DATA WITH BERT

The Ultimate SQL Server JSON Cheat Sheet

Published on: 2017-03-07

EXAMPLES FOR HANDLING JSON DATA IN SQL SERVER 2016+

This post is a reference of my examples for processing JSON data in SQL Server. For more detailed explanations of these functions, please see my post series on JSON in SQL Server 2016:

- *Part 1—Parsing JSON*
- *Part 2—Creating JSON*
- *Part 3—Updating, Adding, and Deleting JSON*
- *Part 4—JSON Performance Comparison*

Additionally, the complete reference for SQL JSON handling can be found at MSDN: https://msdn.microsoft.com/en-us/library/dn921897.aspx

Parsing JSON

Getting string JSON data into a SQL readable form.

ISJSON()

Checks to see if the input string is valid JSON.

```
SELECT ISJSON('{ "Color" : "Blue" }') -- Returns 1, valid
-- Output: 1

SELECT ISJSON('{ "Color" : Blue }') -- Returns 0, invalid,
missing quotes
-- Output: 0

SELECT ISJSON('{ "Number" : 1 }') -- Returns 1, valid, numbers
are allowed
-- Output: 1

SELECT ISJSON('{ "PurchaseDate" : "2015-08-18T00:00:00.000Z" }')
-- Returns 1, valid, dates are just strings in ISO 8601 date
format https://en.wikipedia.org/wiki/ISO_8601
-- Output: 1

SELECT ISJSON('{ "PurchaseDate" : 2015-08-18 }') -- Returns 0,
invalid
-- Output: 0
```

JSON_VALUE()

Extracts a specific scalar string value from a JSON string using JSON path expressions.

```
-- See
https://gist.github.com/bertwagner/356bf47732b9e35d2156daa943e049
e9 for a formatted version of this JSON
DECLARE @garage nvarchar(1000) = N'{ "Cars": [{ "Make":
"Volkswagen", "Model": { "Base": "Golf", "Trim": "GL" }, "Year":
2003, "PurchaseDate": "2006-10-05T00:00:00.000Z" }, { "Make":
"Subaru", "Model": { "Base": "Impreza", "Trim": "Premium" },
"Year": 2016, "PurchaseDate": "2015-08-18T00:00:00.000Z" }] }'

SELECT JSON_VALUE(@garage, '$.Cars[0].Make') -- Return the make
of the first car in our array
-- Output: Volkswagen

SELECT CAST(JSON_VALUE(@garage, '$.Cars[0].PurchaseDate') as
datetime2) -- Return the Purchase Date of the first car in our
array and convert it into a DateTime2 datatype
-- Output: 2006-10-05 00:00:00.0000000

SELECT JSON_VALUE(@garage, '$.Cars') -- This returns NULL because
the values of Cars is an array instead of a simple object
-- Output: NULL

SELECT JSON_VALUE(@garage, '$.Cars[1].Model') -- This is also
invalid because JSON_VALUE cannot return an array...only scalar
values allowed!
-- Output: NULL

SELECT JSON_VALUE(@garage, '$.Cars[1].Model.Base') -- Much better
-- Output: Impreza
```

STRICT VS. LAX MODE

If the JSON path cannot be found, determines if the function should return a NULL or an error message.

```
-- Lax (default: function will return an error if invalid JSON
path specified
SELECT JSON_VALUE('{ "Color" : "Red" }', '$.Shape') --lax is the
default, so you don't need to be explicitly state it
-- Output: NULL

SELECT JSON_VALUE('{ "Color" : "Red" }', 'lax $.Shape')
```

```
-- Output: NULL

-- Strict: function will return an error if invalid JSON path
specified
SELECT JSON_VALUE('{ "Color" : "Red" }', 'strict $.Shape')
-- Output: Property cannot be found on the specified JSON path.
```

JSON_QUERY()

Returns a JSON fragment for the specified JSON path.

```
-- See
https://gist.github.com/bertwagner/356bf47732b9e35d2156daa943e049
e9 for a formatted version of this JSON
DECLARE @garage nvarchar(1000) = N'{ "Cars": [{ "Make":
"Volkswagen", "Model": { "Base": "Golf", "Trim": "GL" }, "Year":
2003, "PurchaseDate": "2006-10-05T00:00:00.000Z" }, { "Make":
"Subaru", "Model": { "Base": "Impreza", "Trim": "Premium" },
"Year": 2016, "PurchaseDate": "2015-08-18T00:00:00.000Z" }] }'

-- This returns NULL because the values of Cars is an array
instead of a simple object
SELECT JSON_VALUE(@garage, '$.Cars')
-- Output: NULL

-- Using JSON_QUERY() however returns the JSON string
representation of our array object
SELECT JSON_QUERY(@garage, '$.Cars')
-- Output: [{ "Make": "Volkswagen", "Model": { "Base": "Golf",
"Trim": "GL" }, "Year": 2003, "PurchaseDate": "2006-10-
05T00:00:00.000Z" }, { "Make": "Subaru", "Model": { "Base":
"Impreza", "Trim": "Premium" }, "Year": 2016, "PurchaseDate":
"2015-08-18T00:00:00.000Z" }]

-- This instance of JSON_VALUE() correctly returns a singular
scalar value
SELECT JSON_VALUE(@garage, '$.Cars[0].Make')
-- Output: Volkswagen

-- Using JSON_QUERY will not work for returning scalar values -
it only will return JSON strings for complex objects
SELECT JSON_QUERY(@garage, '$.Cars[0].Make')
-- Output: NULL
```

This is useful to help filter an array and then extract values with JSON_VALUE():

```
-- See
https://gist.github.com/bertwagner/356bf47732b9e35d2156daa943e049
e9 for a formatted version of this JSON
DECLARE @garage nvarchar(1000) = N'{ "Cars": [{ "Make":
"Volkswagen", "Model": { "Base": "Golf", "Trim": "GL" }, "Year":
2003, "PurchaseDate": "2006-10-05T00:00:00.000Z" }, { "Make":
"Subaru", "Model": { "Base": "Impreza", "Trim": "Premium" },
"Year": 2016, "PurchaseDate": "2015-08-18T00:00:00.000Z" }] }'

-- We use JSON_QUERY to get the JSON representation of the Cars
array
SELECT JSON_QUERY(@garage, '$.Cars')
-- Output: [{ "Make": "Volkswagen", "Model": { "Base": "Golf",
"Trim": "GL" }, "Year": 2003, "PurchaseDate": "2006-10-
05T00:00:00.000Z" }, { "Make": "Subaru", "Model": { "Base":
"Impreza", "Trim": "Premium" }, "Year": 2016, "PurchaseDate":
"2015-08-18T00:00:00.000Z" }]

-- If we combine it with JSON_VALUE we can then pull out specific
scalar values
SELECT JSON_VALUE(JSON_QUERY(@garage, '$.Cars') , '$[0].Make')
-- Output: Volkswagen
```

OPEN_JSON()

Returns a SQL result set for the specified JSON path. The result set includes columns identifying the datatypes of the parsed data.

```
-- See
https://gist.github.com/bertwagner/356bf47732b9e35d2156daa943e049
e9 for a formatted version of this JSON
DECLARE @garage nvarchar(1000) = N'{ "Cars": [{ "Make":
"Volkswagen", "Model": { "Base": "Golf", "Trim": "GL" }, "Year":
2003, "PurchaseDate": "2006-10-05T00:00:00.000Z" }, { "Make":
"Subaru", "Model": { "Base": "Impreza", "Trim": "Premium" },
"Year": 2016, "PurchaseDate": "2015-08-18T00:00:00.000Z" }] }'

SELECT * FROM OPENJSON(@garage, '$.Cars') -- Displaying the
```

```
values of our "Cars" array.  We additionally get the order of the
JSON objects outputted in the "key" column and the JSON object
datatype in the "type" column
/* Output:
key    value
type
------ ------------------------------------------------------------
------------------------------------------------------------------
--------- ----
0      { "Make": "Volkswagen", "Model": { "Base": "Golf", "Trim":
"GL" }, "Year": 2003, "PurchaseDate": "2006-10-05T00:00:00.000Z"
}        5
1      { "Make": "Subaru", "Model": { "Base": "Impreza", "Trim":
"Premium" }, "Year": 2016, "PurchaseDate": "2015-08-
18T00:00:00.000Z" }    5
*/

SELECT * FROM OPENJSON(@garage, '$.Cars[0]') -- Specifying the
first element in our JSON array.  JSON arrays are zero-index
based
/* Output:
key              value                                type
---------------- ------------------------------------ ----
Make             Volkswagen                           1
Model            { "Base": "Golf", "Trim": "GL" }     5
Year             2003                                 2
PurchaseDate     2006-10-05T00:00:00.000Z             1
*/

SELECT * FROM OPENJSON(@garage, '$.Cars[0].Model') -- Pulling the
Model property from the first element in our Cars array
/* Output:
key     value   type
------- ------- ----
Base    Golf    1
Trim    GL      1
*/
```

```
-- See
https://gist.github.com/bertwagner/356bf47732b9e35d2156daa943e049
e9 for a formatted version of this JSON
DECLARE @garage nvarchar(1000) = N'{ "Cars": [{ "Make":
"Volkswagen", "Model": { "Base": "Golf", "Trim": "GL" }, "Year":
2003, "PurchaseDate": "2006-10-05T00:00:00.000Z" }, { "Make":
"Subaru", "Model": { "Base": "Impreza", "Trim": "Premium" },
"Year": 2016, "PurchaseDate": "2015-08-18T00:00:00.000Z" }] }'
```

```
-- Here we retrieve the Make of each vehicle in our Cars array
SELECT JSON_VALUE(value, '$.Make') FROM OPENJSON(@garage,
'$.Cars')
/* Output:
------------
Volkswagen
Subaru
*/

-- Parsing and converting some JSON dates to SQL DateTime2
SELECT CAST(JSON_VALUE(value, '$.PurchaseDate') as datetime2)
FROM OPENJSON(@garage, '$.Cars')
/* Output:
--------------------------
2006-10-05 00:00:00.0000000
2015-08-18 00:00:00.0000000
*/

-- We can also format the output schema of a JSON string using
the WITH option.  This is especially cool because we can bring up
values from sub-arrays (see Model.Base and Model.Trim) to our
top-level row result
SELECT * FROM OPENJSON(@garage, '$.Cars')
 WITH (Make varchar(20) 'strict $.Make',
       ModelBase nvarchar(100) '$.Model.Base',
           ModelTrim nvarchar(100) '$.Model.Trim',
            Year int '$.Year',
       PurchaseDate datetime2 '$.PurchaseDate')
/* Output:
Make           ModelBase   Year        PurchaseDate
-------------- ----------- ----------- --------------------------
-
Volkswagen     Golf        2003        2006-10-05
00:00:00.0000000
Subaru         Impreza     2016        2015-08-18
00:00:00.0000000
*/
```

Creating JSON

Creating JSON data from either strings or result sets.

FOR JSON AUTO

Automatically creates a JSON string from a SELECT statement. Quick and dirty.

```
-- Create our table with test data
DROP TABLE IF EXISTS ##Garage;
CREATE TABLE ##Garage
(
        Id int IDENTITY(1,1),
        Make varchar(100),
        BaseModel varchar(50),
        Trim varchar(50),
        Year int,
        PurchaseDate datetime2
);
INSERT INTO ##Garage VALUES ('Volkswagen', 'Golf', 'GL', 2003,
'2006-10-05');
INSERT INTO ##Garage VALUES ('Subaru', 'Impreza', 'Premium',
2016, '2015-08-18');

-- Take a look at our data
SELECT * FROM ##Garage;
```

```
-- AUTO will format a result into JSON following the same
structure of the result set
SELECT Make, BaseModel, Trim, Year, PurchaseDate
FROM ##Garage
FOR JSON AUTO;
-- Output:
[{"Make":"Volkswagen","BaseModel":"Golf","Trim":"GL","Year":2003,
"PurchaseDate":"2006-10-05T00:00:00"},
{"Make":"Subaru","BaseModel":"Impreza","Trim":"Premium","Year":20
16,"PurchaseDate":"2015-08-18T00:00:00"}]

-- Using aliases will rename JSON keys
SELECT Make AS [CarMake]
FROM ##Garage
FOR JSON AUTO;
-- Output: [{"CarMake":"Volkswagen"},{"CarMake":"Subaru"}]

-- Any joined tables will get created as nested JSON objects.
The alias of the joined tables becomes the name of the JSON key
SELECT g1.Make,  Model.BaseModel as Base, Model.Trim, g1.Year,
g1.PurchaseDate
FROM ##Garage g1
```

```
INNER JOIN ##Garage Model on g1.Id = Model.Id
FOR JSON AUTO;
-- Output:
[{"Make":"Volkswagen","Year":2003,"PurchaseDate":"2006-10-
05T00:00:00","Model":[{"Base":"Golf","Trim":"GL"}]},
{"Make":"Subaru","Year":2016,"PurchaseDate":"2015-08-
18T00:00:00","Model":[{"Base":"Impreza","Trim":"Premium"}]}]

-- Finally we can encapsulate our entire JSON result in a parent
element by specifiying the ROOT option
SELECT Make, BaseModel, Trim, Year, PurchaseDate
FROM ##Garage
FOR JSON AUTO, ROOT('Cars');
-- Output: {"Cars":
[{"Make":"Volkswagen","BaseModel":"Golf","Trim":"GL","Year":2003,
"PurchaseDate":"2006-10-05T00:00:00"},
{"Make":"Subaru","BaseModel":"Impreza","Trim":"Premium","Year":20
16,"PurchaseDate":"2015-08-18T00:00:00"}]}
```

FOR JSON PATH

Formats a SQL query into a JSON string, allowing the user to define structure and formatting.

```
-- PATH will format a result using dot syntax in the column
aliases.  Here's an example with just default column names
SELECT Make, BaseModel, Trim, Year, PurchaseDate
FROM ##Garage
FOR JSON PATH, ROOT('Cars');
-- Output: {"Cars":
[{"Make":"Volkswagen","BaseModel":"Golf","Trim":"GL","Year":2003,
"PurchaseDate":"2006-10-05T00:00:00"},
{"Make":"Subaru","BaseModel":"Impreza","Trim":"Premium","Year":20
16,"PurchaseDate":"2015-08-18T00:00:00"}]}

-- And here is the same example, just assigning aliases to define
JSON nested structure
SELECT Make, BaseModel as [Model.Base], Trim AS [Model.Trim],
Year, PurchaseDate
FROM ##Garage
FOR JSON PATH, ROOT('Cars');
-- Output: {"Cars":[{"Make":"Volkswagen","Model":
{"Base":"Golf","Trim":"GL"},"Year":2003,"PurchaseDate":"2006-10-
05T00:00:00"},{"Make":"Subaru","Model":
```

```
{"Base":"Impreza","Trim":"Premium"},"Year":2016,"PurchaseDate":"2
015-08-18T00:00:00"}]}

-- We can actually go multiple levels deep with this type of
alias dot notation nesting
SELECT Make, BaseModel as [Model.Base], Trim AS [Model.Trim],
'White' AS [Model.Color.Exterior], 'Black' AS
[Model.Color.Interior], Year, PurchaseDate
FROM ##Garage
FOR JSON PATH, ROOT('Cars');
-- Output: {"Cars":[{"Make":"Volkswagen","Model":
{"Base":"Golf","Trim":"GL","Color":
{"Exterior":"White","Interior":"Black"}},"Year":2003,"PurchaseDat
e":"2006-10-05T00:00:00"},{"Make":"Subaru","Model":
{"Base":"Impreza","Trim":"Premium","Color":
{"Exterior":"White","Interior":"Black"}},"Year":2016,"PurchaseDat
e":"2015-08-18T00:00:00"}]}

-- Concatenating data rows with UNION or UNION ALL just adds the
row as a new element as part of the JSON array
SELECT Make,  BaseModel AS [Model.Base], Trim AS [Model.Trim],
Year, PurchaseDate
FROM ##Garage WHERE Id = 1
UNION ALL
SELECT Make,  BaseModel, Trim, Year, PurchaseDate
FROM ##Garage WHERE Id = 2
FOR JSON PATH, ROOT('Cars');
-- Output: {"Cars":[{"Make":"Volkswagen","Model":
{"Base":"Golf","Trim":"GL"},"Year":2003,"PurchaseDate":"2006-10-
05T00:00:00"},{"Make":"Subaru","Model":
{"Base":"Impreza","Trim":"Premium"},"Year":2016,"PurchaseDate":"2
015-08-18T00:00:00"}]}

-- We can even include our FOR JSON in our SELECT statement to
generate JSON strings for each row of our result set
SELECT g1.*, (SELECT Make, BaseModel AS [Model.Base], Trim AS
[Model.Trim], Year, PurchaseDate FROM ##Garage g2 WHERE g2.Id =
g1.Id FOR JSON PATH, ROOT('Cars')) AS [Json]
FROM ##Garage g1
/* Output:
Id  Make          BaseModel    Trim      Year    PurchaseDate
Json
--- ------------ ------------ --------- ------- ---------------
----------- ------------------------------------------------------
------------------------------------------------------------------
----
1   Volkswagen    Golf         GL        2003    2006-10-05
00:00:00.0000000 {"Cars":[{"Make":"Volkswagen","Model":
{"Base":"Golf","Trim":"GL"},"Year":2003,"PurchaseDate":"2006-10-
```

```
05T00:00:00"}]}
2   Subaru        Impreza      Premium   2016    2015-08-18
00:00:00.0000000 {"Cars":[{"Make":"Subaru","Model":
{"Base":"Impreza","Trim":"Premium"},"Year":2016,"PurchaseDate":"2
015-08-18T00:00:00"}]}
*/
```

Modifying JSON

Updating, adding to, and deleting from JSON data.

JSON_MODIFY()

Allows the user to update properties and values, add properties and values, and delete properties and values (the delete is unintuitive, see below).

Modify:

```
-- See
https://gist.github.com/bertwagner/356bf47732b9e35d2156daa943e049
e9 for a formatted version of this JSON
DECLARE @garage nvarchar(1000) = N'{ "Cars": [{ "Make":
"Volkswagen", "Model": { "Base": "Golf", "Trim": "GL" }, "Year":
2003, "PurchaseDate": "2006-10-05T00:00:00.000Z" }, { "Make":
"Subaru", "Model": { "Base": "Impreza", "Trim": "Premium" },
"Year": 2016, "PurchaseDate": "2015-08-18T00:00:00.000Z" }] }'

-- I upgraded some features in my Volkswagen recently,
technically making it equivalent to a "GLI" instead of a "GL".
-- Let's update our JSON using JSON_MODIFY:
SET @garage = JSON_MODIFY(@garage, '$.Cars[0].Model.Trim', 'GLI')
SELECT @garage
-- Output: { "Cars": [{ "Make": "Volkswagen", "Model": { "Base":
"Golf", "Trim": "GLI" }, "Year": 2003, "PurchaseDate": "2006-10-
05T00:00:00.000Z" }, { "Make": "Subaru", "Model": { "Base":
"Impreza", "Trim": "Premium" }, "Year": 2016, "PurchaseDate":
"2015-08-18T00:00:00.000Z" }] }
```

Add:

```
-- See
https://gist.github.com/bertwagner/356bf47732b9e35d2156daa943e049
e9 for a formatted version of this JSON
DECLARE @garage nvarchar(1000) = N'{ "Cars": [{ "Make":
"Volkswagen", "Model": { "Base": "Golf", "Trim": "GLI" }, "Year":
2003, "PurchaseDate": "2006-10-05T00:00:00.000Z" }, { "Make":
"Subaru", "Model": { "Base": "Impreza", "Trim": "Premium" },
"Year": 2016, "PurchaseDate": "2015-08-18T00:00:00.000Z" }] }'

-- I decided to sell my Golf.  Let's add a new "SellDate"
property to the JSON saying when I sold my Volkswagen.
-- If we use strict mode, you'll see we can't add SellDate
because the key never existed before
--SELECT JSON_MODIFY(@garage, 'append strict $.Cars[0].SellDate',
'2017-02-17T00:00:00.000Z')
-- Output: Property cannot be found on the specified JSON path.

-- However, in lax mode (default), we have no problem adding the
SellDate
SELECT JSON_MODIFY(@garage, 'append lax $.Cars[0].SellDate',
'2017-02-17T00:00:00.000Z')
-- Output: { "Cars": [{ "Make": "Volkswagen", "Model": { "Base":
"Golf", "Trim": "GLI" }, "Year": 2003, "PurchaseDate": "2006-10-
05T00:00:00.000Z" ,"SellDate":["2017-02-17T00:00:00.000Z"]}, {
"Make": "Subaru", "Model": { "Base": "Impreza", "Trim": "Premium"
}, "Year": 2016, "PurchaseDate": "2015-08-18T00:00:00.000Z" }] }

-- After selling my Golf, I bought another car a few days later:
A new Volkswagen Golf GTI.  Let's add it to our garge:
-- Note the use of JSON_QUERY; this is so our string is
interpreted as a JSON object instead of a plain old string
SET @garage = JSON_MODIFY(@garage, 'append $.Cars', JSON_QUERY('{
"Make": "Volkswagen", "Model": { "Base": "Golf", "Trim": "GTI" },
"Year": 2017, "PurchaseDate": "2017-02-19T00:00:00.000Z" }'))
SELECT @garage;
-- Output: { "Cars": [{ "Make": "Volkswagen", "Model": { "Base":
"Golf", "Trim": "GLI" }, "Year": 2003, "PurchaseDate": "2006-10-
05T00:00:00.000Z" }, { "Make": "Subaru", "Model": { "Base":
"Impreza", "Trim": "Premium" }, "Year": 2016, "PurchaseDate":
"2015-08-18T00:00:00.000Z" },{ "Make": "Volkswagen", "Model": {
"Base": "Golf", "Trim": "GTI" }, "Year": 2017, "PurchaseDate":
"2017-02-19T00:00:00.000Z" }] }
```

Delete property:

```
DECLARE @garage nvarchar(1000) = N'{ "Cars": [{ "Make":
"Volkswagen", "Model": { "Base": "Golf", "Trim": "GLI" }, "Year":
2003, "PurchaseDate": "2006-10-05T00:00:00.000Z", "SellDate" :
"2017-02-17T00:00:00.000Z" }, { "Make": "Subaru", "Model": {
"Base": "Impreza", "Trim": "Premium" }, "Year": 2016,
"PurchaseDate": "2015-08-18T00:00:00.000Z" },{ "Make":
"Volkswagen", "Model": { "Base": "Golf", "Trim": "GTI" }, "Year":
2017, "PurchaseDate": "2017-02-19T00:00:00.000Z" }] }'

-- Let's remove the PurchaseDate property on my original
Volkswagen Golf since it's not relevant anymore:
SET @garage = JSON_MODIFY(@garage, '$.Cars[0].PurchaseDate',
NULL)
SELECT @garage
-- Output: { "Cars": [{ "Make": "Volkswagen", "Model": { "Base":
"Golf", "Trim": "GLI" }, "Year": 2003, "SellDate" : "2017-02-
17T00:00:00.000Z" }, { "Make": "Subaru", "Model": { "Base":
"Impreza", "Trim": "Premium" }, "Year": 2016, "PurchaseDate":
"2015-08-18T00:00:00.000Z" },{ "Make": "Volkswagen", "Model": {
"Base": "Golf", "Trim": "GTI" }, "Year": 2017, "PurchaseDate":
"2017-02-19T00:00:00.000Z" }] }
```

Delete from array (this is not intuitive, see my Microsoft Connect item to fix this: https://connect.microsoft.com/SQLServer/feedback/details/3120404/sql-modify-json-null-delete-is-not-consistent-between-properties-and-arrays)

```
DECLARE @garage nvarchar(1000) = N'{ "Cars": [{ "Make":
"Volkswagen", "Model": { "Base": "Golf", "Trim": "GLI" }, "Year":
2003, "SellDate" : "2017-02-17T00:00:00.000Z" }, { "Make":
"Subaru", "Model": { "Base": "Impreza", "Trim": "Premium" },
"Year": 2016, "PurchaseDate": "2015-08-18T00:00:00.000Z" },{
"Make": "Volkswagen", "Model": { "Base": "Golf", "Trim": "GTI" },
"Year": 2017, "PurchaseDate": "2017-02-19T00:00:00.000Z" }] }'

-- I realize it's not worth keeping the original Volkswagen in my
@garage data any longer, so let's completely remove it.
-- Note, if we use NULL as per the MSDN documentation, we don't
actually remove the first car element of the array - it just gets
```

```
replaced with NULL
-- This is problematic if we expect the indexes of our array to
shift by -1.
SELECT JSON_MODIFY(@garage, '$.Cars[0]', NULL)
-- Output: { "Cars": [null, { "Make": "Subaru", "Model": {
"Base": "Impreza", "Trim": "Premium" }, "Year": 2016,
"PurchaseDate": "2015-08-18T00:00:00.000Z" },{ "Make":
"Volkswagen", "Model": { "Base": "Golf", "Trim": "GTI" }, "Year":
2017, "PurchaseDate": "2017-02-19T00:00:00.000Z" }] }

-- To truly delete it (and not have the NULL appear as the first
item in the array) we have to convert to a rowset, select
everything that's not the first row, aggregate the rows into a
string (UGH) and then recreate as JSON.
-- This is incredibly ugly.  The STREAM_AGG() function in SQL
vNext should make it a little cleaner, but why doesn't the
JSON_MODIFY NULL syntax just get rid of the element in the array?
-- I have opened a Microsoft connect issue for this here:
https://connect.microsoft.com/SQLServer/feedback/details/3120404
SELECT JSON_QUERY('{ "Cars" : [' +
                STUFF((
                            SELECT       ',' + value
               FROM OPENJSON(@garage, '$.Cars')
                            WHERE [key] <> 0
               FOR XML PATH('')), 1, 1, '') + '] }')
-- Output: { "Cars" : [{ "Make": "Subaru", "Model": { "Base":
"Impreza", "Trim": "Premium" }, "Year": 2016, "PurchaseDate":
"2015-08-18T00:00:00.000Z" },{ "Make": "Volkswagen", "Model": {
"Base": "Golf", "Trim": "GTI" }, "Year": 2017, "PurchaseDate":
"2017-02-19T00:00:00.000Z" }] }
```

SQL JSON Performance Tuning

SQL JSON functions are already fast. Adding computed columns and indexes makes them extremely fast.

COMPUTED COLUMN JSON INDEXES

JSON indexes are simply regular indexes on computed columns.

Add a computed column:

```
-- Car data source: https://github.com/arthurkao/vehicle-make-
model-data
IF OBJECT_ID('dbo.Cars') IS NOT NULL
BEGIN
        DROP TABLE dbo.Cars;
END
CREATE TABLE dbo.Cars
(
        Id INT IDENTITY(1,1),
        CarDetails NVARCHAR(MAX)
);
-- See
https://gist.github.com/bertwagner/1df2531676112c24cd1ab298fc750e
b2 for the full untruncated version of this code
DECLARE @cars nvarchar(max) = '[
{"year":2001,"make":"ACURA","model":"CL"},
{"year":2001,"make":"ACURA","model":"EL"},...]';

INSERT INTO dbo.Cars (CarDetails)
SELECT value FROM OPENJSON(@cars, '$');

SELECT * FROM dbo.Cars;
/*
Output:
Id          CarDetails
----------- ----------------------------------------------
1           {"year":2001,"make":"ACURA","model":"CL"}
2           {"year":2001,"make":"ACURA","model":"EL"}
3           {"year":2001,"make":"ACURA","model":"INTEGRA"}
...
*/
```

```
-- Remember to turn on "Include Actual Execution Plan" for all of
these examples

-- Before we add any computed columns/indexes, let's see our
execution plan for our SQL statement with a JSON predicate
SELECT * FROM dbo.Cars WHERE JSON_VALUE(CarDetails, '$.model') =
'Golf'
/*
Output:
Id          CarDetails
----------- ----------------------------------------------------
1113        {"year":2001,"make":"VOLKSWAGEN","model":"GOLF"}
2410        {"year":2002,"make":"VOLKSWAGEN","model":"GOLF"}
```

```
3707        {"year":2003,"make":"VOLKSWAGEN","model":"GOLF"}
...
*/
-- The execution plan shows a Table Scan, not very efficient

-- We can now add a non-persisted computed column for our "model"
JSON property.
ALTER TABLE dbo.Cars
ADD CarModel AS JSON_VALUE(CarDetails, '$.model');

-- We add the distinct to avoid parameter sniffing issues.
-- Our execution plan now shows the extra computation that is
occuring for every row of the table scan.
SELECT DISTINCT * FROM dbo.Cars WHERE JSON_VALUE(CarDetails,
'$.model') = 'Golf'
SELECT DISTINCT * FROM dbo.Cars WHERE CarModel = 'Golf'
```

Add an index to our computed column:

```
-- Add an index onto our computed column
CREATE CLUSTERED INDEX CL_CarModel ON dbo.Cars (CarModel)

-- Check the execution plans again
SELECT DISTINCT * FROM dbo.Cars WHERE JSON_VALUE(CarDetails,
'$.model') = 'Golf'
SELECT DISTINCT * FROM dbo.Cars WHERE CarModel = 'Golf'
-- We now get index seeks!
```

```
-- Indexed computed column returns results in ~1ms
SELECT * FROM dbo.Cars WHERE CarModel = 'Golf'
```

Performance test:

```
-- Turn on stats and see how long it takes to parse the ~20k JSON
array elements
SET STATISTICS TIME ON

-- Test #1
-- Test how long it takes to parse each property from all ~20k
```

```
elements from the JSON array
-- SQL returns this query in ~546ms
SELECT JSON_VALUE(value, '$.year') AS [Year], JSON_VALUE(value,
'$.make') AS Make, JSON_VALUE(value, '$.model') AS Model FROM
OPENJSON(@cars, '$')

-- Test #2
-- Time to deserialize and query just Golfs without computed
column + index
-- This takes ~255ms in SQL Server
SELECT * FROM OPENJSON(@cars, '$') WHERE JSON_VALUE(value,
'$.model') = 'Golf'

-- Test #3
-- Time it takes to compute the same query for Golf's with a
computed column and clustered index
-- This takes ~1ms on SQL Server
SELECT * FROM dbo.Cars WHERE CarModel = 'Golf'

-- Test #4
-- Serializing data on SQL Server takes ~110ms
SELECT * FROM dbo.Cars FOR JSON AUTO

-- What about serializing/deserializing smaller JSON datasets?
-- Let's create our smaller set
DECLARE @carsSmall nvarchar(max) = '[
{"year":2001,"make":"ACURA","model":"CL"},
{"year":2001,"make":"ACURA","model":"EL"},
{"year":2001,"make":"ACURA","model":"INTEGRA"},
{"year":2001,"make":"ACURA","model":"MDX"},
{"year":2001,"make":"ACURA","model":"NSX"},
{"year":2001,"make":"ACURA","model":"RL"},
{"year":2001,"make":"ACURA","model":"TL"}]';

-- Test #5
-- Running our query results in the data becoming deserialized in
~0ms
SELECT JSON_VALUE(value, '$.year') AS [Year], JSON_VALUE(value,
'$.make') AS Make, JSON_VALUE(value, '$.model') AS Model FROM
OPENJSON(@carsSmall, '$')
--30ms in sql

-- Test #6
-- And serialized in ~0ms
SELECT TOP 7  * FROM dbo.Cars FOR JSON AUTO
```

Thanks for reading. You might also enjoy [following me on Twitter.]

Want to learn even more SQL?

Sign up for my newsletter to receive weekly SQL tips!

Email Address

SUBSCRIBE

9 thoughts on “The Ultimate SQL Server JSON Cheat Sheet”

Paul H

October 25, 2018 at 10:11 am

Good job Bert! Excellent examples to refer to. Thanks for sharing:)

Bert

October 25, 2018 at 1:01 pm

Thanks for reading, glad you find it useful!

Paul H

November 30, 2018 at 11:25 am

I have a question if you don’t mind? Where you have the example:

SELECT JSON_VALUE(@garage, '$.Cars[0].Make') — Return the make of the first car in our array

How could I return $.Cars[1].Make, $.Cars[2].Make etc. when I don't know how many values are in the array?

Thanks,
Paul

Bert

November 30, 2018 at 11:29 am

Hi Paul, you'd want to use the OPENJSON function to return the array values as rows.

Paul H

November 30, 2018 at 11:37 am

Ah, well that is what I am using. This is part of the query I am writing:

```
SELECT
r.ts
,r.scheduled_nb_test_template_id
,r.agent_id
,r.nb_test_id
,r.result_values
,r.result_values_key
,result_values_value
FROM OPENJSON(@scheduled_nb_test_results_JSON_doc,'$.nb_test_results'
)
WITH (
ts bigint '$.ts'
,scheduled_nb_test_template_id INT '$.scheduled_nb_test_template_id'
,agent_id INT '$.agent_id'
,nb_test_id INT '$.nb_test_id'
,result_values NVARCHAR(max) '$.result_values' AS JSON
```

```
,result_values_key nVARCHAR(max) '$.result_values[0].key'
,result_values_value nVARCHAR(50) '$.result_values[0].value'
) r
```

You can see the array part at the bottom. However, there is a variable number of key and value columns in the array. Sometimes just one [0], sometimes up to six [5]. How do I return them all without knowing in advance how many there will be?
Thanks.

Paul H

November 30, 2018 at 11:50 am

Don't worry Bert! I have stumbled across the answer – using CROSS APPLY:

```
SELECT
r.ts
--,DATEADD(ss,CONVERT(INT, LEFT(ts, 10)),'19700101') AS [TimeStamp] --
convert epoch time to datetime
,r.scheduled_nb_test_template_id
,r.agent_id
,r.nb_test_id
,r.result_values
--,r.result_values_key
--,result_values_value
,t1.[key]
,t1.[value]
FROM OPENJSON(@scheduled_nb_test_results_JSON_doc,'$.nb_test_results'
)
WITH (
ts bigint '$.ts'
,scheduled_nb_test_template_id INT '$.scheduled_nb_test_template_id'
,agent_id INT '$.agent_id'
,nb_test_id INT '$.nb_test_id'
,result_values NVARCHAR(max) '$.result_values' AS JSON
--,result_values_key nVARCHAR(max) '$.result_values[0].key'
```

```
–,result_values_value nVARCHAR(50) '$.result_values[0].value'
) r

CROSS APPLY OPENJSON(r.result_values, N'strict $')
WITH(
[key] nvarchar(max) '$.key'
,[value] nvarchar(max) '$.value'
) AS t1
```

Thanks for your help.

James Halligan

December 17, 2019 at 6:58 am

Hi there,

using SQL below to extract JSON to SQL Server...
Any idea how do I deal with addressline which is nested twice under address.addresslines. Every other field is nested once (or not at all)

thanks in advance

James

```
Declare @JSON varchar(max)

SELECT @JSON = BulkColumn
FROM OPENROWSET (BULK '\\pcgan0911\c$\deaspExtractFile.json',
SINGLE_CLOB) as j

SELECT * FROM OPENJSON (@JSON)
With (
RECORD_TYPE VARCHAR(100) '$.recordType',
TAX_YEAR INT '$.taxYear',
PPSN VARCHAR(4000) '$.employeeID.employeePpsn',
EMPLOY_ID VARCHAR(4000) '$.employeeID.employmentID',
```

```
FIRST_NAME VARCHAR(4000) '$.name.firstName',
FAMILY_NAME VARCHAR(4000) '$.name.familyName',
ADDRESS_LINES VARCHAR(4000) '$.address.addresslines.addressline',
) as Dataset
Go
```

---

ponvel shanmuganathan

May 22, 2020 at 11:22 am

```
drop table if exists #tt
create table #TT (TopicsJson varchar(max))

declare @json varchar(max)
set @json = '[{"TopicGroup":"TopicGroup-1","Topics":
["Perks_Complex","Perks_NonSale","Perks_Simple","Perks_Terms_and_Conditions"]},{"TopicGroup":"TopicGroup-2","Topics":["CSAT Negative"]},
{"TopicGroup":"TopicGroup-3","Topics":
["Connection_Issues","COVID_19"]}]'

insert into #tt values (@json)

select _TopicGroup, _Topics
from #tt a with(nolock)
outer apply openjson(TopicsJson, '$')
WITH(
_TopicGroup VARCHAR(100) '$.TopicGroup',
_TopicsJson NVARCHAR(MAX) '$.Topics' AS JSON
)
outer apply openjson(_TopicsJson)
WITH(
_Topics VARCHAR(100) '$'
)
where ISJSON(TopicsJson) > 0
```

above sample is working as expected for me. When topic group and topics associated to millions of records and query based on Topics and TopicGroup

taking forever to return back. Is there a way to create computed column on top of Topics and TopicGroup ? . Please share syntax if any. Thanks in advance.

Ballingam Chepuri

August 16, 2020 at 12:49 pm

Hey, I have a question on how to append one json object to another? For example I have json object called “cars” created using FOR JSON PATH, ROOT(‘cars’);
I want to append “boats” created using FOR JSON PATH, ROOT(‘boats’);

Preferably create such a json output using t-sql. Thank you!