**Advanced JSON Techniques in SQL Server 2016 - Part 2**

|  |  |
| --- | --- |
|  | By: [Aaron Bertrand](https://www.mssqltips.com/sqlserverauthor/49/aaron-bertrand/)  FROM: <https://www.mssqltips.com/sqlservertip/4081/advanced-json-techniques-in-sql-server-2016--part-2/> |

**Problem**

In my previous tip, [Advanced JSON Techniques in SQL Server 2016 - Part 1](https://www.mssqltips.com/sqlservertip/4073/sql-server-2016-advanced-json-techniques--part-1/), I gave examples of three new functions to support JSON features: OPENJSON(), ISJSON(), and JSON\_VALUE(). In this tip, I wanted to show how you can use JSON\_VALUE() to index particular data points within a JSON string.

**Solution**

Let's say you have a simple table to store data about customers, and because different customers can have different attributes, you've chosen to store those attributes in an NVARCHAR column using JSON structure. The original table is pretty simple:

CREATE TABLE dbo.Customers

(

CustomerID INT NOT NULL,

DateCreated DATETIME2 NOT NULL DEFAULT SYSDATETIME(),

[GUID] UNIQUEIDENTIFIER NOT NULL DEFAULT NEWID(),

Attributes NVARCHAR(4000)

CONSTRAINT PK\_Cust\_A PRIMARY KEY(CustomerID),

CONSTRAINT IsValidJSON\_A CHECK (ISJSON(Attributes) = 1)

);

A sample row would have the following JSON structure in the Attributes column:

{"HairColor" : "red", "EyeColor : "green", "YearJoined" : 1974 }

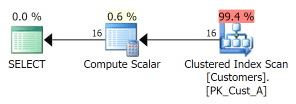
Now, if I want to find all the customers with red hair, I would run a query like this:

SELECT CustomerID, JSON\_VALUE(Attributes, '$.HairColor')

FROM dbo.Customers\_A

WHERE JSON\_VALUE(Attributes, '$.HairColor') = N'red';

This, of course, would have to scan the entire table, according to [SQL Sentry Plan Explorer](https://www.sqlsentry.com/products/plan-explorer/sql-server-query-view):



A little trick [I've learned from Jovan Popovic](http://blogs.msdn.com/b/jocapc/archive/2015/05/16/json-support-in-sql-server-2016.aspx), though, is that you can create a computed column from JSON\_VALUE(), and index the computed column. So let's create a new copy of the table:

CREATE TABLE dbo.Customers\_B

(

CustomerID INT NOT NULL,

DateCreated DATETIME2 NOT NULL DEFAULT SYSDATETIME(),

[GUID] UNIQUEIDENTIFIER NOT NULL DEFAULT NEWID(),

Attributes NVARCHAR(4000),

**HairColor AS JSON\_VALUE(Attributes, '$.HairColor'),**

CONSTRAINT PK\_Cust\_B PRIMARY KEY(CustomerID),

CONSTRAINT IsValidJSON\_B CHECK (ISJSON(Attributes) = 1)

);

And then we can create this index:

CREATE INDEX HairColor ON dbo.Customers\_B(HairColor);

Note that you'll get a warning here, because SQL Server has no way to know how long a specific JSON attribute will be - it could, theoretically, be almost the entire column:

Warning! The maximum key length for a nonclustered index is 1700 bytes. The index 'HairColor' has maximum length of 8000 bytes. For some combination of large values, the insert/update operation will fail.

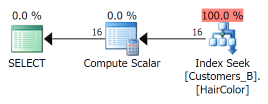
Now we can make the query simpler:

SELECT CustomerID, HairColor

FROM dbo.Customers\_B

WHERE HairColor = N'red';

And now we are likely to get a much more efficient index seek:



We also get the same plan for the original query, so SQL Server is already doing a great job of recognizing the JSON\_VALUE() arguments and matching it to our indexed, computed column.

We could also choose to make the column persisted, which can mean less work for queries at the cost of more work for DML:

CREATE TABLE dbo.Customers\_C

(

CustomerID INT NOT NULL,

DateCreated DATETIME2 NOT NULL DEFAULT SYSDATETIME(),

[GUID] UNIQUEIDENTIFIER NOT NULL DEFAULT NEWID(),

Attributes NVARCHAR(4000),

HairColor AS JSON\_VALUE(Attributes, '$.HairColor') **PERSISTED**,

CONSTRAINT PK\_Cust\_C PRIMARY KEY(CustomerID),

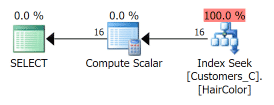
CONSTRAINT IsValidJSON\_C CHECK (ISJSON(Attributes) = 1)

);

GO

CREATE INDEX HairColor ON dbo.Customers\_C(HairColor);

This leads again to the same plan and, at least at my small data volumes, similar performance. At scale, though, [persisted computed columns are worth considering](https://www.mssqltips.com/sqlservertip/3346/what-effect-does-persisting-a-computed-column-have-in-sql-server):



If you want to avoid the warning from above, or - more importantly - failures from inserting JSON strings with extremely long values for the HairColor attribute - you can create this index instead:

CREATE TABLE dbo.Customers\_D

(

CustomerID INT NOT NULL,

DateCreated DATETIME2 NOT NULL DEFAULT SYSDATETIME(),

[GUID] UNIQUEIDENTIFIER NOT NULL DEFAULT NEWID(),

Attributes NVARCHAR(4000),

HairColor AS JSON\_VALUE(Attributes, '$.HairColor') PERSISTED,

CONSTRAINT PK\_Cust\_D PRIMARY KEY(CustomerID),

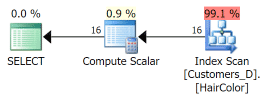
CONSTRAINT IsValidJSON\_D CHECK (ISJSON(Attributes) = 1)

);

GO

**CREATE INDEX HairColor ON dbo.Customers\_D(CustomerID) INCLUDE(HairColor);**

This works, however since the HairColor value is no longer in the key for the index, there is no way for SQL Server to satisfy this specific query without performing a scan - though it chooses this skinnier index instead of the wider clustering key:



Also note that I tried creating a filtered index, since, wouldn't it be nice to be able to create a smaller index that only has the rows we might care about? Alas, I tried this:

CREATE INDEX HairColor2 ON dbo.Customers\_B(HairColor) WHERE HairColor = N'red';

And was promptly told:

Msg 10609, Level 16, State 1   
Filtered index 'HairColor2' cannot be created on table 'dbo.Customers\_B' because the column 'HairColor' in the filter expression is a computed column. Rewrite the filter expression so that it does not include this column.

**Summary**

JSON structures can be stored in SQL Server, but by default they just look like big blobs of text to the optimizer. You can use computed columns and indexes to make operations for extracting and searching specific JSON elements more efficient.

**Next Steps**

* Download [SQL Server 2016 CTP3 when it becomes available](https://www.microsoft.com/en-us/evalcenter/evaluate-sql-server-2016) (or register for a trial of [Azure SQL Database](https://azure.microsoft.com/en-us/pricing/free-trial/), where this feature will also be available).
* Make sure your database compatibility level is 130; during the CTP period at least, JSON features may not function under older compatibility levels.
* Try out JSON in scenarios where it may seem useful.
* See these related tips and other resources:
  + [Advanced JSON Techniques in SQL Server 2016 - Part 1](https://www.mssqltips.com/sqlservertip/4073/sql-server-2016-advanced-json-techniques--part-1/)
  + [Introducing JSON for SQL Server 2016](https://www.mssqltips.com/sqlservertip/4015/introducing-json-for-sql-server-2016)
  + [JSON Support in SQL Server 2016](https://www.mssqltips.com/sqlservertip/4014/json-support-in-sql-server-2016)
  + [SQL Server Database Engine Blog](http://blogs.msdn.com/b/sqlserverstorageengine)

Last Update: 11/3/2015