1. Add a new code cell and add the following code and run it:

codeCopy

```
from notebookutils import mssparkutils
from pyspark.sql.types import *
from pyspark.sql.functions import *

# Create a folder
inputPath = 'Files/data/'
mssparkutils.fs.mkdirs(inputPath)

# Create a stream that reads data from the folder, using a JSON schema
jsonSchema = StructType([
StructField("device", StringType(), False),
StructField("status", StringType(), False)
])
iotstream =
spark.readStream.schema(jsonSchema).option("maxFilesPerTrigger",
1).json(inputPath)

# Write some event data to the folder
device_data = '''{"device":"Dev1","status":"ok"}
{"device":"Dev1","status":"ok"}
{"device":"Dev1","status":"ok"}
{"device":"Dev2","status":"error"}
{"device":"Dev1","status":"ok"}
{"device":"Dev1","status":"error"}
{"device":"Dev2","status":"ok"}
{"device":"Dev2","status":"error"}
{"device":"Dev1","status":"ok"}'''

mssparkutils.fs.put(inputPath + "data.txt", device_data, True)

print("Source stream created...")
```

Ensure the message *Source stream created...* is displayed. The code you just ran has created a streaming data source based on a folder to which some data has been saved, representing readings from hypothetical IoT devices.

1. In a new code cell, add and run the following code:

codeCopy

```
 # Write the stream to a delta table
 delta_stream_table_path = 'Tables/iotdevicedata'
 checkpointpath = 'Files/delta/checkpoint'
 deltastream =
iotstream.writeStream.format("delta").option("checkpointLocation",
checkpointpath).start(delta_stream_table_path)
 print("Streaming to delta sink...")
```

This code writes the streaming device data in Delta format to a folder named iotdevicedata. Because the path for the folder location in the Tables folder, a table will automatically be created for it.

1. In a new code cell, add and run the following code:

codeCopy

```
%%sql
SELECT * FROM IotDeviceData;
```

This code queries the IotDeviceData table, which contains the device data from the streaming source.

1. In a new code cell, add and run the following code:

codeCopy

```
# Add more data to the source stream
more_data = '''{"device":"Dev1","status":"ok"}
{"device":"Dev1","status":"ok"}
{"device":"Dev1","status":"ok"}
{"device":"Dev1","status":"ok"}
{"device":"Dev1","status":"error"}
{"device":"Dev2","status":"error"}
{"device":"Dev1","status":"ok"}'''

mssparkutils.fs.put(inputPath + "more-data.txt", more_data, True)
```

This code writes more hypothetical device data to the streaming source.

1. Re-run the cell containing the following code:

codeCopy

```
%%sql
SELECT * FROM IotDeviceData;
```

This code queries the IotDeviceData table again, which should now include the additional data that was added to the streaming source.

1. In a new code cell, add code to stop the stream and run the cell:

   codeCopy

```
deltastream.stop()
```

Clean up resources

In this exercise, you’ve learned how to work with Delta tables in Microsoft Fabric.

If you’ve finished exploring your lakehouse, you can delete the workspace you created for this exercise.

1. In the bar on the left, select the icon for your workspace to view all of the items it contains.
2. In the … menu on the toolbar, select **Workspace settings**.
3. In the General section, select **Remove this workspace**.