How to parse JSON in SQL Server

September 15, 2020 by Esat Erkec

In this article, we will learn how to parse and query JSON in SQL Server with the help of the OPENJSON function. Firstly, we will briefly look at the data structure of the JSON and then we will learn details of the parsing and querying JSON data using the OPENJSON function.

What is JSON?

JSON (**J**ava **S**cript **O**bject **N**otation) is a lightweight text-based data format that helps to interchange data easily between modern applications. At the same time, JSON is used by the NoSQL (Microsoft Azure Cosmos DB, CouchDB, etc.) databases to store the unstructured data. The small data size is the main feature of the JSON and this property enables to an interchange of the data easily between the applications. At the same time, JSON offers the following advantages:

- The JSON data structure is easily readable by humans and machines
- JSON has a compact data structure and it does not include unnecessary data notations
- JSON has extensive usage. All modern programming languages and application platforms support working with JSON
- JSON has a straightforward syntax

The JSON data structure is essentially based on a key-value pair format. The keys must be string data type and the values data types must be in JSON data type. JSON supports the following data types :

- string
- number
- boolean
- null
- object
- array

At the same time, a JSON object can contain two different data structures:

- An object type can contain multiple key-value pairs inside the JSON text
- An array can contain single or multiple values

- An array can contain single or multiple values

The following JSON represents some attributes of a car brand. The **color** attribute represents an array and the **Model** attribute represents an object in this JSON object.

```
{
"owner": null,
"brand": "BMW",
"year": 2020,
"status": false,
"color": [
    "red",
    "white",
    "yellow"
],
"Model": {
    "name": "BMW M4",
    "Fuel Type": "Petrol",
    "TransmissionType": "Automatic",
    "Turbo Charger": "true",
    "Number of Cylinder": 4
}
}
```

OPENJSON() function parses JSON in SQL Server

As we mentioned in the previous section, JSON is used widely by the applications. In this context, the conversion of the JSON data into the relational format is becoming more important.

OPENJSON is a table-valued function that helps to parse JSON in SQL Server and it returns the data values and types of the JSON text in a table format. Now, we will look at the syntax of this function.

```
OPENJSON( jsonExpression  [, jsonPath ] )
[ WITH (column_mapping_definition1
        [,column_mapping_definition2]
        [,… column_mapping_definitionN])
]
```

The **jsonExpression** is an input parameter that specifies the JSON text that will be parsed by the OPENJSON function.

The **jsonPath** is an optional parameter and it is used to seek a specified JSON expression in the JSON text and the OPENJSON function parses only this part of the JSON text.

The **WITH** clause is an optional parameter that can be used to explicitly specifies the schema.

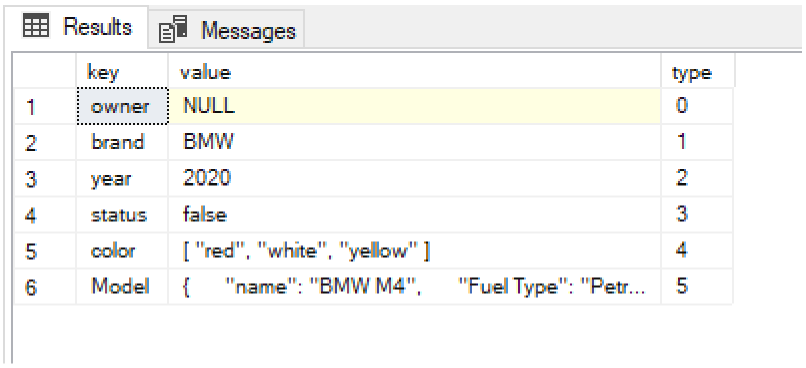
In the following example, we can see a very simple usage of the OPENJSON function. It will parse JSON in SQL Server and return the parsed JSON in the default schema format.

```
DECLARE @json NVarChar(2048) = N'{
    "owner": null,
  "brand": "BMW",
  "year": 2020,
   "status": false,
  "color": [ "red", "white", "yellow" ],


  "Model": {
    "name": "BMW M4",
    "Fuel Type": "Petrol",
    "TransmissionType": "Automatic",
    "Turbo Charger": "true",
    "Number of Cylinder": 4

  }
}';

SELECT * FROM OpenJson(@json);
```



	key	value	type
1	owner	NULL	0
2	brand	BMW	1
3	year	2020	2
4	status	false	3
5	color	["red", "white", "yellow"]	4
6	Model	{ "name": "BMW M4", "Fuel Type": "Petr...	5

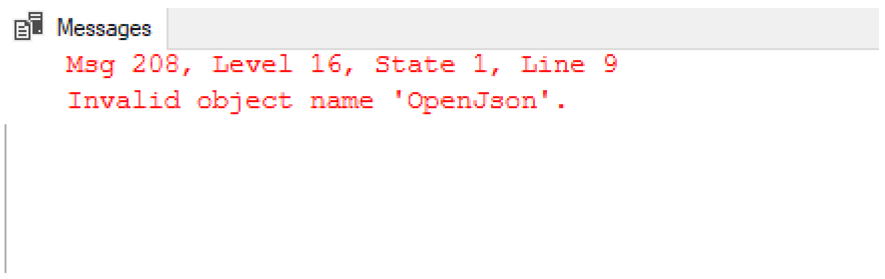
In this result set, we can observe that the OPENJSON function is executed with the default schema and it has returned three columns:

- The **key** column indicates the name of the key
- The **value** column shows the value of the key
- The **type** column indicates the data types of the key column through the numbers. The following table illustrates the possible values of the type column and their data type explanations

Type column	JSON data type
0	null
1	string
2	int
3	true/false
4	array

5	object

Tip: *OPENJSON function does not work in the databases with compatibility level less than 130 and returns the following error. So we can not use the OPENJSON function to parse JSON in SQL Server*



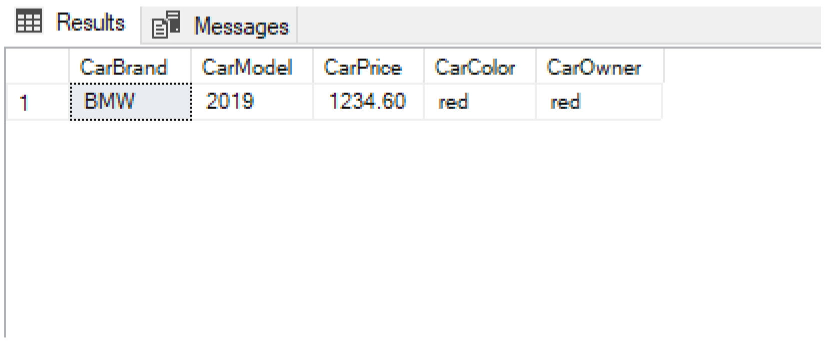
```
Msg 208, Level 16, State 1, Line 9
Invalid object name 'OpenJson'.
```

Using OPENJSON with explicit schema

The returning columns of the JSON result set can be defined by the users. In this usage method, we need to specify output columns and their types and then we can pass this user-defined schema to OPENJSON through WITH keyword.

```
DECLARE @json NVarChar(2048) = N'{
"brand": "BMW",
"year": 2019,
"price": 1234.6,
"color": "red",
"owner": null
}'

SELECT * FROM OpenJson(@json)
WITH (CarBrand VARCHAR(100) '$.brand',
CarModel INT '$.year',
CarPrice MONEY '$.price',
CarColor VARCHAR(100) '$.color',
CarOwner NVARCHAR(200) '$.owner'
)
```

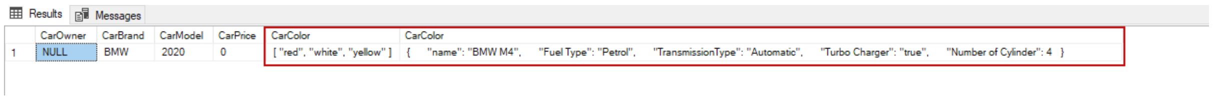


	CarBrand	CarModel	CarPrice	CarColor	CarOwner
1	BMW	2019	1234.60	red	red

We need to use **AS JSON** keyword to specify JSON objects or arrays which are contained in the JSON text. In addition, the column data type must be NVARCHAR(MAX).

```
DECLARE @json NVarChar(2048) = N'{
"owner": null,
"brand": "BMW",
"year": 2020,
    "status": false,
"color": [ "red", "white", "yellow" ],


"Model": {
    "name": "BMW M4",
    "Fuel Type": "Petrol",
    "TransmissionType": "Automatic",
    "Turbo Charger": "true",
    "Number of Cylinder": 4
}
}';
SELECT * FROM OpenJson(@json)
WITH (CarOwner NVARCHAR(200) '$.owner',
CarBrand NVARCHAR(200) '$.brand',
CarModel INT '$.year',
CarPrice BIT '$.status',
CarColor NVARCHAR(MAX) '$.color' AS JSON,
CarColor NVARCHAR(MAX) '$.Model' AS JSON
)
```



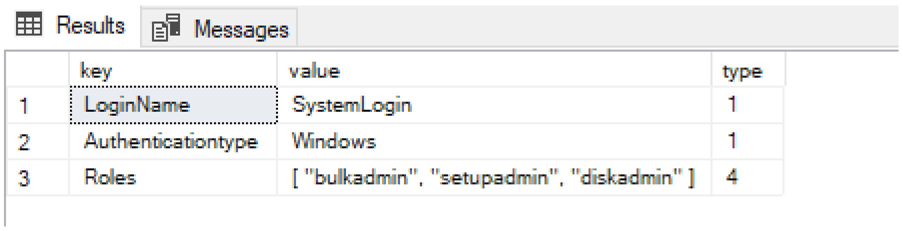
	CarOwner	CarBrand	CarModel	CarPrice	CarColor	CarColor
1	NULL	BMW	2020	0	["red", "white", "yellow"]	{ "name": "BMW M4", "Fuel Type": "Petrol", "TransmissionType": "Automatic", "Turbo Charger": "true", "Number of Cylinder": 4 }

How to parse the JSON arrays with OPENJSON?

JSON arrays are used to store multiple key values. The JSON arrays are specified with square brackets (**[ ]**) in the JSON text and values are separated with a comma. When we parse a JSON text which includes an array through OPENJSON, the result set will look like as below.

```
DECLARE @json NVarChar(max)='{
    "LoginName" : "SystemLogin",
    "Authenticationtype" : "Windows",
    "Roles" : [ "bulkadmin", "setupadmin", "diskadmin" ]
}'

select * from OPENJSON(@json)
```

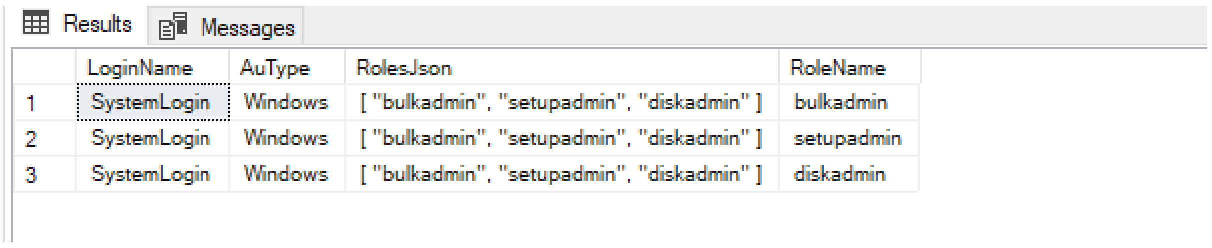


	key	value	type
1	LoginName	SystemLogin	1
2	Authenticationtype	Windows	1
3	Roles	["bulkadmin", "setupadmin", "diskadmin"]	4

As we can see, the OPENJSON function has parsed the root level objects but the array has returned as JSON text. When we want to convert this JSON array data into the relational format, we need to use the OPENJSON function again and join to the root level table.

```
DECLARE @json NVarChar(max)='{
    "LoginName" : "SystemLogin",
    "Authenticationtype" : "Windows",
    "Roles":[ "bulkadmin", "setupadmin", "diskadmin" ]}'

select LoginName,AuType,RolesJson,RoleName from OPENJSON(@json)
WITH(
LoginName VARCHAR(20) '$.LoginName' ,
AuType VARCHAR(20) '$.Authenticationtype',
RolesJson nvarchar(MAX)  '$.Roles' AS JSON)
CROSS APPLY OPENJSON(RolesJson) WITH (
RoleName VARCHAR(20) '$')
```



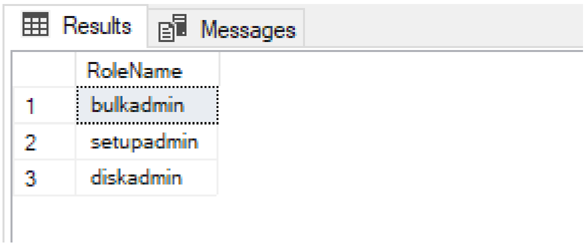
Results Messages

	LoginName	AuType	RolesJson	RoleName
1	SystemLogin	Windows	["bulkadmin", "setupadmin", "diskadmin"]	bulkadmin
2	SystemLogin	Windows	["bulkadmin", "setupadmin", "diskadmin"]	setupadmin
3	SystemLogin	Windows	["bulkadmin", "setupadmin", "diskadmin"]	diskadmin

In another way, we can use the jsonPath parameter to covert the JSON array into a relational format.

```
DECLARE @json NVarChar(max)='{
    "LoginName" : "SystemLogin",
    "Authenticationtype" : "Windows",
    "Roles":[ "bulkadmin", "setupadmin", "diskadmin" ]}'

select *from OPENJSON(@json,'$.Roles')
WITH(
RoleName VARCHAR(20) '$' )
```



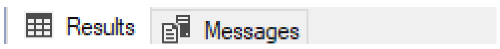
Results Messages

	RoleName
1	bulkadmin
2	setupadmin
3	diskadmin

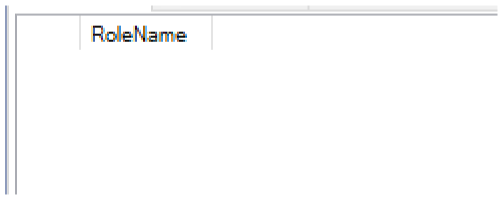
Tip: *The* ***lax*** *and* ***strict*** *are the two JSON path mode. In the* ***lax*** *mode, if the specified JSON path expression does not find in the JSON text it does not return an error and it is the default path mode. For example, there is not any* ***Unknown*** *attribute in the JSON text but the OPENJSON function does not return any error*

```
DECLARE @json NVarChar(max)='{
    "LoginName" : "SystemLogin",
    "Authenticationtype" : "Windows",
    "Roles":[ "bulkadmin", "setupadmin", "diskadmin" ]}'

select *from OPENJSON(@json,'$.Unknown')
WITH(
RoleName VARCHAR(20) '$' )
```



Results Messages

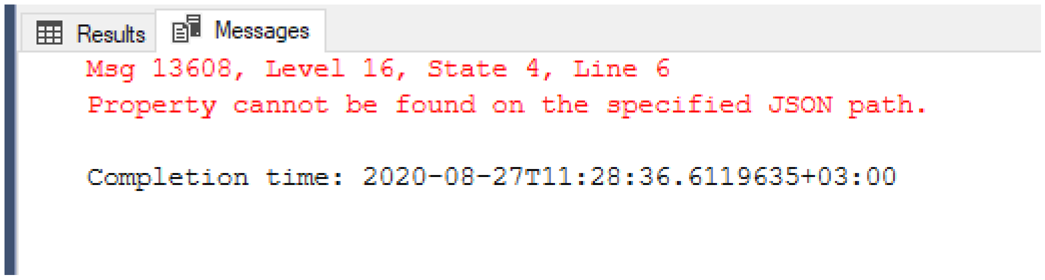


```
RoleName
```

In the **strict** mode, if the specified JSON path expression does not find in the JSON text it returns an error.

```
DECLARE @json NVarChar(max)='{
    "LoginName" : "SystemLogin",
    "Authenticationtype" : "Windows",
    "Roles":[ "bulkadmin", "setupadmin", "diskadmin" ]}'

select *from OPENJSON(@json,'strict $.Unknown')
WITH(
RoleName VARCHAR(20) '$' )
```



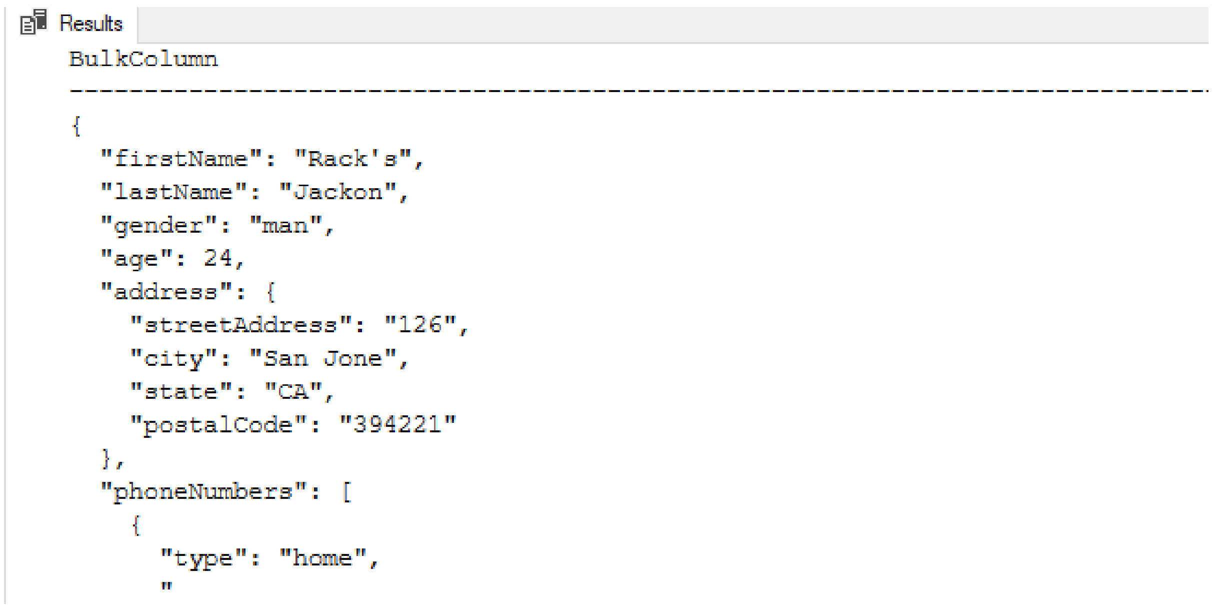
```
Msg 13608, Level 16, State 4, Line 6
Property cannot be found on the specified JSON path.

Completion time: 2020-08-27T11:28:36.6119635+03:00
```

How to parse a JSON file with OPENJSON?

OPENROWSET function is used to read data from files from the file system. The below query will read the JSON file contents in the specified file path.

```
SELECT *
FROM OPENROWSET (BULK 'C:\sample-json-file.json', SINGLE_CLOB) as JsonFile
```

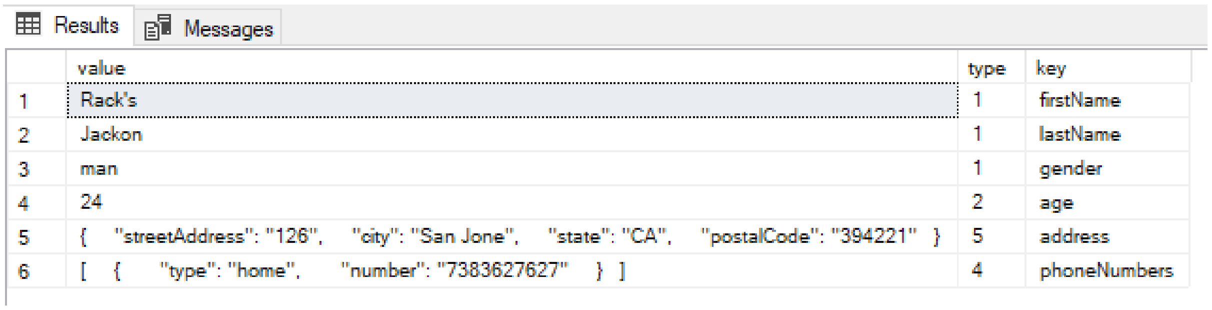


```
BulkColumn
---------------------------------------------------------------------------------
{
  "firstName": "Rack's",
  "lastName": "Jackon",
  "gender": "man",
  "age": 24,
  "address": {
    "streetAddress": "126",
    "city": "San Jone",
    "state": "CA",
    "postalCode": "394221"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "
```

```
(1 row affected)
```

Now, we will convert the JSON data to the relational format with the help of the OPENJSON function.

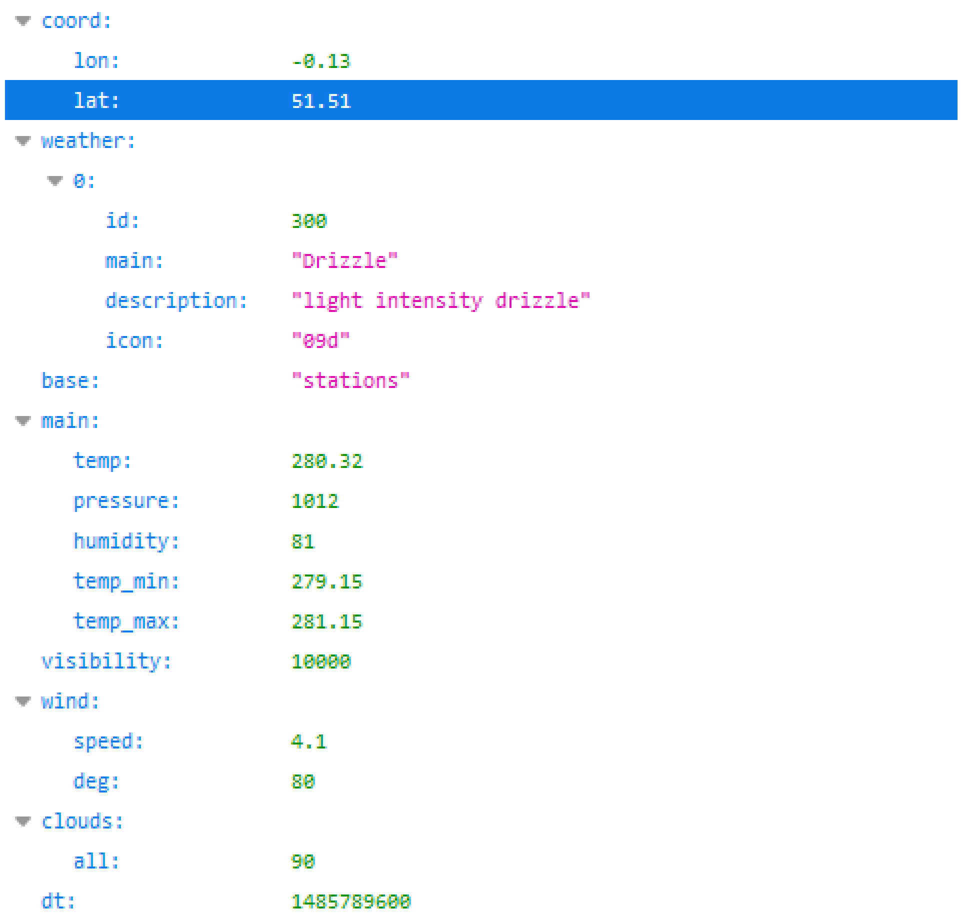
```
SELECT [value] ,[type],[key]
FROM OPENROWSET (BULK 'C:\sample-json-file.json', SINGLE_CLOB) as JsonFile
CROSS APPLY OPENJSON(BulkColumn)
```


Results Messages

	value	type	key
1	Rack's	1	firstName
2	Jackon	1	lastName
3	man	1	gender
4	24	2	age
5	{ "streetAddress": "126", "city": "San Jone", "state": "CA", "postalCode": "394221" }	5	address
6	[{ "type": "home", "number": "7383627627" }]	4	phoneNumbers

How to call JSON API in SQL Server?

The JSON API enables us to communicate between two application platforms in the JSON data format. These API's results are returns in the JSON format and accept the data that's formatted as JSON. Such as, this JSON API returns weather data information for London.


```
▼ coord:
    lon:              -0.13
    lat:              51.51
▼ weather:
  ▼ 0:
      id:             300
      main:           "Drizzle"
      description:    "light intensity drizzle"
      icon:           "09d"
  base:               "stations"
▼ main:
    temp:             280.32
    pressure:         1012
    humidity:         81
    temp_min:         279.15
    temp_max:         281.15
  visibility:         10000
▼ wind:
    speed:            4.1
    deg:              80
▼ clouds:
    all:              90
  dt:                 1485789600
```

```
▼ sys:
    type:          1
    id:            5091
```

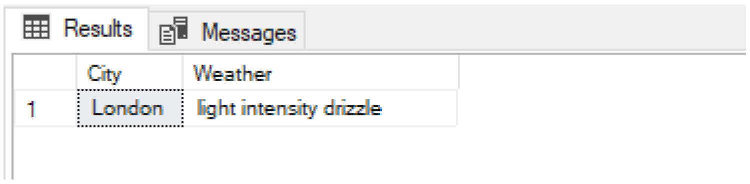
In order to communicate any JSON API from the SQL Server, we can use OLE Automation Stored Procedures. These procedures provides to access OLE or COM objects directly over SQL Server. By default OLE Automation Stored Procedures are disabled and we need to enable them.

```
EXEC sp_configure 'show advanced options', 1;
RECONFIGURE;
GO
EXEC sp_configure 'Ole Automation Procedures', 1;
RECONFIGURE;
GO
```

The following query will get weather information result from the HTTP server and then convert the JSON text into the table.

```
Declare @WinHttpObject as Int;
Declare @ResponseJsonText as Varchar(8000);

Exec sp_OACreate 'WinHttp.WinHttpRequest.5.1', @WinHttpObject OUT;
Exec sp_OAMethod @WinHttpObject, 'open', NULL, 'get',
'https://samples.openweathermap.org/data/2.5/weather?q=London,uk&appid=439d4b804bc
8187953eb36d2a8c26a02',
'false'
Exec sp_OAMethod @WinHttpObject, 'send'
Exec sp_OAMethod @WinHttpObject, 'responseText', @ResponseJsonText OUTPUT
Exec sp_OADestroy @WinHttpObject
IF ISJSON(@ResponseJsonText)=1
BEGIN
SELECT City,Weather FROM OPENJSON(@ResponseJsonText)
WITH(City VARCHAR(100) '$.name')
CROSS APPLY
OPENJSON(@ResponseJsonText,'$.weather') WITH(Weather  VARCHAR(100) '$.description'
)
END
```



	City	Weather
1	London	light intensity drizzle

Now, we will explain the above query line by line.

```
Declare @WinHttpObject as Int;
Declare @ResponseJsonText as Varchar(8000);
Exec sp_OACreate 'WinHttp.WinHttpRequest.5.1', @WinHttpObject OUT;
```

As a first step, we declared the required variables and create an instance of the WinHTTP object with the help of the **sp_OACreate** procedure.

```
Exec sp_OAMethod @WinHttpObject, 'open', NULL, 'get',
'https://samples.openweathermap.org/data/2.5/weather?q=London,uk&appid=439d4b804bc
8187953eb36d2a8c26a02','false'
```

```
Exec sp_OAMethod @WinHttpObject, 'send'
Exec sp_OAMethod @WinHttpObject, 'responseText', @ResponseJsonText OUTPUT
Exec sp_OADestroy @WinHttpObject
```

In this part of the query **sp_OAMethod** procedure opened an HTTP connection and sent an HTTP request to the server. ResponseText method retrieved the response of the web server as a text. As a last, the **sp_OADestroy** procedure destroyed the created instance of the object.

```
IF ISJSON(@ResponseJsonText)=1
```

Before parsing the HTTP server result, we checked that the response text is a valid JSON format. The ISJSON function controls whether an expression is valid JSON text. It returns 1 if the input expression is a valid JSON otherwise it returns 0.

In the final part of the query, we used the OPENJSON function to parse the returned JSON text. The city name is stored in the **name** attribute so we defined a schema to obtain this value. The second schema parsed the weather array and got the **description** value from this array.

```
BEGIN
SELECT City, Weather FROM OPENJSON(@ResponseJsonText)
WITH(City VARCHAR(100) '$.name')
CROSS APPLY
OPENJSON(@ResponseJsonText,'$.weather') WITH(Weather  VARCHAR(100) '$.description'
)
END
```

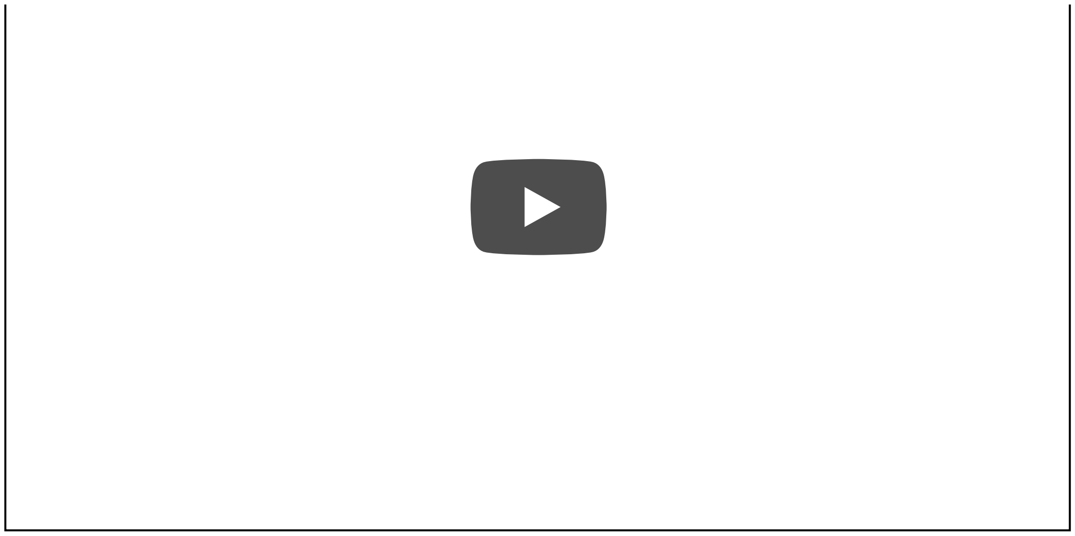
Conclusion

In this article, we explored the JSON notion and learned the data structure of this object. We also explored parse JSON in SQL Server through the OPENJSON function. JSON is used by modern applications to exchange data and store unstructured data. On the other hand, relational databases are an indispensable part of our software development life cycle. .JSON is not an alternative for the relational database model but it has a schema-less data structure that makes it more flexible to store unstructured data and combining this data model to relational data called a hybrid data model. All these features help in storing and processing both relational and JSON data by using T-SQL and this hybrid model opens a gate of the new data age.

See more

To boost SQL coding productivity, check out these SQL tools for SSMS and Visual Studio including T-SQL formatting, refactoring, auto-complete, text and data search, snippets and auto-replacements, SQL code and object comparison, multi-db script comparison, object decryption and more

An introduction to ApexSQL Complete



Esat Erkec

Esat Erkec is a SQL Server professional who began his career 8+ years ago as a Software Developer. He is a SQL Server Microsoft Certified Solutions Expert.

Most of his career has been focused on SQL Server Database Administration and Development. His current interests are in database administration and Business Intelligence. You can find him on LinkedIn.

View all posts by Esat Erkec

Related Posts:

1. **SQL Server JSON functions: a bridge between NoSQL and relational worlds**
2. **Import JSON data into SQL Server**
3. **How to import/export JSON data using SQL Server 2016**
4. **The JSON_QUERY() function to extract objects from JSON Data**
5. **Native JSON Support in SQL Server 2016**

Functions, JSON

28,821 Views