Work with JSON data in SQL pools

Note:

You are not required to complete the processes, tasks, activities, or steps presented in this example. The various samples provided are for illustrative purposes only and it's likely that if you try this out you will encounter issues in your system.

Synapse dedicated SQL Pools supports JSON format data to be stored using standard NVARCHAR table columns. The JSON format enables representation of complex or hierarchical data structures in tables. It allows to transform arrays of JSON objects into table format. The performance of JSON data can be optimized by using columnstore indexes and memory optimized tables.

Insert JSON data - JSON data can be inserted using the usual T-SQL INSERT statements.

Read JSON data - JSON data can be read using the following T-SQL functions and provides the ability to perform aggregation and filter on JSON values.

- ISJSON – verify if text is valid JSON
- JSON_VALUE – extract a scalar value from a JSON string
- JSON_QUERY – extract a JSON object or array from a JSON string

Modify JSON data - JSON data can be modified and queried using the following T-SQL functions providing ability to update JSON string using T-SQL and convert hierarchical data into flat tabular structure.

- JSON_MODIFY – modifies a value in a JSON string
- OPENJSON – convert JSON collection to a set of rows and columns

You can also query JSON files using SQL serverless. The query's objective is to read the following type of JSON files using **OPENROWSET**.

- Standard JSON files where multiple JSON documents are stored as a JSON array.
- Line-delimited JSON files, where JSON documents are separated with new-line character. Common extensions for these types of files are jsonl, ldjson, and ndjson.

Read JSON documents

The easiest way to see the content of your JSON file is to provide the file URL to the OPENROWSET function, specify **csv FORMAT**, and set the values of **0x0b** for the **fieldterminator** and **fieldquote** variables. If you need to read line-delimited JSON files, then this is enough. If you have classic JSON file, you would need to set values **0x0b** for **rowterminator**. The OPENROWSET function will parse JSON and return every document in the following format:

```
{"date_rep":"2020-07-
24","day":24,"month":7,"year":2020,"cases":3,"deaths":0,"geo_id":"AF"}
{"date_rep":"2020-07-
25","day":25,"month":7,"year":2020,"cases":7,"deaths":0,"geo_id":"AF"}
{"date_rep":"2020-07-
26","day":26,"month":7,"year":2020,"cases":4,"deaths":0,"geo_id":"AF"}
{"date_rep":"2020-07-
27","day":27,"month":7,"year":2020,"cases":8,"deaths":0,"geo_id":"AF"}
```

If the file is publicly available, or if your Azure Active Directory identity can access this file, you should see the content of the file using the query like the one shown in the following examples.

Read JSON files

The following two sample queries read JSON and line-delimited JSON files (.JSONL). The first query is reading line-delimited JSON file and you can find example below.

```
select top 10 *
from
    openrowset(
        bulk 'https://pandemicdatalake.blob.core.windows.net/public/curated/
covid-19/ecdc_cases/latest/ecdc_cases.jsonl',
        format = 'csv',
        fieldterminator ='0x0b',
        fieldquote = '0x0b'
    ) with (doc nvarchar(max)) as rows
```

The second query is reading standard JSON file, and you can find that example below.

```
select top 10 *
from
    openrowset(
        bulk 'https://pandemicdatalake.blob.core.windows.net/public/curated/
covid-19/ecdc_cases/latest/ecdc_cases.json',
        format = 'csv',
        fieldterminator ='0x0b',
        fieldquote = '0x0b',
        rowterminator = '0x0b' --> You need to override rowterminator to read cl
assic JSON
    ) with (doc nvarchar(max)) as rows
```

This second example is reading the classic JSON file:

```
select top 10 *
from
    openrowset(
        bulk 'https://pandemicdatalake.blob.core.windows.net/public/curated/
covid-19/ecdc_cases/latest/ecdc_cases.json',
        format = 'csv',
        fieldterminator ='0x0b',
        fieldquote = '0x0b',
        rowterminator = '0x0b' --> You need to override rowterminator to read cl
assic JSON
    ) with (doc nvarchar(max)) as rows
```

The JSON document in the preceding sample query includes an array of objects. The query returns each object as a separate row in the result set. Make sure that you can access this file. If your file is protected with SAS key or custom identity, you would need to set up server level credential for sql login

Define a data source

The previous example uses a full path to the file. As an alternative, you can create an external data source with the location that points to the root folder of the storage, and use that data source and the relative path to the file in the OPENROWSET function:

```
create external data source covid
with (location = 'https://pandemicdatalake.blob.core.windows.net/public/curated/covid-19/ecdc_cases');
go
select top 10 *
from
    openrowset(
        bulk 'latest/ecdc_cases.jsonl',
        data_source = 'covid',
        format = 'csv',
        fieldterminator ='0x0b',
        fieldquote = '0x0b'
    ) with (doc nvarchar(max)) as rows
go
select top 10 *
from
    openrowset(
        bulk 'latest/ecdc_cases.json',
        data_source = 'covid',
        format = 'csv',
        fieldterminator ='0x0b',
        fieldquote = '0x0b',
        rowterminator = '0x0b' --> You need to override rowterminator to read cl
assic JSON
    ) with (doc nvarchar(max)) as rows
```

If a data source is protected with SAS key or custom identity, you can configure data source with database scoped credential

In the following sections, you can see how to query various types of JSON files.

Parse JSON documents

The queries in the previous examples return every JSON document as a single string in a separate row of the result set. You can use functions JSON_VALUE and OPENJSON to parse the values in JSON documents and return them as relational values, as it's shown in the following example:

PARSE JSON DOCUMENTS

date_rep	**cases**	**geo_id**
2020-07-24	3	AF
2020-07-25	7	AF
2020-07-26	4	AF
2020-07-27	8	AF

Which is a json file containing the following structure:

```
{
    "date_rep":"2020-07-24",
    "day":24,
    "month":7,
    "year":2020,
    "cases":13,
    "deaths":0,
    "countries_and_territories":"Afghanistan",
    "geo_id":"AF",
    "country_territory_code":"AFG",
    "continent_exp":"Asia",
    "load_date":"2020-07-25 00:05:14",
    "iso_country":"AF"
}
```

Note

If these documents are stored as line-delimited JSON, you need to set **FIELDTERMINATOR** and **FIELDQUOTE** to **0x0b**. If you have standard JSON format you need to set **ROWTERMINATOR** to **0x0b**.

Query JSON files using JSON_VALUE

The following query below shows you how to use JSON_VALUE to retrieve scalar values (title, publisher) from JSON documents:

```
select
    JSON_VALUE(doc, '$.date_rep') AS date_reported,
    JSON_VALUE(doc, '$.countries_and_territories') AS country,
    JSON_VALUE(doc, '$.cases') as cases,
    doc
from
    openrowset(
        bulk 'latest/ecdc_cases.jsonl',
        data_source = 'covid',
        format = 'csv',
        fieldterminator ='0x0b',
        fieldquote = '0x0b'
    ) with (doc nvarchar(max)) as rows
order by JSON_VALUE(doc, '$.geo_id') desc
```

Query JSON files using OPENJSON

The following query uses OPENJSON. It will retrieve COVID statistics reported in Serbia:

```
select *
from
    openrowset(
        bulk 'latest/ecdc_cases.jsonl',
        data_source = 'covid',
        format = 'csv',
        fieldterminator ='0x0b',
        fieldquote = '0x0b'
    ) with (doc nvarchar(max)) as rows
    cross apply openjson (doc)
        with ( date_rep datetime2,
                   cases int,
                   fatal int '$.deaths',
                   country varchar(100) '$.countries_and_territories')
where country = 'Serbia'
order by country, date_rep desc;
```