ADVANCED
TECHNOLOGY
DAYS | 28. i 29. STUDENI 2024.
CINESTAR ARENA ZAGREB

JSON in the world of MSSQL

github.com/matesic-damir/presentations

Damir Matešić, M. Sc. Inf.

- Senior Database Architect @SPAN.eu

- AD 2018 - Leading Data Events in Croatia
- AD 2019 - Introduced SQL/Data Saturday in Croatia
- AD 2020 - Co-founder & organizer of #Dataweekender...

- W: blog.matesic.info
- @: **dmatesic@gmail.com**
- in: linkedin.com/in/dmatesic

Storing JSON in a SQL Database

MS SQL 2016

SQL 2 JSON

- Pretty much like creating XML data (FOR XML) -> **FOR JSON**
- Two modes supported:

  - FOR JSON AUTO
  - FOR JSON PATH

- Additional options

  - INCLUDE_NULL_VALUES
  - ROOT
  - WITHOUT_ARRAY_WRAPPER

SQL 2 JSON – data conversion

Source data type	Destination data type
Char, Varchar, Nchar, NVarchar, Text, Ntext, Date, DateTime, DateTime2, DateTimeOffset, Time, UniqueIdentifier, Smallmoney, Money, XML, HierarchyId, Sql_Variant	String
Tinyint, Smallint, Int, Bigint, Decimal, Float, Numeric	Number
Bit	Boolean
Binary, Varbinary, Image, Rowversion, Timestamp	Base 64 encoded string
null	null
geography, geometry, and CLR-based user defined data types	**not supported**

```sql
SELECT
     C.[CustomerID]
     , C.[CustomerName]
     , C.PhoneNumber
     , C.FaxNumber
     , C.WebsiteURL
     , GETDATE() AS DataDateTime
FROM
     [Sales].[Customers] AS C
FOR JSON AUTO;
```

```json
[
    {
        "CustomerID": 1,
        "CustomerName": "Tailspin Toys (Head Office)",
        "PhoneNumber": "(308) 555-0100",
        "FaxNumber": "(308) 555-0101",
        "WebsiteURL": "http:\/\/www.tailspintoys.com",
        "DataDateTime": "2024-08-27T09:56:45.490"
    },
    {
        "CustomerID": 5,
        "CustomerName": "Tailspin Toys (Gasport, NY)",
        "PhoneNumber": "(212) 555-0100",
        "FaxNumber": "(212) 555-0101",
        "WebsiteURL": "http:\/\/www.tailspintoys.com\/Gasport",
        "DataDateTime": "2024-08-27T09:56:45.490"
    }
]
```

```sql
SELECT
     C.[CustomerID]
     , C.[CustomerName]
     , C.PhoneNumber AS 'Contact.Phone'
     , C.FaxNumber AS 'Contact.Fax'
     , C.WebsiteURL
     , GETDATE() AS DataDateTime
FROM
     [Sales].[Customers] AS C
FOR JSON PATH;
```

```json
[
    {
        "CustomerID": 1,
        "CustomerName": "Tailspin Toys (Head Office)",
        "Contact": {
            "Phone": "(308) 555-0100",
            "Fax": "(308) 555-0101"
        },
        "WebsiteURL": "http:\/\/www.tailspintoys.com",
        "DataDateTime": "2024-08-27T09:59:42.907"
    },
    {
        "CustomerID": 5,
        "CustomerName": "Tailspin Toys (Gasport, NY)",
        "Contact": {
            "Phone": "(212) 555-0100",
            "Fax": "(212) 555-0101"
        },
        "WebsiteURL": "http:\/\/www.tailspintoys.com\/Gasport",
        "DataDateTime": "2024-08-27T09:59:42.907"
    }
]
```

JSON 2 SQL

- OPENJSON
- **rowset function** (table-valued function)
- Two types of return tables:

  - Default schema
  - Explicit schema

OPENJSON - default schema

- OPENJSON (Expression, [Path])

    - **Expression** – JSON object in Unicode text format
    - **Path** – optional argument to specify a fragment (sub-node) of the input expression

- Return - table result with three columns

    - Key
    - Value

0 -> null	1 -> string	2 -> int	3 -> true/false	4 -> array	5 -> object

```sql
DECLARE @JSON_data NVARCHAR(MAX) = N'{
"Name": "John Doe",
"BlogURL": "http:\/\/blog.matesic.info",
"Born": 1979,
"Pets":null,
"BornAfterWoodstock": true,
"FavoriteDrinks": [
{"Name": "Gin and tonic","Drink": "Occasionally"},
{"Name": "Craft beer","Drink": "Occasionally"},
{"Name": "Coffe with milk","Drink": "Daily"},
{"Name": "Cold water","Drink": "Daily"}],
"Parents": {"Mom": "Iva","Dad": "Boris"}
}';
SELECT * FROM OPENJSON(@JSON_data);
```

	key	value	type
1	Name	John Doe	1
2	BlogURL	http://blog.matesic.info	1
3	Born	1979	2
4	Pets	NULL	0
5	BornAfterWoodstock	true	3
6	FavoriteDrinks	[{"Name": "Gin and tonic","Drink": "Occasiona...	4
7	Parents	{"Mom": "Iva","Dad": "Boris"}	5

OPENJSON - explicit schema

- OPENJSON (Expression, [Path])
- [WITH (
-      columnName dataType [columnPath] [AS JSON]
-      [, columnName dataType [columnPath] [AS JSON]]
-   )]
- columnName – Name of the output column
- dataType – Data type of the output column
- columnPath – Optional argument to specify a fragment (sub-node) of the column
- AS JSON – Optional argument to specify that the referenced property contains an inner JSON object or array. If used, the column must be NVARCHAR(MAX) data type
- WITH keyword - at least one column must be specified!!!

```sql
DECLARE @JSON_data NVARCHAR(MAX) = N'{
"Name": "John Doe",
"BlogURL": "http:\/\/blog.matesic.info",
"Born": 1979,
"Pets":null,
"BornAfterWoodstock": true,
"FavoriteDrinks": [
{"Name": "Gin and tonic","Drink": "Occasionally"},{"Name": "Craft beer","Drink": "Occasionally"},
{"Name": "Coffe with milk","Drink": "Daily"},{"Name": "Cold water","Drink": "Daily"}],
"Parents": {"Mom": "Iva","Dad": "Boris"}
}';
SELECT * FROM OPENJSON(@JSON_data) WITH (
    Name NVARCHAR(256) '$.Name',
    [Blog URL] NVARCHAR(256) '$.BlogURL',
    Born INT '$.Born',
    Pets NVARCHAR(256) '$.Pets',
    [Favorite drinks] NVARCHAR(MAX) '$.FavoriteDrinks' AS JSON,
    Parents NVARCHAR(MAX) '$.Parents' AS JSON
) Data;
```

	Name	Blog URL	Born	Pets	Favorite drinks	Parents
1	John Doe	http://blog.matesic.info	1979	NULL	[{"Name": "Gin and tonic","Drink": "Occasiona...	{"Mom": "Iva","Dad": "Boris"}

JSON_VALUE

- extracts a scalar value (primitive data type) from a JSON string

- JSON_VALUE (Expression, [Path])

  - **Expression** – JSON object in Unicode text format
  - **Path** – optional argument to specify a fragment (sub-node) of the input expression

- Return – result of nvarchar(4000) data type with the same collation as in the input expression.

- Can be used in SELECT, WHERE, and ORDER BY clauses

```sql
DECLARE @JSON_data NVARCHAR(MAX) = N'{
"Name": "John Doe",
"BlogURL": "http:\/\/blog.matesic.info",
"Born": 1979,
"Pets":null,
"BornAfterWoodstock": true,
"FavoriteDrinks": [
{"Name": "Gin and tonic","Drink": "Occasionally"},{"Name": "Craft beer","Drink": "Occasionally"},
{"Name": "Coffe with milk","Drink": "Daily"},{"Name": "Cold water","Drink": "Daily"}],
"Parents": {"Mom": "Iva","Dad": "Boris"}
}';
SELECT
JSON_VALUE(@JSON_data, '$.Name') AS Name,
JSON_VALUE(@JSON_data, '$.BlogURL') AS BlogURL,
JSON_VALUE(@JSON_data, '$.Spouse') AS Spouse,
JSON_VALUE(@JSON_data, '$.BornAfterWoodstock') AS BornAfterWoodstock,
JSON_VALUE(@JSON_data, '$.FavoriteDrinks[0].Name') AS FavoriteDrink,
JSON_VALUE(@JSON_data, '$.NonExistingNode') AS NonExistingNode,
JSON_VALUE(@JSON_data, '$.Parents') AS Parents;
```

Name	BlogURL	Pets	BornAfterWoodstock	FavoriteDrink	NonExistingNode	Parents
John Doe	http://blog.matesic.info	NULL	true	Gin and tonic	NULL	NULL

JSON_QUERY

- extract a JSON fragment or to get a complex value (object or array)

- JSON_QUERY (Expression, [Path])

  - **Expression** – JSON object in Unicode text format
  - **Path** – optional argument to specify a fragment (sub-node) of the input expression

- Return – nvarchar(max) if the input string is defined as (n)varchar(max); otherwise -> nvarchar(4000)

```sql
DECLARE @JSON_data NVARCHAR(MAX) = N'{
"Name": "John Doe",
"BlogURL": "http:\\www.microsoft.com",
"Born": 1979,
"Pets":null,
"BornAfterWoodstock": true,
"FavoriteColors": ["Red", "Purple", "Green"]
}';
SELECT
    JSON_QUERY(@JSON_data, '$.Name') AS Name
    , JSON_QUERY(@JSON_data, '$.BornAfterWoodstock') AS BornAfterWoodstock
    , JSON_QUERY(@JSON_data, '$.FavoriteColors') AS FavoriteColors
    , JSON_QUERY(@JSON_data, '$.FavoriteColors[1]') AS SecondColor
```

Name	BornAfterWoodstock	FavoriteColors	SecondColor
NULL	NULL	["Red", "Purple", "Green"]	NULL

Modifying JSON data

- JSON_MODIFY (expression , path , newValue)

  - **Expression** – JSON object in Unicode text format
  - **Path** – A JSON path expression that specifies the property to update
  - **newValue** – The new value for the property specified by path

- Return - updated JSON string

- Adding, Removing, Updating JSON property

- Multiple changes

```sql
-- Adding currently presenting - 1 (bool)
DECLARE @JSON_data NVARCHAR(MAX) = N'{
"Name": "John Doe",
"BlogURL": "http:\/\/www.microsoft.com"
}';
PRINT JSON_MODIFY(@JSON_data, '$."Currently presenting"', CAST(1 AS BIT))
```

```
{
    "Name": "John Doe",
    "BlogURL": "http:\/\/www.microsoft.com",
    "Currently presenting":true
}
```

```sql
-- Adding MS SQL meetups - array
DECLARE @MeetupList NVARCHAR(256) = N'["New SQL 2016/2017 functions","SQL & JSON"]';
DECLARE @JSON_data NVARCHAR(MAX) = N'{
"Name": "John Doe",
"BlogURL": "http:\/\/www.microsoft.com"
}';
PRINT JSON_MODIFY(@JSON_data, '$.Meetups', JSON_QUERY(@MeetupList));
```

```
{
    "Name": "John Doe",
    "BlogURL": "http:\/\/www.microsoft.com",
    "Meetups":["New SQL 2016/2017 functions","SQL & JSON"]
}
```

```sql
-- Removing FavoriteDrinks node
DECLARE @JSON_data NVARCHAR(MAX) = N'{
"Name": "John Doe",
"BlogURL": "http:\/\/www.microsoft.com",
"FavoriteDrinks": [
{"Name": "Gin and tonic","Drink": "Occasionally"},
{"Name": "Craft beer","Drink": "Occasionally"},
{"Name": "Coffe with milk","Drink": "Daily"},
{"Name": "Cold water","Drink": "Daily"}]
,"Meetups":["New SQL 2016/2017 functions","SQL & JSON"]}';
PRINT JSON_MODIFY(@JSON_data, '$.FavoriteDrinks', NULL);
```

```
{
    "Name": "John Doe",
    "BlogURL": "http:\/\/www.microsoft.com",
    "Meetups":["New SQL 2016/2017 functions","SQL & JSON"]
}
```

```sql
-- Update JSON property to NULL instead of remove - strict!
DECLARE @JSON_data NVARCHAR(MAX) = N'{
"Name": "John Doe",
"BlogURL": "http:\/\/www.microsoft.com",
"FavoriteDrinks": [
{"Name": "Gin and tonic","Drink": "Occasionally"},
{"Name": "Craft beer","Drink": "Occasionally"},
{"Name": "Coffe with milk","Drink": "Daily"},
{"Name": "Cold water","Drink": "Daily"}]
,"Meetups":["New SQL 2016/2017 functions","SQL & JSON"]}';
PRINT JSON_MODIFY(@JSON_data, 'strict $.FavoriteDrinks', NULL);
```

```
{
    "Name": "John Doe",
    "BlogURL": "http:\/\/www.microsoft.com",
    "FavoriteDrinks": null,
    "Meetups":["New SQL 2016/2017 functions","SQL & JSON"]
}
```

ISJSON

- To JSON or not to JSON ?

- ISJSON (expression)

- **Expression** – The string to test

- Return – int
  - - 1 - string contains valid JSON
  - - 0 - string is not valid JSON
  - - NULL - input expression is NULL

```
{
    "Name": "John Doe",
    "Name": "John Doe",
    "BlogURL": "http:\/\/www.microsoft.com"
}
```

T&T - Import JSON from a file

```sql
SELECT [key], [value], [type]
FROM OPENROWSET (BULK 'C:\Temp\JSON_data.json', SINGLE_CLOB) AS x
CROSS APPLY OPENJSON(BulkColumn);
```

key	value	type
Name	John Doe	1
BlogURL	http://blog.matesic.info	1
Meetups	["New SQL 2016/2017 functions...	4

T&T - Indexing JSON data

```sql
USE WideWorldImporters;
GO
DROP TABLE IF EXISTS dbo.JSONIndexing;
GO
CREATE TABLE dbo.JSONIndexing(
    [CustomerID] INT NOT NULL,
    [CustomerData] NVARCHAR(2000) NULL,
    CONSTRAINT PK_JSONIndexing PRIMARY KEY CLUSTERED ([CustomerID])
);
GO
INSERT INTO dbo.JSONIndexing ([CustomerID], [CustomerData])
SELECT
    [CustomerID]
    , ( SELECT
            C1.[CustomerName] AS Name
            , PC1.FullName AS PrimaryContact
            , C1.PhoneNumber AS 'Contact.Phone', C1.FaxNumber AS 'Contact.Fax'
            , C1.WebsiteURL
            , C1.DeliveryAddressLine1 AS 'Delivery.AddressLine1', C1.DeliveryAddressLine2 AS 'Delivery.AddressLine2'
            , C1.DeliveryPostalCode AS 'Delivery.PostalCode', DC.CityName AS'Delivery.CityName'
            , C1.PostalAddressLine1 'Postal.AddressLine1' , C1.PostalAddressLine2 'Postal.AddressLine2', C1.PostalPostalCode AS 'Postal.PostalCode'
            , PC.CityName AS'Postal.CityName'
        FROM
            [Sales].[Customers] C1
            LEFT JOIN [Application].[People] PC1 ON C1.PrimaryContactPersonID = PC1.PersonID
            LEFT JOIN [Application].[Cities] DC ON C1.DeliveryCityID = DC.CityID
            LEFT JOIN [Application].[Cities] PC ON C1.PostalCityID = PC.CityID
        WHERE
            C1.CustomerID = C.CustomerID FOR JSON PATH, WITHOUT_ARRAY_WRAPPER)
FROM [Sales].[Customers] C
GO
```

```sql
-- Select Idaho City customers
SELECT *
FROM dbo.JSONIndexing
WHERE JSON_VALUE([CustomerData],'$.Postal.CityName') LIKE '%Idaho City%';
```
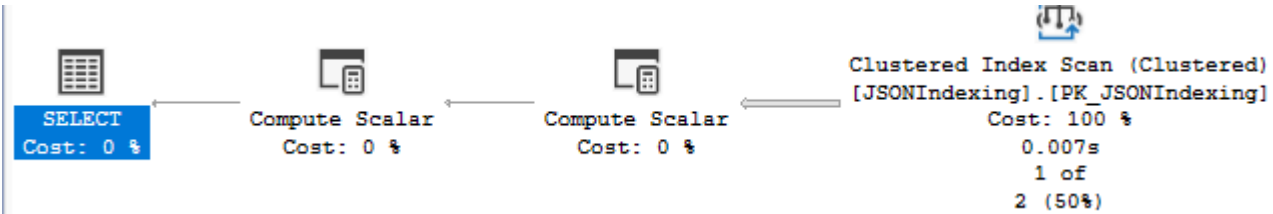

Table 'JSONIndexing'. Scan count 1, logical reads 75
CPU time = 15 ms

```sql
-- Add computed column with index
ALTER TABLE dbo.JSONIndexing ADD Customer_City AS JSON_VALUE([CustomerData], '$.Postal.CityName');
CREATE INDEX IDX_Customer_City ON dbo.JSONIndexing(Customer_City);

-- Select Idaho City customers
SELECT *
FROM dbo.JSONIndexing
WHERE JSON_VALUE([CustomerData],'$.Postal.CityName') LIKE '%Idaho City%';
```
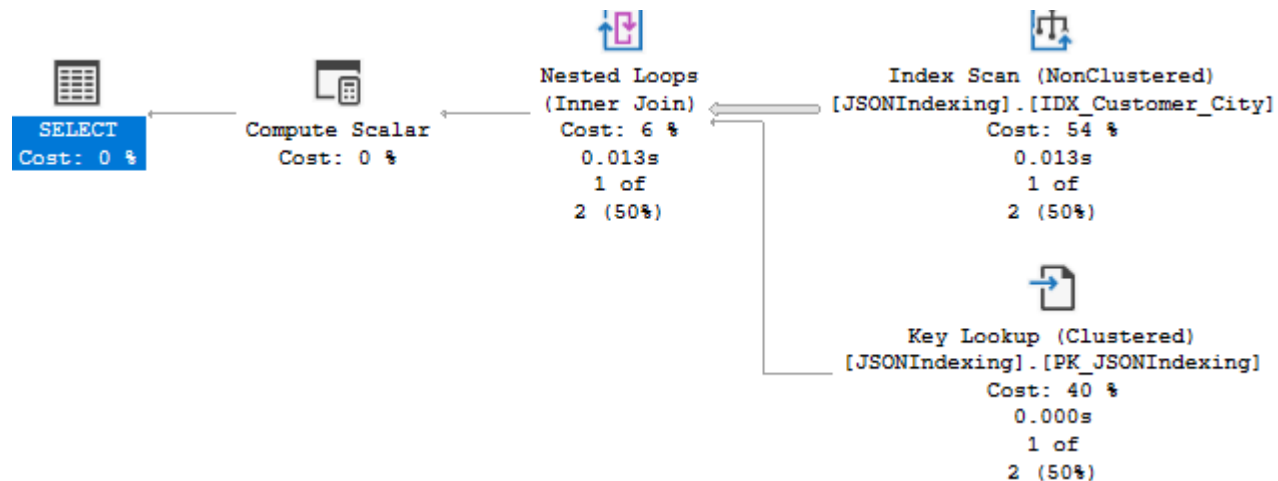

Table 'JSONIndexing'. Scan count 1, logical reads 7
CPU time = 0 ms

T&T – Compare records with hash

```sql
SELECT
    P.*
    , ColumnHashCode =
        HASHBYTES (
            'SHA2_512'
            , (SELECT J.* FROM [Person].[Person] J WHERE J.BusinessEntityID = P.BusinessEntityID FOR JSON AUTO)
        )
FROM
    [Person].[Person] P
```

BusinessEntityID	PersonType	NameStyle	Title	FirstName	MiddleName	LastName	Suffix	EmailPromotion	AdditionalContactInfo	ColumnHashCode
1	EM	0	NULL	Ken	J	Sánchez	NULL	0	NULL	0xD290929D52056B2A70D92FFE3A0D8951AED91A782C0AE0E...
2	EM	0	NULL	Terri	Lee	Duffy	NULL	1	NULL	0xF49354271A5E16AE732901E52A67E879712671F90DC52B677..
3	EM	0	NULL	Roberto	NULL	Tamburello	NULL	0	NULL	0xB9139ED4C434E939B8D7870F57F5245EA532C9B458DA5B00..
4	EM	0	NULL	Rob	NULL	Walters	NULL	0	NULL	0x6E87EA80F2ADCC5D25E59FCC35A9F6BA8C7D469AF40AD95..
5	EM	0	Ms.	Gail	A	Erickson	NULL	0	NULL	0x5C1880E25FAB556C1E65655D4DF23FBDD643D4BAF210BD2..

MS SQL 2022

ISJSON

```
1   SELECT ISJSON('[{"First name":"Bob","Last name":"Doe"}]');
0   SELECT ISJSON('[{"First name":"Bob","Last name:"Doe"}]');

    DECLARE @JSON_data1 NVARCHAR(MAX) = N'{
    "Name": "John Doe",
    "Born": 1979,
    "FavoriteDrinks": [{"Name": "Gin and tonic","Drink": "Occasionally"},{"Name": "Coffe with milk","Drink": "Daily"}]
    }';
1   SELECT ISJSON(@JSON_data1);

    DECLARE @JSON_data2 NVARCHAR(MAX) = N'{
    "Name": "John Doe",
    "Born": 1979,
    "FavoriteDrinks": [{"Name": "Gin and tonic","Drink": "Occasionally"},{"Name": "Coffe with milk","Drink": "Daily"}]
    }';
1   SELECT ISJSON(@JSON_data2, VALUE);

0   SELECT ISJSON ('test string', VALUE)

1   SELECT ISJSON ('[{"First name":"Bob","Last name":"Doe"}]', VALUE)

    DECLARE @JSON_data3 NVARCHAR(MAX) = N'{
    "Name": "John Doe",
    "BornAfterWoodstock": true,
    "FavoriteDrinks": [{"Name": "Gin and tonic","Drink": "Occasionally"},{"Name": "Coffe with milk","Drink": "Daily"}]
    }';
1   SELECT ISJSON (@JSON_data3, OBJECT)

0   SELECT ISJSON ('"test string"', OBJECT)

    DECLARE @JSON_data4 NVARCHAR(MAX) = N'{
    "Name": "John Doe",
    "Born": 1979,
    "FavoriteDrinks": [{"Name": "Gin and tonic","Drink": "Occasionally"},{"Name": "Coffe with milk","Drink": "Daily"}]
    }';
0   SELECT ISJSON (@JSON_data4, ARRAY)

1   SELECT ISJSON ('[{"Name": "Gin and tonic","Drink": "Occasionally"},{"Name": "Coffe with milk","Drink": "Daily"}]', ARRAY)

1   SELECT ISJSON ('"test string"', SCALAR)

0   SELECT ISJSON ('test string', SCALAR)
```

JSON_PATH_EXISTS

```sql
DROP TABLE IF EXISTS sql_requests_table_json_object;
GO
SELECT JSON_OBJECT('command': r.command, 'status': r.status, 'database_id': r.database_id, 'wait_type': r.wait_type, 'wait_resource': r.wait_resource, 'user': s.is_user_process) as json_object, r.command
INTO sql_requests_table_json_object
FROM sys.dm_exec_requests r
JOIN sys.dm_exec_sessions s
ON r.session_id = s.session_id
ORDER BY r.session_id;
GO
SELECT * FROM sql_requests_table_json_object;
GO
```

json_object	command
{"command":"TASK MANAGER","status":"sleeping","databas...	TASK MANAGER
{"command":"TASK MANAGER","status":"sleeping","databas...	TASK MANAGER
{"command":"TASK MANAGER","status":"sleeping","databas...	TASK MANAGER
{"command":"TASK MANAGER","status":"sleeping","databas...	TASK MANAGER
{"command":"TASK MANAGER","status":"sleeping","databas...	TASK MANAGER

```sql
SELECT
    JSON_PATH_EXISTS(json_object, '$.status')
    , JSON_PATH_EXISTS(command, '$.status')
FROM sql_requests_table_json_object;
```

1	1	0
2	1	0
3	1	0
4	1	0
5	1	0

JSON_OBJECT

```sql
DROP TABLE IF EXISTS sql_requests_table_json_object;
GO
SELECT JSON_OBJECT(
    'command': r.command, 'status': r.status, 'database_id': r.database_id, 'wait_type': r.wait_type
    , 'wait_resource': r.wait_resource, 'user': s.is_user_process) as json_object, r.command
INTO sql_requests_table_json_object
FROM sys.dm_exec_requests r
JOIN sys.dm_exec_sessions s
ON r.session_id = s.session_id
ORDER BY r.session_id;
GO
SELECT * FROM sql_requests_table_json_object;
GO
```

json_object	command
{"command":"TASK MANAGER","status":"sleeping","database_id":1,"wait_type":null,"wait_resource":"","user":false}	TASK MANAGER
{"command":"TASK MANAGER","status":"sleeping","database_id":1,"wait_type":null,"wait_resource":"","user":false}	TASK MANAGER
{"command":"TASK MANAGER","status":"sleeping","database_id":1,"wait_type":null,"wait_resource":"","user":false}	TASK MANAGER
{"command":"TASK MANAGER","status":"sleeping","database_id":1,"wait_type":null,"wait_resource":"","user":false}	TASK MANAGER
{"command":"TASK MANAGER","status":"sleeping","database_id":1,"wait_type":null,"wait_resource":"","user":false}	TASK MANAGER

JSON_ARRAY

```sql
DROP TABLE IF EXISTS sql_requests_json_array;
GO
SELECT r.session_id, JSON_ARRAY(r.command, r.status, r. database_id, r.wait_type, r.wait_resource, s.is_user_process) as json_array, r.command
INTO sql_requests_json_array
FROM sys.dm_exec_requests r
JOIN sys.dm_exec_sessions s
ON r.session_id = s.session_id
ORDER BY r.session_id;
GO
SELECT * FROM sql_requests_json_array;
GO
```

session_id	json_array	command
1	["TASK MANAGER","sleeping",1,"",false]	TASK MANAGER
2	["TASK MANAGER","sleeping",1,"",false]	TASK MANAGER
3	["TASK MANAGER","sleeping",1,"",false]	TASK MANAGER
4	["TASK MANAGER","sleeping",1,"",false]	TASK MANAGER
5	["TASK MANAGER","sleeping",1,"",false]	TASK MANAGER

MS SQL (2025) Azure (preview)

JSON_OBJECTAGG

- Constructs a JSON object from an aggregation of SQL data or columns

- The key/value pairs can be specified as input values,

```sql
SELECT TOP(5) c.object_id, JSON_OBJECTAGG(c.name:c.column_id) AS columns
FROM sys.columns AS c
GROUP BY c.object_id;
```

object_id	columns
3	{"bitpos":12,"cid":6,"colguid":13,"hbcolid":3,"maxinrowlen":8,"nullbit":11,"offset":10,"ordkey":7,"ordlock":14,"rcmodified":4,"rscolid":2,"rsid":1,"status":9,"ti":5}
5	{"cmprlevel":9,"fgidfs":7,"fillfact":10,"idmajor":3,"idminor":4,"lockres":17,"maxint":13,"maxleaf":12,"maxnullbit":11,"minint":15,"minleaf":14,"numpart":5,"ownertype":2,"rcrows":8,"rowsetid":1,"rsguid":16,"scope_id":18,"status":6}
6	{"cloneid":6,"dbfragid":8,"id":1,"partid":3,"rowsetid":7,"segid":5,"status":9,"subid":2,"version":4}
7	{"auid":1,"fgid":5,"ownerid":3,"pcdata":10,"pcreserved":11,"pcused":9,"pgfirst":6,"pgfirstiam":8,"pgroot":7,"status":4,"type":2}
8	{"fileid":2,"filename":4,"name":3,"status":1}

JSON_ARRAYAGG

- Constructs a JSON array from an aggregation of SQL data or columns

```sql
SELECT TOP(5) c.object_id, JSON_ARRAYAGG(c.name ORDER BY c.column_id) AS column_list
FROM sys.columns AS c
GROUP BY c.object_id;
```

object_id	column_list
3	["rsid","rscolid","hbcolid","rcmodified","ti","cid","ordkey","maxinrowlen","status","offset","nullbit","bitpos","colguid","ordlock"]
5	["rowsetid","ownertype","idmajor","idminor","numpart","status","fgidfs","rcrows","cmprlevel","fillfact","maxnullbit","maxleaf","maxint","minleaf","minint","rsguid","lockres","scope_id"]
6	["id","subid","partid","version","segid","cloneid","rowsetid","dbfragid","status"]
7	["auid","type","ownerid","status","fgid","pgfirst","pgroot","pgfirstiam","pcused","pcdata","pcreserved"]
8	["status","fileid","name","filename"]

JSON data type

- The new native JSON data type that stores JSON documents in a native binary format

- The JSON type provides a high-fidelity storage of JSON documents optimized for easy querying and manipulation, and provides the following benefits over storing JSON data in varchar or nvarchar:

- More efficient reads, as the document is already parsed
- More efficient writes, as the query can update individual values without accessing the entire document
- More efficient storage, optimized for compression
- No change in compatibility with existing code

JSON data type

- There are no special index types for JSON data for now (SQL 2025)

- Can't be used as key column in a CREATE INDEX statement!

- Can be specified as an included column in an index definition
- Can appear in the WHERE clause of a filtered index

```sql
DROP TABLE IF EXISTS [dbo].[OrdersJSON];
DROP TABLE IF EXISTS [dbo].[OrdersN];

CREATE TABLE [dbo].[OrdersJSON]
(
    [order_id] [INT] NOT NULL,
    [order_details] [JSON] NOT NULL,
    CONSTRAINT [PK_OrdersJSON] PRIMARY KEY CLUSTERED ([order_id] ASC)
);

CREATE TABLE [dbo].[OrdersN]
(
    [order_id] [INT] NOT NULL,
    [order_details] [NVARCHAR](MAX) NOT NULL,
    CONSTRAINT [PK_OrdersN] PRIMARY KEY CLUSTERED ([order_id] ASC)
);

INSERT INTO dbo.OrdersJSON
(
    order_id,
    order_details
)
SELECT OrderID, (SELECT o.* FROM sales.orders o WHERE o.OrderID=f.orderid FOR JSON AUTO) FROM Sales.Orders f;

INSERT INTO dbo.OrdersN
(
    order_id,
    order_details
)
SELECT OrderID, (SELECT o.* FROM sales.orders o WHERE o.OrderID=f.orderid FOR JSON AUTO) FROM Sales.Orders f;

SELECT TOP(100) * FROM dbo.OrdersJSON;
SELECT TOP(100) * FROM dbo.OrdersN;
```

order_id	order_details
1	[{"OrderID":1,"CustomerID":832,"SalespersonPersonID":2,"ContactPersonID":3032,"BackorderOrderID":45,"OrderDate":"2013-01-01","ExpectedDeliveryDate":"2013-01-02",
2	[{"OrderID":2,"CustomerID":803,"SalespersonPersonID":8,"ContactPersonID":3003,"BackorderOrderID":46,"OrderDate":"2013-01-01","ExpectedDeliveryDate":"2013-01-02",
3	[{"OrderID":3,"CustomerID":105,"SalespersonPersonID":7,"ContactPersonID":1209,"BackorderOrderID":47,"OrderDate":"2013-01-01","ExpectedDeliveryDate":"2013-01-02",
4	[{"OrderID":4,"CustomerID":57,"SalespersonPersonID":16,"PickedByPersonID":3,"ContactPersonID":1113,"OrderDate":"2013-01-01","ExpectedDeliveryDate":"2013-01-02","
5	[{"OrderID":5,"CustomerID":905,"SalespersonPersonID":3,"ContactPersonID":3105,"BackorderOrderID":48,"OrderDate":"2013-01-01","ExpectedDeliveryDate":"2013-01-02",
6	[{"OrderID":6,"CustomerID":976,"SalespersonPersonID":13,"PickedByPersonID":3,"ContactPersonID":3176,"OrderDate":"2013-01-01","ExpectedDeliveryDate":"2013-01-02"
7	[{"OrderID":7,"CustomerID":575,"SalespersonPersonID":8,"ContactPersonID":2349,"BackorderOrderID":49,"OrderDate":"2013-01-01","ExpectedDeliveryDate":"2013-01-02",
8	[{"OrderID":8,"CustomerID":964,"SalespersonPersonID":7,"ContactPersonID":3164,"BackorderOrderID":50,"OrderDate":"2013-01-01","ExpectedDeliveryDate":"2013-01-02",

order_id	order_details
1	[{"OrderID":1,"CustomerID":832,"SalespersonPersonID":2,"ContactPersonID":3032,"BackorderOrderID":45,"OrderDate":"2013-01-01","ExpectedDeliveryDate":"2013-01-02",
2	[{"OrderID":2,"CustomerID":803,"SalespersonPersonID":8,"ContactPersonID":3003,"BackorderOrderID":46,"OrderDate":"2013-01-01","ExpectedDeliveryDate":"2013-01-02",
3	[{"OrderID":3,"CustomerID":105,"SalespersonPersonID":7,"ContactPersonID":1209,"BackorderOrderID":47,"OrderDate":"2013-01-01","ExpectedDeliveryDate":"2013-01-02",
4	[{"OrderID":4,"CustomerID":57,"SalespersonPersonID":16,"PickedByPersonID":3,"ContactPersonID":1113,"OrderDate":"2013-01-01","ExpectedDeliveryDate":"2013-01-02","
5	[{"OrderID":5,"CustomerID":905,"SalespersonPersonID":3,"ContactPersonID":3105,"BackorderOrderID":48,"OrderDate":"2013-01-01","ExpectedDeliveryDate":"2013-01-02",
6	[{"OrderID":6,"CustomerID":976,"SalespersonPersonID":13,"PickedByPersonID":3,"ContactPersonID":3176,"OrderDate":"2013-01-01","ExpectedDeliveryDate":"2013-01-02"
7	[{"OrderID":7,"CustomerID":575,"SalespersonPersonID":8,"ContactPersonID":2349,"BackorderOrderID":49,"OrderDate":"2013-01-01","ExpectedDeliveryDate":"2013-01-02",
8	[{"OrderID":8,"CustomerID":964,"SalespersonPersonID":7,"ContactPersonID":3164,"BackorderOrderID":50,"OrderDate":"2013-01-01","ExpectedDeliveryDate":"2013-01-02",

TableName	SchemaName	rows	TotalSpaceKB	TotalSpaceMB	UsedSpaceKB	UsedSpaceMB	UnusedSpaceKB	UnusedSpaceMB
OrdersN	dbo	73595	53512	52.26	53368	52.12	144	0.14
OrdersJSON	dbo	73595	39432	38.51	39232	38.31	200	0.20

```sql
SELECT
    JSON_VALUE(order_details, '$[0].CustomerID') AS CustomerID,
    COUNT(*) AS Cnt
FROM [dbo].[OrdersJSON]
WHERE JSON_VALUE(order_details, '$[0].CustomerID') = 1050
GROUP BY JSON_VALUE(order_details, '$[0].CustomerID')
ORDER BY CustomerID;


SELECT
    JSON_VALUE(order_details, '$[0].CustomerID') AS CustomerID,
    COUNT(*) AS Cnt
FROM [dbo].[OrdersN]
WHERE JSON_VALUE(order_details, '$[0].CustomerID') = 1050
GROUP BY JSON_VALUE(order_details, '$[0].CustomerID')
ORDER BY CustomerID;


SELECT * FROM [dbo].[OrdersJSON] O CROSS APPLY OPENJSON(O.order_details)


SELECT *
FROM [dbo].[OrdersJSON] O
    CROSS APPLY
    OPENJSON(CAST(O.order_details AS NVARCHAR(MAX)))
    WITH
    (
        OrderID INT '$.OrderID',
        OrderDate DATE '$.OrderDate'
    ) F
WHERE F.OrderDate = '2013-01-25';

SELECT *
FROM [dbo].[OrdersN] O
    CROSS APPLY
    OPENJSON(O.order_details)
    WITH
    (
        OrderID INT '$.OrderID',
        OrderDate DATE '$.OrderDate'
    ) F
WHERE F.OrderDate = '2013-01-25';
```

```
Table 'Worktable'. Scan count 0, logical reads 0
Table 'OrdersJSON'. Scan count 1, logical reads 4904

 SQL Server Execution Times:
   CPU time = 172 ms,   elapsed time = 169 ms.


Table 'OrdersN'. Scan count 5, logical reads 7005
Table 'Worktable'. Scan count 0, logical reads 0

 SQL Server Execution Times:
   CPU time = 372 ms,   elapsed time = 95 ms.


Msg 13665, Level 16, State 2, Line 53
OpenJson support not yet supported for JSON native data type.


Table 'OrdersJSON'. Scan count 1, logical reads 4904

SQL Server Execution Times:
   CPU time = 1563 ms,   elapsed time = 1588 ms.


Table 'OrdersN'. Scan count 1, logical reads 6671

SQL Server Execution Times:
   CPU time = 593 ms,   elapsed time = 583 ms.
```

THANK YOU