<https://docs.microsoft.com/en-us/sql/t-sql/functions/openjson-transact-sql>

# OPENJSON (Transact-SQL)

2016-6-10 10 min to read Contributors

* [Douglas Laudenschlager](https://github.com/douglaslMS)

* [Jovan Popovic](https://github.com/JocaPC)

* [Craig Guyer](https://github.com/craigg-msft)

**THIS TOPIC APPLIES TO:** yesSQL Server (starting with 2016)yesAzure SQL DatabasenoAzure SQL Data Warehouse noParallel Data Warehouse

**OPENJSON** is a table-value function that parses JSON text and returns objects and properties from the input JSON parameter as rows and columns. **OPENJSON** provides a rowset view over a JSON document, with the ability to explicitly specify the columns in the rowset and the property paths to use to populate the columns. Since **OPENJSON** returns a set of rows, you can use **OPENJSON** function in FROM clause of Transact-SQL statements like any other table, view, or table-value function.

##### Note

The **OPENJSON** function is available only under compatibility level 130 (or higher). If your database compatibility level is lower than 130, SQL Server will not be able to find and execute OPENJSON function. Other JSON functions are available at all compatibility levels. You can check compatibility level in sys.databases view or in database properties. You can change a compatibility level of database using the following command:  
ALTER DATABASE DatabaseName SET COMPATIBILITY\_LEVEL = 130

Note that compatibility level 120 might be default even in new Azure SQL Databases.

Use OPENJSON to import JSON data into SQL Server, or to convert JSON data to relational format for an app or service that can't consume JSON directly.

Topic link icon[Transact-SQL Syntax Conventions](https://docs.microsoft.com/en-us/sql/t-sql/language-elements/transact-sql-syntax-conventions-transact-sql)

## Syntax

Copy

OPENJSON( jsonExpression [ , path ] ) [ <with\_clause> ]

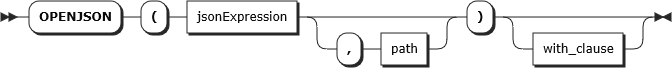
<with\_clause> ::= WITH ( { colName type [ column\_path ] [ AS JSON ] } [ ,...n ] )

##### Note

column\_type must be NVARCHAR(MAX) if **AS JSON** option is used.

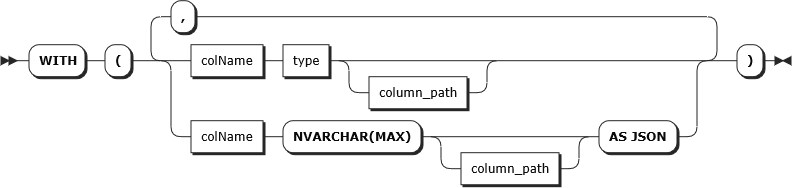
**OPENJSON** table-value function parses jsonExpression provided as a first argument and returns one or many rows containing data from JSON objects in this expression. jsonExpression might contain nested sub-objects. If user wants to parse sub-object placed on some path within jsonExpression, he can specify second **path**parameter that will define where is placed JSON sub-object that should be parsed.

openjson



By default, **OPENJSON** table-value function returns three columns with key name, value, and type of each {key:value} pair that is found in the jsonExpression. As an alternative, user can explicitly specify the schema of the result set that will return **OPENJSON** function using with\_clause:

with\_clause



with\_clause contains the list of the columns that will be returned by **OPENJSON** with their types. By default, **OPENJSON** matches keys in jsonExpression with the column names in with\_clause. If column name does not match key name, user can specify optional column\_path that represents [JSON Path Expression](https://docs.microsoft.com/en-us/sql/relational-databases/json/json-path-expressions-sql-server) that references some key within the jsonExpression.

## Arguments

### jsonExpression

Is a Unicode character expression containing the JSON text.

OPENJSON iterates over the elements of the array or the properties of the object in the JSON expression and returns one row for each element or property. The following example returns each property of an object provided as jsonExpression:

tsqlCopy

DECLARE @json NVARCHAR(4000) = N'{

"StringValue":"John",

"IntValue":45,

"TrueValue":true,

"FalseValue":false,

"NullValue":null,

"ArrayValue":["a","r","r","a","y"],

"ObjectValue":{"obj":"ect"}

}'

SELECT \*

FROM OPENJSON(@json)

**Results**

| key | value | type |
| --- | --- | --- |
| StringValue | John | 1 |
| IntValue | Doe | 2 |
| TrueValue | true | 3 |
| FalseValue | false | 3 |
| NullValue | NULL | 0 |
| ArrayValue | ["a","r","r","a","y"] | 4 |
| ObjectValue | {"obj":"ect"} | 5 |

### path

Is a JSON path expression that references an object or an array within jsonExpression. **OPENJSON** will seek into JSON text at the specified position and parse only referenced fragment. For more info, see [JSON Path Expressions (SQL Server)](https://docs.microsoft.com/en-us/sql/relational-databases/json/json-path-expressions-sql-server).

The following example returns a nested object by specifying the path:

tsqlCopy

DECLARE @json NVARCHAR(4000) = N'{

"path": {

"to":{

"sub-object":["en-GB", "en-UK","de-AT","es-AR","sr-Cyrl"]

}

}

}';

SELECT [key],value

FROM OPENJSON(@json,'$.path.to."sub-object"')

**Results**

| Key | Value |
| --- | --- |
| 0 | en-GB |
| 1 | en-UK |
| 2 | de-AT |
| 3 | es-AR |
| 4 | sr-Cyrl |

**OPENJSON** returns indexes of the elements in JSON as keys if JSON array is parsed.

In SQL Server 2017 and in Azure SQL Database, you can provide a variable as the value of path.

The comparison used to match path steps with the properties of the JSON expression is case-sensitive and collation-unaware (that is, a BIN2 comparison).

### with\_clause

Explicitly defines output schema that will be returned by **OPENJSON** function. In the with\_clause can be used following elements:

colName  
Is the name for the output column.

By default, OPENJSON uses the name of the column to match a property in the JSON text. For example, if you specify the column name in the schema, OPENJSON tries to populate this column with the property "name" in the JSON text.

You can override this default mapping by using the column\_path argument.

type  
Is the data type for the output column.

column\_path  
Is the JSON path that specifies the property to return in the specified column. For more info, see the description of the path parameter previously in this topic.

Use column\_path to override default mapping rules if the name of an output column doesn't match the name of the property.

For more info, see [JSON Path Expressions (SQL Server)](https://docs.microsoft.com/en-us/sql/relational-databases/json/json-path-expressions-sql-server).

The comparison used to match path steps with the properties of the JSON expression is case-sensitive and collation-unaware (that is, a BIN2 comparison).

AS JSON  
Use AS JSON option in column definition to specify that referenced property contains inner object or array. If you don't specify AS JSON for a column, the function returns a scalar value (for example, int, string, true, false) from the specified JSON property on the specified path. If the path represents an object or an array, the function returns null in lax mode or an error in strict mode indicating that the property can't be found at the specified path.  
This behavior is similar to the behavior of the JSON\_VALUE function.

If you specify AS JSON for a column, the function returns a JSON fragment from the specified JSON property on the specified path. If the path represents a scalar value, the function returns null in lax mode or an error in strict mode indicating that the property can't be found at the specified path. This behavior is similar to the behavior of the JSON\_QUERY function.

##### Note

If you want to return nested JSON fragment from some JSON property, you MUST specify **AS JSON** flag. Without this option, OPENJSON will return NULL value instead of referenced JSON object or array, or it will return run-time error in strict mode (property cannot be found).

If you specify AS JSON option, the type of the column must be NVARCHAR(MAX).

For example, the following query returns and formats the elements of an array.

tsqlCopy

DECLARE @json NVARCHAR(MAX) = N'[

{

"Order": {

"Number":"SO43659",

"Date":"2011-05-31T00:00:00"

},

"AccountNumber":"AW29825",

"Item": {

"Price":2024.9940,

"Quantity":1

}

},

{

"Order": {

"Number":"SO43661",

"Date":"2011-06-01T00:00:00"

},

"AccountNumber":"AW73565",

"Item": {

"Price":2024.9940,

"Quantity":3

}

}

]'

SELECT \*

FROM OPENJSON ( @json )

WITH (

Number varchar(200) '$.Order.Number',

Date datetime '$.Order.Date',

Customer varchar(200) '$.AccountNumber',

Quantity int '$.Item.Quantity',

[Order] nvarchar(MAX) AS JSON

)

**Results**

| Number | Date | Customer | Quantity | Order |
| --- | --- | --- | --- | --- |
| SO43659 | 2011-05-31T00:00:00 | AW29825 | 1 | {"Number":"SO43659","Date":"2011-05-31T00:00:00"} |
| SO43661 | 2011-06-01T00:00:00 | AW73565 | 3 | {"Number":"SO43661", "Date":"2011-06-01T00:00:00"} |

## Return Value

Columns that will be returned as a result of OPENJSON function depend on WITH option.

1. When you call OPENJSON with the default schema - that is, when you don't specify an explicit schema in the WITH clause - the function returns a table with the following columns.
   1. **Key**. An nvarchar(4000) value that contains the name of the specified property or the index of the element in the specified array. The key column has a BIN2 collation.
   2. **Value**. An nvarchar(max) value that contains the value of the property. The value column inherits its collation from jsonExpression.
   3. **Type**. An int value that contains the type of the value. The **Type** column is returned only when you use OPENJSON with the default schema. The type column has one of the following values.

| Value of the Type column | JSON data type |
| --- | --- |
| 0 | null |
| 1 | string |
| 2 | int |
| 3 | true/false |
| 4 | array |
| 5 | object |

* 1. Only first level properties are returned. The statement fails if the JSON text is not properly formatted.

1. When you call OPENJSON and you specify an explicit schema in the WITH clause, the function returns a table with the schema that you defined in the WITH clause.

## Remarks

json\_path used in the second argument of **OPENJSON** or in with\_clause might start with **lax** or **strict** keyword. In **lax** mode **OPENJSON** will not raise any error if object or value on the specified path is not found. **OPENJSON**will returne either empty result set or NULL value if targeted object cannot be found. In **strict** mode error will be returned if referenced path cannot be found. If you don't specify mode, OPENJSON parses the root object using lax path mode (that is, as if you had specified the **lax** option in the path expression).

Some of the examples on this page explicitly specify the path mode, lax or strict. This is optional. If you don't explicitly specify a path mode, lax mode is the default. For more info about path mode and path expressions, see [JSON Path Expressions (SQL Server)](https://docs.microsoft.com/en-us/sql/relational-databases/json/json-path-expressions-sql-server).

Column names in **with\_clause** are matched with the keys in JSON text. If you specify column name [Address.Country] it will be matched with the key "Address.Country". If you need to reference nested key "Country" within the object "Address", you would need to specify the path "$.Address.Country" in column path.

json\_path may contain keys with alphanumeric characters. Escape key name in json\_path with double quotes if you have some special characters in the keys. As an example, '$."my key $1".regularKey."key with . dot" would match value 1 in the following JSON text:

Copy

{

"my key $1": {

"regularKey":{

"key with . dot": 1

}

}

}

## Examples

### Example 1 - Convert a JSON array value to a temporary table

In this example, list of identifiers are provided as JSON array of numbers. Following query converts JSON array to table of identifiers and filters all products with specified ids.

tsqlCopy

DECLARE @pSearchOptions NVARCHAR(4000) =N'[1,2,3,4]'

SELECT \*

FROM products

INNER JOIN OPENJSON(@pSearchOptions) AS productTypes

ON product.productTypeID=productTypes.value

This query is equivalent to the following example. However, in example below you would need to embed numbers on client side instead of passing them as parameters.

tsqlCopy

SELECT \*

FROM products

WHERE product.productTypeID IN(1,2,3,4)

### Example 2 - Merge properties from two JSON objects

The following example selects a union of all the properties of two JSON objects. The two objects have a duplicate "name" property. The example uses the key value to exclude the duplicate row from the results.

tsqlCopy

DECLARE @json1 NVARCHAR(MAX),@json2 NVARCHAR(MAX)

SET @json1=N'{"name": "John", "surname":"Doe"}'

SET @json2=N'{"name": "John", "age":45}'

SELECT \*

FROM OPENJSON(@json1)

UNION ALL

SELECT \*

FROM OPENJSON(@json2)

WHERE [key] NOT IN(SELECT [key] FROM OPENJSON(@json1))

### Example 3 - Join rows with JSON data stored in table cells using CROSS APPLY

In the following example, the SalesOrderHeader table has a SalesReason text column that contains an array of SalesOrderReasons in JSON format. The SalesOrderReasons objects contain properties like "Quality" and "Manufacturer". The example creates a report that joins every sales order row to the related sales reasons. The OPENJSON operator expands the JSON array of sales reasons as if the reasons were stored in a separate child table. Then the CROSS APPLY operator joins each sales order row to the rows returned by the OPENJSON table-valued function.

tsqlCopy

SELECT SalesOrderID,OrderDate,value AS Reason

FROM Sales.SalesOrderHeader

CROSS APPLY OPENJSON(SalesReasons)

##### Tip

When you have to expand JSON arrays stored in individual fields and join them with their parent rows, you typically use the Transact-SQL CROSS APPLY operator. For more info about CROSS APPLY, see [FROM (Transact-SQL)](https://docs.microsoft.com/en-us/sql/t-sql/queries/from-transact-sql).

The same query can be re-written using OPENJSON with explicitly defined return schema:

tsqlCopy

SELECT SalesOrderID, OrderDate, value AS Reason

FROM Sales.SalesOrderHeader

CROSS APPLY OPENJSON (SalesReasons) WITH (value nvarchar(100) '$')

In this example, '$' path references each element in arrays. You can use this type of query if you want to explicitly cast returned value.

### Example 4 - Combine relational rows and JSON elements with CROSS APPLY

The following query returns the results shown in the following table.

tsqlCopy

SELECT store.title, location.street, location.lat, location.long

FROM store

CROSS APPLY OPENJSON(store.jsonCol, 'lax $.location')

WITH (street varchar(500) , postcode varchar(500) '$.postcode' ,

lon int '$.geo.longitude', lat int '$.geo.latitude')

AS location

**Results**

| title | street | postcode | lon | lat |
| --- | --- | --- | --- | --- |
| Whole Food Markets | 17991 Redmond Way | WA 98052 | 47.666124 | -122.10155 |
| Sears | 148th Ave NE | WA 98052 | 47.63024 | -122.141246,17 |

### Example 5 - Import JSON data into SQL Server

The following example loads an entire JSON object into a SQL Server table.

tsqlCopy

DECLARE @json NVARCHAR(max) = N'{

"id" : 2,

"firstName": "John",

"lastName": "Smith",

"isAlive": true,

"age": 25,

"dateOfBirth": "2015-03-25T12:00:00",

"spouse": null

}';

INSERT INTO Person

SELECT \*

FROM OPENJSON(@json)

WITH (id int,

firstName nvarchar(50), lastName nvarchar(50),

isAlive bit, age int,

dateOfBirth datetime2, spouse nvarchar(50)

## See Also

[JSON Path Expressions (SQL Server)](https://docs.microsoft.com/en-us/sql/relational-databases/json/json-path-expressions-sql-server)  
[Convert JSON Data to Rows and Columns with OPENJSON (SQL Server)](https://docs.microsoft.com/en-us/sql/relational-databases/json/convert-json-data-to-rows-and-columns-with-openjson-sql-server)  
[Use OPENJSON with the Default Schema (SQL Server)](https://docs.microsoft.com/en-us/sql/relational-databases/json/use-openjson-with-the-default-schema-sql-server)  
[Use OPENJSON with an Explicit Schema (SQL Server)](https://docs.microsoft.com/en-us/sql/relational-databases/json/use-openjson-with-an-explicit-schema-sql-server)

**4 Comments**

Sign in

**22 people listening**

+ Follow

Post comment as...

[Kev_K](https://docs.microsoft.com/en-us/sql/t-sql/functions/openjson-transact-sql)

Kev\_K May 8, 2017

I ran into an issue trying to parse a nested JSON object that is a number and not a letter string. Here is what I am trying to parse:

Day": { "Hour": { "11": 5625.0751953125, "10": 1369.0666503906, "9": 20627.30078125, "8":2088.8942871094}select\*from openjson(@json)WITH(day nvarchar(100)'$.Day', hour nvarchar(100)'$.Day.Hour.11')

Error:

Msg 13607, Level 16, State 4, Line 23  
JSON path is not properly formatted. Unexpected character '1' is found at position 22.

There has to be a way to parse objects that begin with number. Any help will be appreciated. Thank you.

LikeReply

[Kev_K](https://docs.microsoft.com/en-us/sql/t-sql/functions/openjson-transact-sql)

Kev\_K May 8, 2017

I found it! Thanks!

Reference keys that contain non-alphanumeric characters in JSON text

Question. I have non-alphanumeric characters in keys in my JSON text. How can I reference these properties?[**+**](https://docs.microsoft.com/en-us/sql/t-sql/functions/openjson-transact-sql)

Answer. You have to surround them with quotes in JSON paths. For example,

JSON\_VALUE(@json, '$."$info"."First Name".value')

[**https://docs.microsoft.com/en-us/sql/relational-databases/json/solve-common-issues-with-json-in-sql-server**](https://docs.microsoft.com/en-us/sql/relational-databases/json/solve-common-issues-with-json-in-sql-server)

LikeReply

[Weitao_Ren](https://docs.microsoft.com/en-us/sql/t-sql/functions/openjson-transact-sql)

Weitao\_Ren Mar 25, 2017

On SQL Server Version 13.0.4202.2:

The following code does't work for [Address.Country], it return null:

DECLARE @json\_string NVARCHAR(MAX) =

N'{"Customers":

    [{"Id":1,"Name":"Basavaraj",

            "Address":{"State":"KA","Country":"India"}},

     {"Id":2,"Name":"Kalpana",

            "Address":{"State":"NY","Country":"United State"}}

    ]

 }'

SELECT \*

FROM OPENJSON(@json\_string,'$.Customers')

WITH(Id INT, Name Varchar(100), [Address.Country] NVarchar(50) )

you can only do it as followed:

DECLARE @json\_string NVARCHAR(MAX) =

 N'{"Customers":

    [{"Id":1,"Name":"Basavaraj",

            "Address":{"State":"KA","Country":"India"}},

     {"Id":2,"Name":"Kalpana",

            "Address":{"State":"NY","Country":"United State"}}

    ]

 }'

SELECT \*

FROM OPENJSON(@json\_string,'$.Customers')

WITH(Id INT, Name Varchar(100), [Address.Country] NVarchar(50) '$.Address.Country')

LikeReply

[Jovan_Popovic](https://docs.microsoft.com/en-us/sql/t-sql/functions/openjson-transact-sql)

Jovan\_Popovic Apr 7, 2017

Hi @Weitao\_Ren we have clarified this in the latest version of doc:

Column names in **with\_clause** are matched with the keys in JSON text. If you specify column name [Address.Country] it will be matched with the key "Address.Country". If you need to reference nested key "Country" within the object "Address", you would need to specify the path "$.Address.Country" in column path.

Thanks,

Jovan