<https://blogs.technet.microsoft.com/dataplatforminsider/2016/01/06/json-in-sql-server-2016-part-2-of-4/>

# Storing JSON Data in SQL Server

Since external systems format information as JSON text, JSON is also stored in SQL Server as text. You can use standard NVARCHAR columns to store JSON data. A simple table where some information stored as JSON is shown in the following example:

CREATE TABLE Person (  
 Id int IDENTITY PRIMARY KEY NONCLUSTERED,  
 FirstName nvarchar(100) NOT NULL,   
 LastName nvarchar(100) NOT NULL,

 InfoJson nvarchar(max)  
   
) WITH (MEMORY\_OPTIMIZED=ON)

Here you can see the main difference between traditional relational-only and document-only systems and the hybrid model that SQL Server 2016 provides. In SQL Server 2016, you can combine both scalar columns (FirstName and LastName in this example) and columns that contain JSON (InfoJSON in this example).

In SQL Server, you can organize your data as one or many scalar or JSON columns and combine them with Spatial or XML columns if needed. In the same table, you can combine standard relational columns that enable faster access and JSON columns that provide flexibility and rapid application development. Unlike relational-only or document-only storage where you need to choose between two principles of data modeling, SQL Server offers you a hybrid data storage model where you can use the best of both data modeling methods.

Although JSON is stored in a text column, it is not just “a plain text." SQL Server has built-in support for optimizing storage of text columns using various compression mechanisms such as [UNICODE compression](https://technet.microsoft.com/en-us/library/ee240835(v=sql.110).aspx?wt.mc_id=WW_CE_DM_OO_BLOG_NONE) that can provide up to a 50% compression ratio. You can also store JSON text in column store tables or compress it explicitly using the built-in [COMPRESS](https://msdn.microsoft.com/en-US/library/mt622775.aspx?wt.mc_id=WW_CE_DM_OO_BLOG_NONE) function that uses the GZip algorithm.

JSON is fully compatible with any SQL Server component or technology that works with NVARCHAR data. In the example above, JSON is stored in an In-memory OLTP (Hekaton) table that provides extreme processing performance. You can store JSON in standard tables, column store indexes, or in FILESTREAM. You can also load it from Hadoop using Polybase external tables, read it from file systems, stretch it to Azure SQL, use any replication method, and more. If you combine tables that store JSON documents with other SQL Server features, such as Temporal or Row-Level Security, you might find some powerful features that are not available in the existing document databases.

If you don’t want to keep JSON as a free text format, you can add a validation that verifies that JSON in the text column is properly formatted using standard CHECK constraints and ISJSON function:

ALTER TABLE Person  
ADD CONSTRAINT [Content should be formatted as JSON]  
 CHECK ( ISJSON( InfoJSON )> 0 )

This is a standard SQL Server check constraint that enables you to validate whether the text stored in the JSON column is properly formatted. This constraint is optional – you can leave a plain text column as in the previous example; however, your queries might fail at runtime if your JSON text in a row is not properly formatted or if you don’t add the ISJSON condition in the WHERE clause to exclude invalid JSON columns.

Since JSON is represented as text, you don’t need to make any changes in your client applications, wait for new drivers, or change protocol. You can read or write JSON documents in your C#, Java, and Node.js applications as standard string values. JSON can be loaded in ORM models as string fields and be directly sent to JavaScript client-side code via Ajax requests. Any ETL tool can also load or read JSON because there is no new format or interface.

# Built-in functions for JSON processing

SQL Server 2016 provides functions for parsing and processing JSON text. JSON built-in functions that are available in SQL Server 2016 are:

* ISJSON( jsonText ) checks if the NVARCHAR text is properly formatted according to the JSON specification. You can use this function to create check constraints on NVARCHAR columns that contain JSON text
* JSON\_VALUE( jsonText, path ) parses jsonText and extracts scalar values on the specified JavaScript-like path (see below for some JSON path examples)
* JSON\_QUERY( jsonText, path ) that parses jsonText and extracts objects or arrays on the specified JavaScript-like path (see below for some JSON path examples)

These functions use JSON paths for referencing values or objects in JSON text. JSON paths use JavaScript-like syntax for referencing properties in JSON text. Some examples are:

* '$' – references entire JSON object in the input text
* '$.property1' – references property1 in JSON object
* '$[4]' – references 5-th element in JSON array (indexes are counted from 0 like in JavaScript)
* '$.property1.property2.array1[5].property3.array2[15].property4' – references complex nested property in the JSON object
* '$.info. "first name"' – references "first name" property in info object. If key contains some special characters such as space, dollar, etc., it should be surrounded with double quotes

Dollar sign ($) represents the input JSON object (similar to root “/” in XPath language). You can add any JavaScript-like property or array after “$” to reference properties in JSON object. One simple example of a query where these built-in functions are used is:

SELECT Id, FirstName, LastName,

     JSON\_VALUE(InfoJSON, '$.info.”social security number”') as SSN,  
     JSON\_QUERY(InfoJSON, '$.skills') as Skills  
FROM Person AS t   
WHERE ISJSON( InfoJSON ) > 0  
AND JSON\_VALUE(InfoJSON, '$.Type') = 'Student'

This query returns first name and last name information from standard table columns, social security numbers, and an array of skills from the JSON column. Results are returned from rows where the InfoJSON cell contains a valid JSON and the type value in the JSON column is ‘Student.’ As you may notice, you can use JSON values in any part of the query, such as order by, group by, etc.

Check out the other posts in this four-part series in the links below (as they become available), or learn more in the [SQL Server 2016 blogging series](https://blogs.technet.microsoft.com/b/dataplatforminsider/archive/2015/12/01/get-the-most-out-of-sql-server-2016.aspx?wt.mc_id=WW_CE_DM_OO_BLOG_NONE).

JSON in SQL Server 2016: [Part 1 of 4](https://blogs.technet.microsoft.com/b/dataplatforminsider/archive/2016/01/05/json-in-sql-server-2016-part-1-of-4.aspx?wt.mc_id=WW_CE_DM_OO_BLOG_NONE)

JSON in SQL Server 2016: [Part 3 of 4](https://blogs.technet.microsoft.com/b/dataplatforminsider/archive/2016/01/07/json-in-sql-server-2016-part-3-of-4.aspx?wt.mc_id=WW_CE_DM_OO_BLOG_NONE)

JSON in SQL Server 2016: [Part 4 of 4](https://blogs.technet.microsoft.com/b/dataplatforminsider/archive/2016/01/08/json-in-sql-server-2016-part-4-of-4.aspx?wt.mc_id=WW_CE_DM_OO_BLOG_NONE)