# When there is no JSON …

Beata Zalewa

zalnet

/bzalewa

## About

In professional life consultant, Microsoft Certified Trainer, administrator, developer, freelancer, sometimes a miracle-worker and magician.

From the beginning of career associated with Microsoft technologies.

Private life in numbers: 1 husband, 1 daughter, 1 cat and 2 dogs. My hobbies are detective stories and photography.

Email: info@zalnet.pl

Skype: beata.zalewa

blog

WWW: https://blog.zalnet.pl

zalnet

/bzalewa

# Agenda

- ✓ ISJSON
- ✓ JSON_VALUE
- ✓ JSON_QUERY
- ✓ JSON_MODIFY
- ✓ OPENJSON
- ✓ FOR JSON
- ✓ Indexing Strategy for JSON Value in SQL Server 2017
- ✓ Apps with JSON

zalnet

/bzalewa

## Short history

Before SQL Server 2016, there were many other databases, which already had the support for JSON:

- MongoDB, Oracle Database, Postgres SQL
- CouchDB, eXistDB, Elasticsearch, BaseX, MarkLogic, OrientDB, Riak

zalnet

/bzalewa

## JSON functions in SQL Server 2017

- JSON functions have been introduced with SQL 2016 in order to support JSON natively in SQL Server 2016. These functions are:

  **- ISJSON**

  **- JSON_VALUE**

  **- JSON_QUERY**

  **- JSON_MODIFY**

  **- OPENJSON**

  **- FOR JSON**

- **Unlike XML, in SQL Server there's no specific data type to accommodate JSON. We need to use NVARCHAR(MAX) type.**

zalnet

## Declare a variable and assign a JSON string to it

- It's simple as assigning a string value to a NVARCHAR type variable.

```
DECLARE @JSONText AS NVARCHAR(4000) = N'{"SpeakerInfo": {
"FirstName": "Beata",
"LastName": "Zalewa",
"DateOfBirth": "25-Sep-1976",
"MonthSalary": 1500
}}'
```

## ISJSON Function

- This is the simplest of the functions for JSON support in SQL Server. It takes one string argument as the input, validate it and returns a BIT value:

- **1** if the provided JSON is a **valid input**

- **0** if it is an **invalid input**.

- If the provided input argument is NULL then the return value will also be NULL.

### Syntax:

**ISJSON(@input)**

**ISJSON (string_expression)**

where **string_expression** can be a table column or a string (i.e. **varchar/nvarchar**) variable or a **string** constant. And this string expression is evaluated to check whether it is a valid JSON.

/bzalewa

## ISJSON Function

- However there's a concern when it comes to validate using ISJSON. ISJSON will not validate whether the key is unique or not.
- If we will use the JSON string with duplicate key value in SQL expression, we will still get the return value as 1, even the JSON string is containing a duplicate key. Most of the JSON validators will find these kind of JSON strings as invalid.
- Page for testing JSON strings: https://jsonlint.com/

zalnet

## Demo

## ISJSON Function

DEMO

(ISJSON.sql)

/bzalewa

## JSON_VALUE Function

- This function returns the scalar value from the input JSON text from the specified JSON path location.
- The array index starts with zero.
- If the index is out of the range it will return a NULL
- The JSON path is case sensitive. Therefore it should match exactly with what you have on the JSON string. If the path is not found it will return NULL.

### Syntax:

**JSON_VALUE ( json_string, json_path )**

where **json_string** is the JSON string from which the scalar value will be extracted and **json_path** is the location of the scalar value in the json_string.

Within json_path we can specify the path mode, it can be lax or strict.

Lax is the default path mode, if json_path is invalid (i.e. it is not present in the json_string) then it returns null, but if path mode is strict it will raise an error.

zalnet

/bzalewa

## JSON_VALUE Function

DEMO

(JSON_VALUE .sql)

zalnet

## JSON_QUERY Function

- JSON_QUERY function will extract and return details as an array string from a given JSON string.
- JSON_QUERY basically returns the JSON fragment (i.e. JSON object or an array) from the input JSON string from the specified JSON path.

### Syntax:

**JSON_QUERY ( json_string, json_path )**

where **json_string** is the JSON string from which the scalar value will be extracted and **json_path** is the location of the scalar value in the json_string.

Within json_path we can specify the path mode, it can be lax or strict.

Lax is the default path mode, if json_path is invalid (i.e. it is not present in the json_string) then it returns null, but if path mode is strict it will raise an error.

/bzalewa

## JSON_QUERY Function

# DEMO

(JSON_QUERY .sql)

zalnet

## JSON_MODIFY Function

- This function is very similar to the xml.modify() functionality available in SQL Server.
- JSON_MODIFY function will be used to append an existing value on a property in a JSON string.
- Even though the name reflects an idea of modifying an existing value, it can be used in three ways:

  - To update a value

  - To delete a value

  - To insert a value

/bzalewa

## JSON_MODIFY function

- **UPDATE:**

You need to provide two things when updating.

- Exact path of the property
- The value which should be updated.

- **DELETE:**

You need to provide two things when removing.

- Exact path of the property
- The second parameter needs to be passed as NULL

- **INSERT:**

When it comes for insertion, there are two ways where a value can be inserted into a JSON string.

- Can be inserted as a Property/Value
- Can be inserted as a new array element

zalnet

/bzalewa

## JSON_MODIFY Function

DEMO

(JSON_MODIFY .sql)

zalnet

/bzalewa

## OPENJSON Function

- Unlike the other functions OPENJSON is a table valued function. Which means it will return a table or a collection of rows, rather than single value.
- This will iterate through JSON object arrays and populate a row for each element.
- This function will return the details as a result set containing the following information.

  - **Key** → Key name of the attribute
  - **Value** → Value underlying the above key
  - **Type** → Data type of the value
- This can be used in two ways.

  - Without a pre-defined schema
  - With a well-defined schema

zalnet

/bzalewa

## OPENJSON Function

# DEMO

(OPENJSON.sql)

zalnet

/bzalewa

## FOR JSON Function

- FOR JSON functionality is pretty much similar to the FOR XML functionality available in SQL Server.
- It's used to export tabular data to JSON format.
- Each row is converted to a JSON object and data on cells will be converted to values on those respective JSON objects.
- Column names/aliases will be used as key names.
- There are two ways which FOR JSON functionality can be used.

  - FOR JSON AUTO

  - FOR JSON PATH

zalnet

/bzalewa

## FOR JSON Function

# DEMO

(FORJSON.sql)

zalnet

/bzalewa

## Indexing Strategy for JSON Value in Sql Server 2017

- If we are storing JSON data in a table column, then we may come across a scenario where we may need to retrieve only the records with specific JSON property value.
- Creating an index on the JSON column isn’t the correct approach.
- It indexes the complete JSON value like any other value in a VARCHAR/NVARCHAR column and we are looking for particular JSON Property value which is not at the beginning of the JSON string.
- It also takes a lot of additional storage space as the complete JSON value is indexed. So, creating such indexes is bad idea.
- But for JSON we have an alternative way of Indexing JSON Property.

zalnet

/bzalewa

Indexing Strategy for JSON Value in Sql Server 2017

# DEMO

(Indexes.sql)

zalnet

/bzalewa

## Apps with JSON

# DEMO

(LoadJSONIntoSQLServer.sql)

RetrieveJSONFromSQLServer.sql

zalnet

/bzalewa

## Resources

Used resources:

https://www.sqlshack.com

https://docs.microsoft.com

http://sqlhints.com

zalnet

/bzalewa

## Q & A

Email: info@zalnet.pl

Skype: beata.zalewa

Blog: https://blog.zalnet.pl

Demo:

https://github.com/bzalewa/DataCommunityLublinMarch2019

Thank you for your precious time☺

zalnet