Lab: Query and Transform the data using serverless SQL pool

Duration: 30 minutes

The main task for this exercise is as follows:

1. Query data in files
2. Access external data in a database
3. Transform data using CETAS statements

Task 1: Query data in files

View files in the data lake

1. In the Azure portal, go to the **synapse-*xx*-*rg*** resource group , and select your Synapse workspace.

2. In the **Overview** page for your Synapse workspace, in the **Open Synapse Studio** card, select **Open** to open Synapse Studio in a new browser tab; signing in if prompted.

3. On the left side of Synapse Studio, use the **››** icon to expand the menu - this reveals the different pages within Synapse Studio that you'll use to manage resources and perform data analytics tasks.

4. On the **Data** page, view the **Linked** tab and verify that your workspace includes a link to your Azure Data Lake Storage Gen2 storage account, which should have a name similar to **synapse*xxxxxxx* (Primary - datalake*xxxxxxx*)**.

5. Expand your storage account and verify that it contains a file system container named **files**.

6. Select the **files** container, and note that it contains a folder named **sales**. This folder contains the data files you are going to query.

7. Open the **sales** folder and the **csv** folder it contains, and observe that this folder contains .csv files for three years of sales data.

8. Right-click any of the files and select **Preview** to see the data it contains. Note that the files do not contain a header row, so you can unselect the option to display column headers.

9. Close the preview, and then use the **↑** button to navigate back to the **sales** folder.

10. In the **sales** folder, open the **json** folder and observe that it contains some sample sales orders in .json files. Preview any of these files to see the JSON format used for a sales order.

11. Close the preview, and then use the **↑** button to navigate back to the **sales** folder.

12. In the **sales** folder, open the **parquet** folder and observe that it contains a subfolder for each year (2019-2021), in each of which a file named **orders.snappy.parquet** contains the order data for that year.

13. Return to the **sales** folder so you can see the **csv**, **json**, and **parquet** folders.

Use SQL to query CSV files

1. Select the **csv** folder, and then in the **New SQL script** list on the toolbar, select **Select TOP 100 rows**.

2. In the **File type** list, select **Text format**, and then apply the settings to open a new SQL script that queries the data in the folder.

3. In the **Properties** pane for **SQL Script 1** that is created, change the name to **Sales CSV query**, and change the result settings to show **All rows**. Then in the toolbar, select **Publish** to save the script and use the **Properties** button (which looks similar to 🗏.) on the right end of the toolbar to hide the **Properties** pane.

4. Review the SQL code that has been generated, which should be similar to this:

```
-- This is auto-generated code
 SELECT
    TOP 100 *
 FROM
    OPENROWSET(
       BULK 'https://datalakexxxxxxx.dfs.core.windows.net/files/sales/csv/',
       FORMAT = 'CSV',
       PARSER_VERSION='2.0'
    ) AS [result]
```

This code uses the OPENROWSET to read data from the CSV files in the sales folder and retrieves the first 100 rows of data.

5. In the **Connect to** list, ensure **Built-in** is selected - this represents the built-in SQL Pool that was created with your workspace.

6. On the toolbar, use the ▷ **Run** button to run the SQL code, and review the results.

7. Note the results consist of columns named C1, C2, and so on. In this example, the CSV files do not include the column headers. While it's possible to work with the data using the generic column names that have been assigned, or by ordinal position, it will be easier to understand the data if you define a tabular schema. To accomplish this, add a WITH clause to the OPENROWSET function as shown here

   (replacing *datalakexxxxxxx* with the name of your data lake storage account), and then rerun the query:

```
SELECT
    TOP 100 *
 FROM
    OPENROWSET(
       BULK 'https://datalakexxxxxxx.dfs.core.windows.net/files/sales/csv/',
       FORMAT = 'CSV',
       PARSER_VERSION='2.0'
    )
    WITH (
       SalesOrderNumber VARCHAR(10) COLLATE Latin1_General_100_BIN2_UTF8,
       SalesOrderLineNumber INT,
       OrderDate DATE,
       CustomerName VARCHAR(25) COLLATE Latin1_General_100_BIN2_UTF8,
       EmailAddress VARCHAR(50) COLLATE Latin1_General_100_BIN2_UTF8,
       Item VARCHAR(30) COLLATE Latin1_General_100_BIN2_UTF8,
       Quantity INT,
       UnitPrice DECIMAL(18,2),
       TaxAmount DECIMAL (18,2)
    ) AS [result]
```

Use SQL to query parquet files

While CSV is an easy format to use, it's common in big data processing scenarios to use file formats that are optimized for compression, indexing, and partitioning. One of the most common of these formats is *parquet*.

1. In the **files** tab contaning the file system for your data lake, return to the **sales** folder so you can see the **csv**, **json**, and **parquet** folders.

2. Select the **parquet** folder, and then in the **New SQL script** list on the toolbar, select **Select TOP 100 rows**.

3. In the **File type** list, select **Parquet format**, and then apply the settings to open a new SQL script that queries the data in the folder. The script should look similar to this:

```
-- This is auto-generated code
SELECT
    TOP 100 *
FROM
    OPENROWSET(
        BULK 'https://datalakexxxxxxx.dfs.core.windows.net/files/sales/parquet/**',
        FORMAT = 'PARQUET'
    ) AS [result]
```

4. Run the code, and note that it returns sales order data in the same schema as the CSV files you explored earlier. The schema information is embedded in the parquet file, so the appropriate column names are shown in the results.

5. Modify the code as follows (replacing *datalakexxxxxxx* with the name of your data lake storage account) and then run it.

```
SELECT YEAR(OrderDate) AS OrderYear,
       COUNT(*) AS OrderedItems
FROM
    OPENROWSET(
        BULK 'https://datalakexxxxxxx.dfs.core.windows.net/files/sales/parquet/**',
        FORMAT = 'PARQUET'
    ) AS [result]
GROUP BY YEAR(OrderDate)
ORDER BY OrderYear
```

6. Note that the results include order counts for all three years - the wildcard used in the BULK path causes the query to return data from all subfolders.

The subfolders reflect *partitions* in the parquet data, which is a technique often used to optimize performance for systems that can process multiple partitions of data in parallel. You can also use partitions to filter the data.

7. Modify the code as follows (replacing *datalakexxxxxxx* with the name of your data lake storage account) and then run it.

```
SELECT YEAR(OrderDate) AS OrderYear,
      COUNT(*) AS OrderedItems
 FROM
    OPENROWSET(
       BULK 'https://datalakexxxxxxx.dfs.core.windows.net/files/sales/parquet/year=*/',
       FORMAT = 'PARQUET'
    ) AS [result]
 WHERE [result].filepath(1) IN ('2019', '2020')
 GROUP BY YEAR(OrderDate)
 ORDER BY OrderYear
```

8. Review the results and note that they include only the sales counts for 2019 and 2020. This filtering is achieved by inclusing a wildcard for the partition folder value in the BULK path (*year=**) and a WHERE clause based on the *filepath* property of the results returned by OPENROWSET (which in this case has the alias *[result]*).

9. Name your script **Sales Parquet query**, and publish it. Then close the script pane.

Use SQL to query JSON files

JSON is another popular data format, so it;s useful to be able to query .json files in a serverless SQL pool.

1. In the **files** tab containing the file system for your data lake, return to the **sales** folder so you can see the **csv**, **json**, and **parquet** folders.
2. Select the **json** folder, and then in the **New SQL script** list on the toolbar, select **Select TOP 100 rows**.
3. In the **File type** list, select **Text format**, and then apply the settings to open a new SQL script that queries the data in the folder. The script should look similar to this:

```
-- This is auto-generated code
 SELECT
    TOP 100 *
 FROM
    OPENROWSET(
       BULK 'https://datalakexxxxxxx.dfs.core.windows.net/files/sales/json/',
       FORMAT = 'CSV',
       PARSER_VERSION = '2.0'
    ) AS [result]
```

The script is designed to query comma-delimited (CSV) data rather then JSON, so you need to make a few modifications before it will work successfully.

4. Modify the script as follows (replacing *datalakexxxxxxx* with the name of your data lake storage account) to:
    - Remove the parser version parameter.
    - Add parameters for field terminator, quoted fields, and row terminators with the character code *0x0b*.
    - Format the results as a single field containing the JSON row of data as an NVARCHAR(MAX) string.

```
SELECT
    TOP 100 *
 FROM
    OPENROWSET(
       BULK 'https://datalakexxxxxxx.dfs.core.windows.net/files/sales/json/',
       FORMAT = 'CSV',
       FIELDTERMINATOR ='0x0b',
       FIELDQUOTE = '0x0b',
       ROWTERMINATOR = '0x0b'
    ) WITH (Doc NVARCHAR(MAX)) as rows
```

5. Run the modified code and observe that the results include a JSON document for each order.

6. Modify the query as follows (replacing *datalakexxxxxxx* with the name of your data lake storage account) so that it uses the JSON_VALUE function to extract individual field values from the JSON data.

```
SELECT JSON_VALUE(Doc, '$.SalesOrderNumber') AS OrderNumber,
       JSON_VALUE(Doc, '$.CustomerName') AS Customer,
       Doc
 FROM
    OPENROWSET(
       BULK 'https://datalakexxxxxxx.dfs.core.windows.net/files/sales/json/',
       FORMAT = 'CSV',
       FIELDTERMINATOR ='0x0b',
       FIELDQUOTE = '0x0b',
       ROWTERMINATOR = '0x0b'
    ) WITH (Doc NVARCHAR(MAX)) as rows
```

7. Name your script **Sales JSON query**, and publish it. Then close the script pane.

Task 2: Access external data in a database

So far, you've used the OPENROWSET function in a SELECT query to retrieve data from files in a data lake. The queries have been run in the context of the **master** database in your serverless SQL pool. This approach is fine for an initial exploration of the data, but if you plan to create more complex queries it may be more effective to use the *PolyBase* capability of Synapse SQL to create objects in a database that reference the external data location.

Create an external data source

By defining an external data source in a database, you can use it to reference the data lake location where the files are stored.

1. In Synapse Studio, on the **Develop** page, in the **+** menu, select **SQL script**.
2. In the new script pane, add the following code (replacing *datalakexxxxxxx* with the name of your data lake storage account) to create a new database and add an external data source to it.

```
CREATE DATABASE Sales
  COLLATE Latin1_General_100_BIN2_UTF8;
GO;

Use Sales;
GO;

CREATE EXTERNAL DATA SOURCE sales_data WITH (
    LOCATION = 'https://datalakexxxxxxx.dfs.core.windows.net/files/sales/'
);
GO;
```

3. Modify the script properties to change its name to **Create Sales DB**, and publish it.
4. Ensure that the script is connected to the **Built-in** SQL pool and the **master** database, and then run it.
5. Switch back to the **Data** page and use the ↻ button at the top right of Synapse Studio to refresh the page. Then view the **Workspace** tab in the **Data** pane, where a **SQL database** list is now displayed. Expand this list to verify that the **Sales** database has been created.
6. Expand the **Sales** database, its **External Resources** folder, and the **External data sources** folder under that to see the **sales_data** external data source you created.

7. In the **…** menu for the **Sales** database, select **New SQL script** > **Empty script**. Then in the new script pane, enter and run the following query:

```
SELECT *
FROM
    OPENROWSET(
        BULK 'csv/*.csv',
        DATA_SOURCE = 'sales_data',
        FORMAT = 'CSV',
        PARSER_VERSION = '2.0'
    ) AS orders
```

   The query uses the external data source to connect to the data lake, and the OPENROWSET function now only need to reference the relative path to the .csv files.

8. Modify the code as follows to query the parquet files using the data source.

```
SELECT *
FROM
    OPENROWSET(
        BULK 'parquet/year=*/*.snappy.parquet',
        DATA_SOURCE = 'sales_data',
        FORMAT='PARQUET'
    ) AS orders
WHERE orders.filepath(1) = '2019'
```

Create an external table

The external data source makes it easier to access the files in the data lake, but most data analysts using SQL are used to working with tables in a database. Fortunately, you can also define external file formats and external tables that encapsulate rowsets from files in database tables.

1. Replace the SQL code with the following statement to define an external data format for CSV files, and an external table that references the CSV files, and run it:

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

CREATE EXTERNAL TABLE dbo.orders
(
    SalesOrderNumber VARCHAR(10),
    SalesOrderLineNumber INT,
    OrderDate DATE,
    CustomerName VARCHAR(25),
    EmailAddress VARCHAR(50),
    Item VARCHAR(30),
    Quantity INT,
    UnitPrice DECIMAL(18,2),
    TaxAmount DECIMAL (18,2)
)
WITH
(
    DATA_SOURCE =sales_data,
    LOCATION = 'csv/*.csv',
    FILE_FORMAT = CsvFormat
);
GO
```

2. Refresh and expand the **External tables** folder in the **Data** pane and confirm that a table named **dbo.orders** has been created in the **Sales** database.

3. In the **…** menu for the **dbo.orders** table, select **New SQL script** > **Select TOP 100 rows**.

4. Run the SELECT script that has been generated, and verify that it retrieves the first 100 rows of data from the table, which in turn references the files in the data lake.

Visualize query results

Now that you've explored various ways to query files in the data lake by using SQL queries, you can analyze the results of these queries to gain insights into the data. Often, insights are easier to uncover by visualizing the query results in a chart; which you can easily do by using the integrated charting functionality in the Synapse Studio query editor.

1. On the **Develop** page, create a new empty SQL query.

2. Ensure that the script is connected to the **Built-in** SQL pool and the **Sales** database.

3. Enter and run the following SQL code:

```
SELECT YEAR(OrderDate) AS OrderYear,
    SUM((UnitPrice * Quantity) + TaxAmount) AS GrossRevenue
FROM dbo.orders
GROUP BY YEAR(OrderDate)
ORDER BY OrderYear;
```

4. In the **Results** pane, select **Chart** and view the chart that is created for you; which should be a line chart.

5. Change the **Category column** to **OrderYear** so that the line chart shows the revenue trend over the three year period from 2019 to 2021:

6. Switch the **Chart type** to **Column** to see the yearly revenue as a column chart:

7. Experiment with the charting functionality in the query editor. It offers some basic charting capabilities that you can use while interactively exploring data, and you can save charts as images to include in reports. However, functionality is limited compared to enterprise data visualization tools such as Microsoft Power BI.

Task 3: Transform data using CREATE EXTERAL TABLE AS SELECT (CETAS) statements

A simple way to use SQL to transform data in a file and persist the results in another file is to use a CREATE EXTERNAL TABLE AS SELECT (CETAS) statement. This statement creates a table based on the requests of a query, but the data for the table is stored as files in a data lake. The transformed data can then be queried through the external table, or accessed directly in the file system (for example, for inclusion in a downstream process to load the transformed data into a data warehouse).

Create an external file format to store the transformed data

To use these objects to work with external tables, you need to create them in a database other than the default **master** database.

1. In Synapse Studio, on the **Develop** page, in the **+** menu, select **SQL script**.
2. In the new script pane, add the following to create a new external file format.

```
Use Sales;
GO;

-- Format for table files
 CREATE EXTERNAL FILE FORMAT ParquetFormat
    WITH (
          FORMAT_TYPE = PARQUET,
          DATA_COMPRESSION = 'org.apache.hadoop.io.compress.SnappyCodec'
       );
 GO;
```

3. Ensure that the script is connected to the **Built-in** SQL pool and the **Sales** database, and then run it.

Create an External table

1. In Synapse Studio, on the **Develop** page, in the **+** menu, select **SQL script**.
2. In the new script pane, add the following code to retrieve and aggregate data from the CSV sales files by using the external data source - noting that the **BULK** path is relative to the folder location on which the data source is defined:

```
USE Sales;
 GO;

 SELECT Item AS Product,
       SUM(Quantity) AS ItemsSold,
       ROUND(SUM(UnitPrice) - SUM(TaxAmount), 2) AS NetRevenue
 FROM
    OPENROWSET(
       BULK 'csv/*.csv',
       DATA_SOURCE = 'sales_data',
       FORMAT = 'CSV',
       PARSER_VERSION = '2.0',
       HEADER_ROW = TRUE
    ) AS orders
 GROUP BY Item;
```

3. Run the script.

4. Modify the SQL code to save the results of query in an external table, like this:

```
 CREATE EXTERNAL TABLE ProductSalesTotals
    WITH (
       LOCATION = 'productsales/',
       DATA_SOURCE = sales_data,
       FILE_FORMAT = ParquetFormat
    )
 AS
 SELECT Item AS Product,
    SUM(Quantity) AS ItemsSold,
    ROUND(SUM(UnitPrice) - SUM(TaxAmount), 2) AS NetRevenue
 FROM
    OPENROWSET(
       BULK 'csv/*.csv',
       DATA_SOURCE = 'sales_data',
       FORMAT = 'CSV',
       PARSER_VERSION = '2.0',
       HEADER_ROW = TRUE
    ) AS orders
 GROUP BY Item;
```

5. Run the script. This time there's no output, but the code should have created an external table based on the results of the query.

6. Name the script **Create ProductSalesTotals table** and publish it.

7. On the **data** page, in the **Workspace** tab, view the contents of the **External tables** folder for the **Sales** SQL database to verify that a new table named **ProductSalesTotals** has been created.

8. In the **…** menu for the **ProductSalesTotals** table, select **New SQL script** > **Select TOP 100 rows**. Then run the resulting script and verify that it returns the aggregated product sales data.

9. On the **files** tab containing the file system for your data lake, view the contents of the **sales** folder (refreshing the view if necessary) and verify that a new **productsales** folder has been created.

10. In the **productsales** folder, observe that a file with a name similar to ABC123DE—-.parquet has been created. This file contains the aggregated product sales data. To prove this, you can select the file and use the **New SQL script** > **Select TOP 100 rows** menu to query it directly.