How to import/export JSON data using SQL Server 2016

sqlshack.com/importexport-json-data-using-sql-server-2016

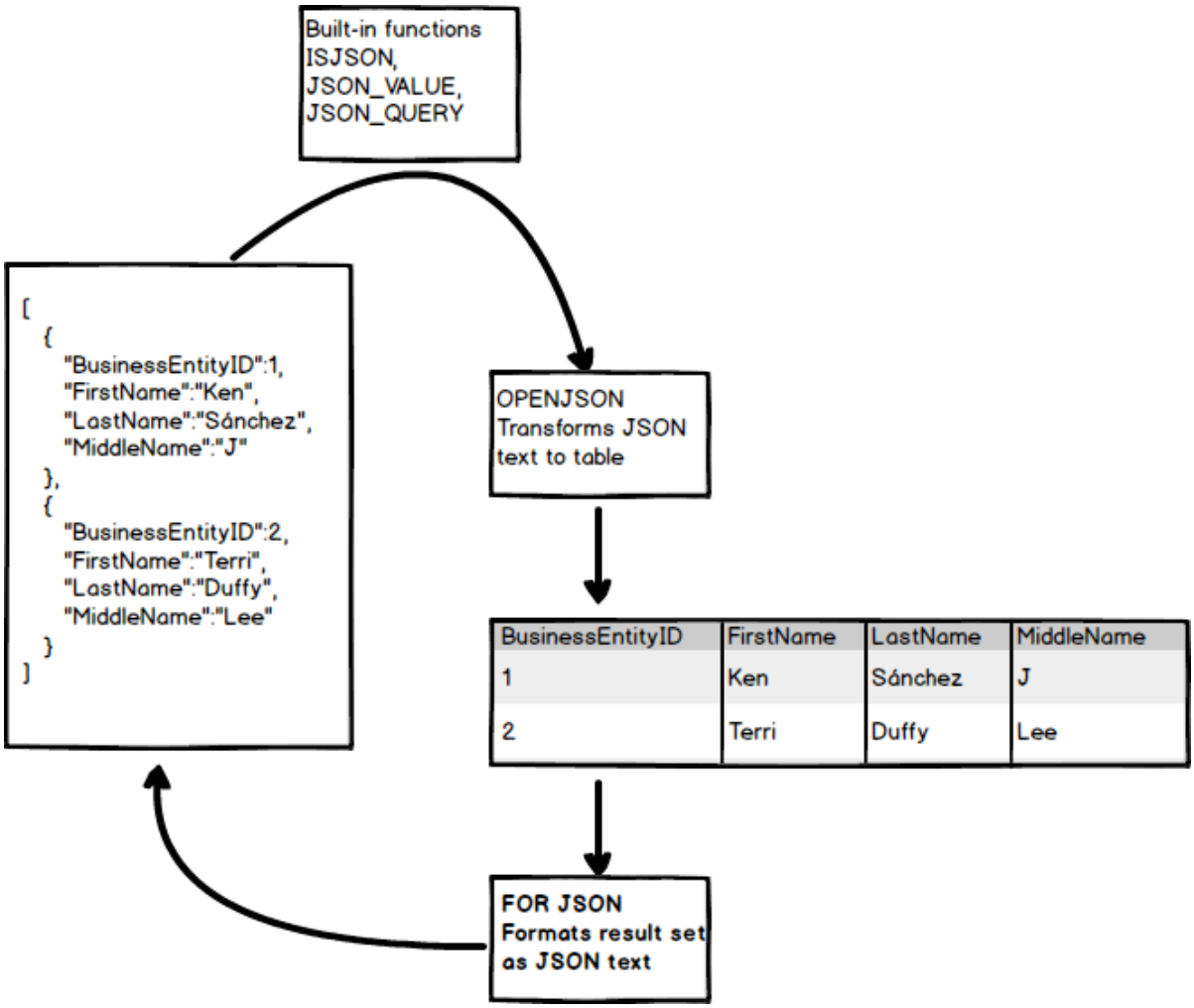
Marko
Zivkovic

March 7, 2017

JSON is an abbreviation for **JavaScript Object Notation**. JSON is very popular and currently the most commonly used data exchange format. Most modern web and mobile services return information formatted as JSON text, all database web services, web browsers (Firefox, Internet Explorer) return results formatted as JSON text or accept data formatted as JSON. Since external systems format information as JSON text, JSON is also stored in SQL Server 2016 as text. You can use standard NVARCHAR columns to store JSON data in SQL Server 2016.

This article will explain how to import JSON data into SQL Server 2016 table and how to export data from SQL Server 2016 table as JSON using SQL Server 2016 built-in functions.

With SQL Server 2016, built-in functions can parse JSON text to read or modify JSON values, transform JSON array of objects into table format, any Transact -SQL query can be run over the converted JSON objects, results of Transact-SQL queries can be formatted into JSON format.



So, let's start. Below is a simple example of JSON:

```
{
      ”BusinessEntityID”:1,
      ”NationalIDNumber”:”295847284″,
      ”JobTitle”:”Chief Executive Officer”,
      ”BirthDate”:”1969-01-29″,
      ”Gender”:”M”
}
```

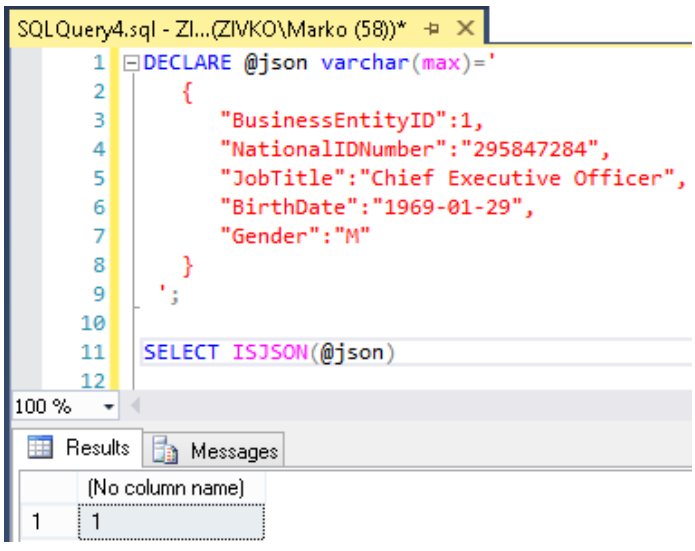
More information about structure of the JSON can be found on this link.

Let's declare a SQL Server variable and put JSON code in it.

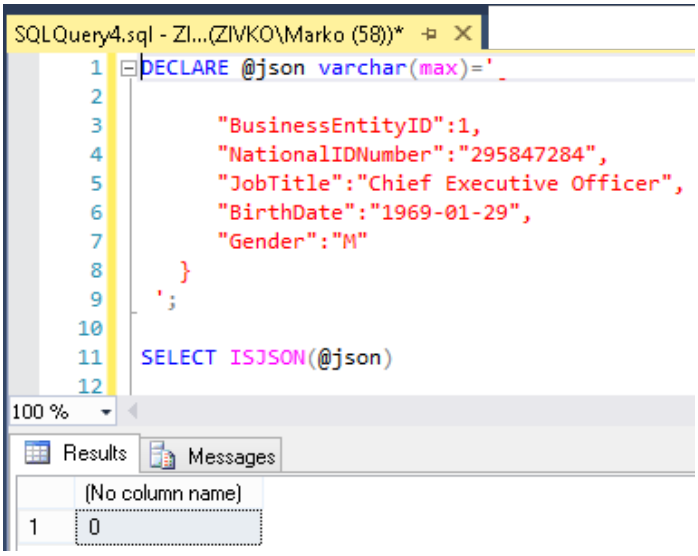
```
1
2    DECLARE @json varchar(max)='
3      {
4         "BusinessEntityID":1,
5         "NationalIDNumber":"295847284",
6         "JobTitle":"Chief Executive
7    Officer",
8         "BirthDate":"1969-01-29",
9         "Gender":"M"
10     }
11   ';
```

One of the built-in JASON functions that are implemented in SQL Server 2016 is ISJSON.

The ISJSON function verifies if it is the code in @json variable formatted as JSON. If the code in the @json variable, formats correctly the output value in the Results grid, 1 will appear:



Otherwise, the output value in the Results grid will be 0. For example, if the open curly bracket is omitted from the example above, the result will be:



To convert the JSON text into a set of rows and columns, the OPENJSON function is used.

Syntax of the OPENJSON function that transforms JSON text to row and columns looks like:

OPENJSON (<json text>)
WITH (<column/type>)

In the WITH clause, the schema of returned rows with name of columns and their types is defined. The OPENJSON function will parse JSON object, match properties in JSON object with column names in the WITH clause and convert their values to specified types.

In the example below, it is shown how to convert JSON text to set of rows and columns:

```
1
2   DECLARE @json varchar(max)='
3       {
4         "BusinessEntityID":1,
5         "NationalIDNumber":"295847284",
6         "JobTitle":"Chief Executive
7   Officer",
8         "BirthDate":"1969-01-29",
9         "Gender":"M"
10      }
11  ';
12
13  SELECT * FROM OPENJSON(@json)
14  WITH (BusinessEntityID int,
15  NationalIDNumber int,
16  JobTitle varchar(100),
17  BirthDate varchar(200),
18  Gender varchar(2)
19  )
```

The result will look like:

BusinessEntityID	NationalIDNumber	JobTitle	BirthDate	Gender
1	295847284	Chief Executive Officer	1969-01-29	M

If the SELECT statement without a WITH clause is executed:

```
1
2    DECLARE @json varchar(max)='
3       {
4         "BusinessEntityID":1,
5         "NationalIDNumber":"295847284",
6         "JobTitle":"Chief Executive
7    Officer",
8         "BirthDate":"1969-01-29",
9         "Gender":"M"
10     }
11   ';
12
13   SELECT * FROM OPENJSON(@json)
```

The following result will appear:

key	value	type
BusinessEntityID	1	2
NationalIDNumber	295847284	1
JobTitle	Chief Executive Officer	1
BirthDate	1969-01-29	1
Gender	M	1

1. key column contains the name of the property
2. value column contains the value of the property
3. type column contains the data type of the value

The type column has six values for the data types:

value	data type
0	null
1	string
2	int
3	true/false
4	array
5	object

To extract a scalar value from the JSON text and use it in the Transact-SQL queries, use the JSON_VALUEfunction. Let's access to a value of the BusinessEntityID property from the @jason variable and use it in the WHERE clause to return some values from the Person.Person table in the AdventureWorks 2014 database. Paste and execute the following code:

```
1
2    USE AdventureWorks2014
3    DECLARE @json varchar(max)='
4       {
5         "BusinessEntityID":1,
6         "NationalIDNumber":"295847284",
7         "JobTitle":"Chief Executive Officer",
8         "BirthDate":"1969-01-29",
9         "Gender":"M"
10      }
11   ';
12
13   SELECT FirstName, LastName FROM Person.Person
14   WHERE  BusinessEntityID = JSON_VALUE(@json, '$.BusinessEntityID')
15
```

The results will look like:

FirstName	LastName
Ken	Sánchez

A Dollar sign $ is used to reference (access) of the properties, objects in JSON text. If it is omitted from the query:

```
1
2    USE AdventureWorks2014
3    DECLARE @json varchar(max)='
4       {
5         "BusinessEntityID":1,
6         "NationalIDNumber":"295847284",
7         "JobTitle":"Chief Executive Officer",
8         "BirthDate":"1969-01-29",
9         "Gender":"M"
10      }
11   ';
12
13   SELECT FirstName, LastName FROM Person.Person
14   WHERE  BusinessEntityID = JSON_VALUE(@json, 'BusinessEntityID')
15
```

The following error may occur:

Msg 13607, Level 16, State 3, Line 14
JSON path is not properly formatted. Unexpected character ‘B’ is found at position 0.

To extract an array or an object from a JSON text use the JSON_QUERY function. Let’s execute the query that contain JSON_QUERY function:

```
1
2    DECLARE @json varchar(max)='
3        {
4          "BusinessEntityID":1,
5          "NationalIDNumber":"295847284",
6          "JobTitle":"Chief Executive Officer",
7          "BirthDate":"1969-01-29",
8          "Gender":"M"
9        }
10   ';
11
12   SELECT JSON_QUERY(@json, '$.BusinessEntityID')
13
```

The result will be:

```
(No column
name)
-----------
NULL
```

The NULL value is returned because the JSON_QUERY function works with arrays and objects not with scalar values. To see the error message instead of the NULL value, type the word strict before dollar sign:

```
1
2    DECLARE @json varchar(max)='
3        {
4          "BusinessEntityID":1,
5          "NationalIDNumber":"295847284",
6          "JobTitle":"Chief Executive Officer",
7          "BirthDate":"1969-01-29",
8          "Gender":"M"
9        }
10   ';
11
12   SELECT JSON_QUERY(@json, 'strict
13   $.BusinessEntityID')
```

When the code above is executed, the following error message will appear:

Msg 13624, Level 16, State 1, Line 12
Object or array cannot be found in the specified JSON path.

Let's add the Contact object in the @json variable and use the JSON_QUERY function:

```
1
2   DECLARE @json varchar(max)='
3      {
4        "BusinessEntityID":1,
5        "NationalIDNumber":"295847284",
6        "JobTitle":"Chief Executive Officer",
7        "BirthDate":"1969-01-29",
8        "Gender":"M",
9        "Contact":{"Home":"036/222-333","Mob":"064/3376222"}
10     }
11  ';
12
13  SELECT JSON_QUERY(@json, '$.Contact')
14
```

The following results will appear:

(No column name)
{“Home”:”036/222-333″,”Mob”:”064/3376222″}

Storing JSON Data in SQL Server 2016

Inserting data into some SQL Server table using data from @json is the same as regular T-SQL. Execute the following code:

```
1
2   DECLARE @json varchar(max)='
3      {
4        "BusinessEntityID":1,
5        "NationalIDNumber":"295847284",
6        "JobTitle":"Chief Executive Officer",
7        "BirthDate":"1969-01-29",
8        "Gender":"M",
9        "Contact":{"Home":"036/222-333","Mob":"064/3376222"}
10     }
11  ';
12
13  SELECT * INTO Person
14  FROM OPENJSON(@json)
15  WITH (BusinessEntityID int,
16        NationalIDNumber int,
17        JobTitle varchar(100),
18        BirthDate varchar(200),
19        Gender varchar(2),
20        Contact varchar(max)
21  )
22
```

The following results will appear:

BusinessEntityID	NationalIDNumber	JobTitle	BirthDate	Gender	Contact
1	295847284	Chief Executive Officer	1969-01-29	M	NULL

As you can see, the Contact column in the Person table have NULL value instead of {“Home”:”036/222-333″,”Mob”:”064/3376222″}.

To insert values from the Contact object in the @json variable into the Contact column under the Person table, the AS JSON clause must be used. Let’s put this clause into a code and execute:

```
1
2   DECLARE @json varchar(max)='
3       {
4         "BusinessEntityID":1,
5         "NationalIDNumber":"295847284",
6         "JobTitle":"Chief Executive Officer",
7         "BirthDate":"1969-01-29",
8         "Gender":"M",
9         "Contact":{"Home":"036/222-333","Mob":"064/3376222"}
10      }
11  ';
12
13  SELECT * INTO Person
14  FROM OPENJSON(@json)
15  WITH (BusinessEntityID int,
16          NationalIDNumber int,
17          JobTitle varchar(100),
18          BirthDate varchar(200),
19          Gender varchar(2),
20          Contact varchar(max) AS JSON
21  )
22
```

But this time an error message will appear:

Msg 13618, Level 16, State 1, Line 30
AS JSON option can be specified only for column of nvarchar(max) type in WITH clause.

As the message says AS JASON option supports only nvarchar(max) data type. Let’s change data type for **Contact** column and execute the query again. After changing the data type of the **Contact** column from **varchar(max)** to **nvarchar(max)** and executing it, the following results will appear:

BusinessEntityID	NationalIDNumber	JobTitle	BirthDate	Gender	Contact
1	295847284	Chief Executive Officer	1969-01-29	M	{“Home”:”036/222-333″,”Mob”:”064/3376222″}

Exporting SQL Server 2016 data as JSON

To format/export query results as JSON, use the FOR JSON clause with the PATH or AUTO mode. When export query results to JSON, one of the mode must be used with the FOR JSON clause, otherwise the following error will occur:

Msg 102, Level 15, State 1, Line 7
Incorrect syntax near ‘JSON’.

The main difference between the PATH and AUTO mode is that, with the PATH mode, a user has a full control over the way of how to format the JSON output while with the AUTO mode the FOR JSON clause will automatically format the JSON output based on the structure of the SELECT statement.

PATH mode

Let's use a simple example to demonstrate what the PATH mode with FOR JSON clause can do. In this example the Pesron.Person table is used from the AdventureWorks 2014 database. In a query editor, the following code should be pasted and executed:

```
1
2    SELECT TOP 3
3          BusinessEntityID,
4          FirstName,
5          LastName,
6          Title,
7          MiddleName
8       FROM Person.Person
9       FOR JSON PATH
10
```

The JSON output will be:

```
[
    {
        "BusinessEntityID":1,
        "FirstName":"Ken",
        "LastName":"Sánchez",
        "MiddleName":"J"
    },
     {
        "BusinessEntityID":2,
        "FirstName":"Terri",
        "LastName":"Duffy",
        "MiddleName":"Lee"
     },
     {
        "BusinessEntityID":3,
        "FirstName":"Roberto",
        "LastName":"Tamburello"
    }
]
```

If you notice, the "Title" and in some sections the "MiddleName"properties don't appear in the JSON output. This is because the "MiddleName" and "Title" contain null values. By default, null values are not included in the JSON output. In order to included null values from the query results into the JSON output use the INCLUDE_NULL_VALUESoption**.**

Let's include INCLUDE_NULL_VALUESoption in an example and execute the query:

```
1
2    SELECT TOP 3
3          BusinessEntityID,
4          FirstName,
5          LastName,
6          Title,
7          MiddleName
8       FROM Person.Person
9       FOR JSON PATH, INCLUDE_NULL_VALUES
10
```

The result will be:

```
[
      {
            “BusinessEntityID”:1,
            “FirstName”:”Ken”,
            “LastName”:”Sánchez”,
            “Title”:null,
            “MiddleName”:”J”
      },
      {
            “BusinessEntityID”:2,
            “FirstName”:”Terri”,
            “LastName”:”Duffy”,
            “Title”:null,
            “MiddleName”:”Lee”
      },
      {
            “BusinessEntityID”:3,
            “FirstName”:”Roberto”,
            “LastName”:”Tamburello”,
            “Title”:null,
            “MiddleName”:null
     }
]
```

With the PATH mode, the dot syntax can be used, for example ‘Item.Title’ to format nested JSON output. For example, let’s add aliases for the Title and MiddleName columns:

```
1
2    SELECT TOP 3
3          BusinessEntityID,
4          FirstName,
5          LastName,
6          Title AS 'Item.Title',
7          MiddleName AS 'Item.MiddleName'
8       FROM Person.Person
9       FOR JSON PATH, INCLUDE_NULL_VALUES
10
```

The JSON output will be:

```
[
	{
		“BusinessEntityID”:1,
		“FirstName”:”Ken”,
		“LastName”:”Sánchez”,
		“Item”:{
			“Title”:null,
			“MiddleName “:”J”
		}
	},
	{
		“BusinessEntityID”:2,
		“FirstName”:”Terri”,
		“LastName”:”Duffy”,
		“Item”:{
			“Title”:null,
			“MiddleName “:”Lee”
		}
	},
	{
		“BusinessEntityID”:3,
		“FirstName”:”Roberto”,
		“LastName”:”Tamburello”,
		“Item”:{
			“Title”:null,
			“MiddleName”:null
		}
	}
]
```

As you can see, the JSON output now contains the “Item” object and the “Title” and “MiddleName” properties inside it.

AUTO mode

AUTO mode will automatically generate the JSON output based on the order of columns in the SELECT statement.

For example, if we use the previously example and instead of the PATH put the AUTO mode after FOR JSON clause

```
1
2    SELECT TOP 3
3          BusinessEntityID,
4          FirstName,
5          LastName,
6          Title AS 'Item.Title',
7          MiddleName AS 'Item.MiddleName'
8       FROM Person.Person
9       FOR JSON AUTO, INCLUDE_NULL_VALUES
10
```

The result will be:

```
[
    {
        “BusinessEntityID”:1,
        “FirstName”:”Ken”,
        “LastName”:”Sánchez”,
        “Item.Title”:null,
        “Item.MiddleName”:”J”
    },
    {
        “BusinessEntityID”:2,
        “FirstName”:”Terri”,
        “LastName”:”Duffy”,
        “Item.Title”:null,
        “Item.MiddleName”:”Lee”
    },
    {
        “BusinessEntityID”:3,
        “FirstName”:”Roberto”,
        “LastName”:”Tamburello”,
        “Item.Title”:null,
        “Item.MiddleName”:null
    }
]
```

AUTO mode in this case when is used will have the results from one table which will not create the nested JSON output and the dot separator will be treated as the key with dots. But when two tables are joined, the columns from the first table will be treated as the properties of the root object and the columns from the second table will be treated as the properties of a nested object. A table name or alias of the second table will be used as a name of the nested array:

When the following query is executed:

```
1
2  SELECT st.TerritoryID, st.Name AS Territory,  s.Name  FROM Sales.SalesTerritory st
3  INNER JOIN Sales.Customer c ON st.TerritoryID = c.TerritoryID
4  INNER JOIN Sales.Store s ON c.StoreID = s.BusinessEntityID
5  WHERE st.TerritoryID=2 AND s.Name LIKE 'W%'
6  FOR JSON AUTO
7
```

The result will be:

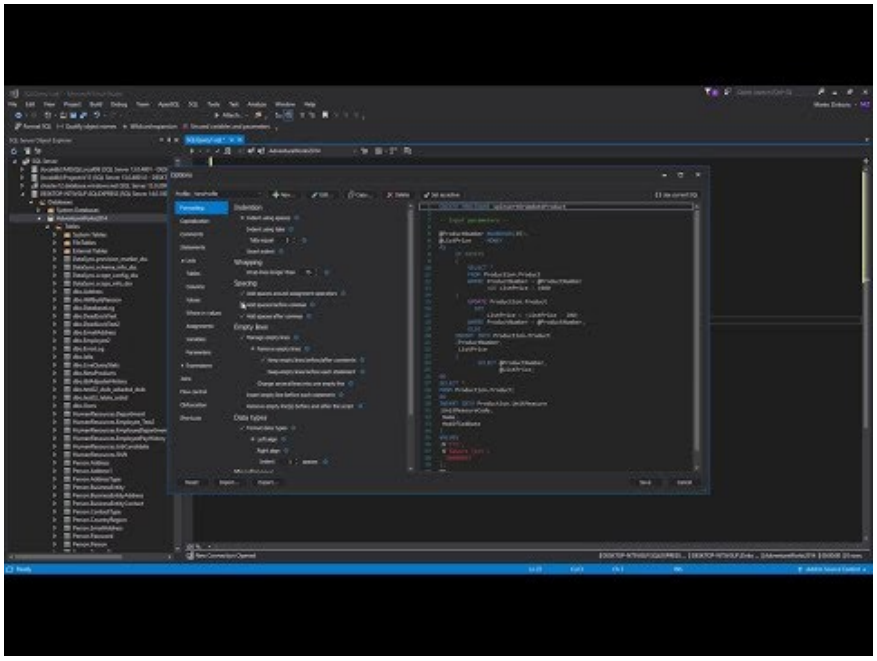
```
[
    {
        “TerritoryID”:2,
        “Territory”:”Northeast”,
        “s”:[
            {
                “Name”:”Wholesale Bikes”
            },
            {
                “Name”:”Wheelsets Storehouse”
```

```
            },
            {
                "Name":"Weekend Tours"
            },
            {
                "Name":"Wholesale Bikes"
            },
            {
                "Name":"Weekend Tours"
            },
            {
                "Name":"Wheelsets Storehouse"
            }
        ]
    }
]
```

See also:

See more

To boost SQL coding productivity, check out these free 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



Watch Video At: https://youtu.be/_lH7yihq6TA

Marko Zivkovic

Marko is a Mechanical engineer, who likes to play basketball, foosball (table-soccer) and listen to rock music. He is interested in SQL code, PHP development, HTML and CSS techniques.

Currently working for ApexSQL LLC as a Software Sales Engineer, he is helping customers with any technical issues and does quality assurance for ApexSQL Complete, ApexSQL Refactor and ApexSQL Search free add-ins.

View all posts by Marko Zivkovic