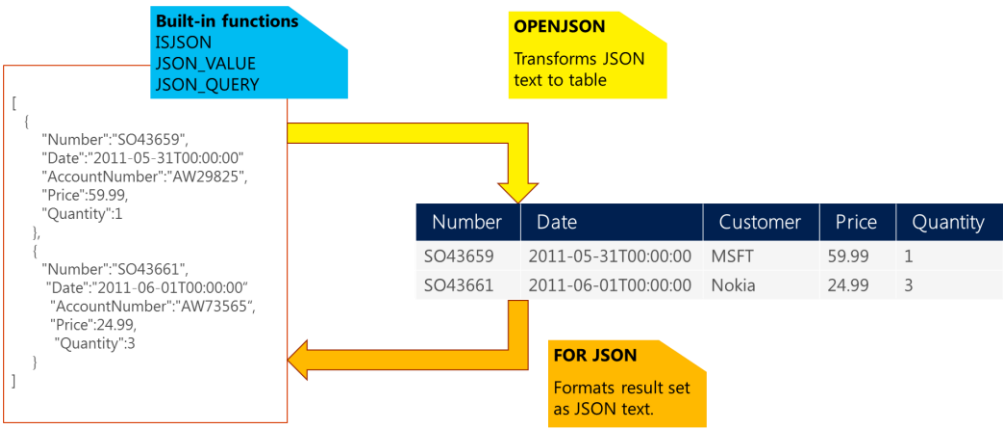
JSON functions in SQL Server enable you to analyze and query JSON data, transform JSON to relational format, and export SQL query results as JSON text.

Part 1 of 4 shown for introduction to JSON functions with SQL 2016



If you have JSON text, you can extract data from JSON or verify that JSON is properly formatted using built-in functions JSON_VALUE, JSON_QUERY, and ISJSON. For more advanced querying and analysis, the OPENJSON function can transform an array of JSON objects into a set of rows. Any SQL query can be executed on the returned result set. Finally, there is the FOR JSON clause that enables you to format query results as JSON text.

We can start with simple examples. In the following Transact-SQL code, we will define a text variable where we will put JSON text:

```
DECLARE @json NVARCHAR(4000)
SET @json =
N'{
   "info":{
     "type":1,

     "address":{
       "town":"Bristol",
       "county":"Avon",
       "country":"England"
     },
     "tags":["Sport", "Water polo"]
   },
   "type":"Basic"
}'
```

Now, we can extract values and objects from JSON text using the JSON_VALUE and JSON_QUERY functions:

```
SELECT
  JSON_VALUE(@json, '$.type') as type,
  JSON_VALUE(@json, '$.info.address.town') as town,
  JSON_QUERY(@json, '$.info.tags') as tags
```

This query will return "Basic", "Bristol", and ["Sport", "Water polo"] values. The JSON_VALUE function returns one scalar value from JSON text (e.g. strings, numbers, true/false) that is placed on a JSON path specified as the second parameter. JSON_QUERY returns an object or array (in this example an array of tags) on the JSON path. JSON built-in functions use JavaScript-like syntax to reference values and objects in JSON text via second parameter.

The OPENJSON function enables you to reference some array in JSON text and return elements from that array:

```
SELECT value
FROM OPENJSON(@json, '$.info.tags')
```

In this example, string values from the tags array are returned. However, the OPENJSON function can return any complex object.

Finally, there is a FOR JSON clause that can format any result set returned by SQL query as JSON text:

```
SELECT object_id, name
FROM sys.tables
FOR JSON PATH
```

--Technet Example

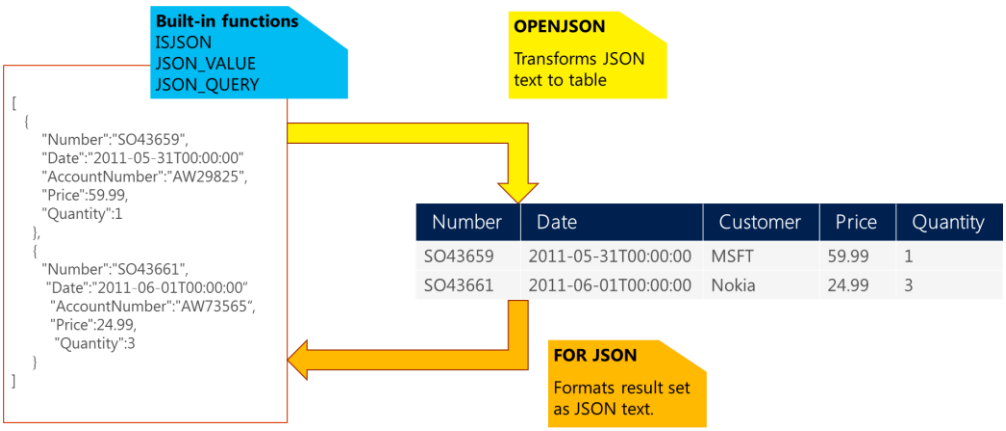
JSON is a popular textual data format used for exchanging data in modern web and mobile applications. JSON is also used for storing unstructured data in log files or NoSQL databases like Microsoft Azure DocumentDB. Many REST web services return results formatted as JSON text or accept data formatted as JSON. For example, most Azure services such as Azure Search, Azure Storage, and Azure DocumentDb have REST endpoints that return or consume JSON. JSON is also the main format for exchanging data between web pages and web servers using AJAX calls.

Here's an example of JSON text:

```
[{
   "name": "John",
   "skills": ["SQL", "C#", "Azure"]
}, {
   "name": "Jane",
   "surname": "Doe"
}]
```

SQL Server provides built-in functions and operators that let you do the following things with JSON text.

- Parse JSON text and read or modify values.
- Transform arrays of JSON objects into table format.
- Run any Transact-SQL query on the converted JSON objects.
- Format the results of Transact-SQL queries in JSON format.



Key JSON capabilities of SQL Server

Here's more info about the key capabilities that SQL Server provides with its built-in JSON support.

Extract values from JSON text and use them in queries

If you have JSON text that's stored in database tables, you can use built-in functions to read or modify values in the JSON text.

- Use the **JSON_VALUE** function to extract a scalar value from a JSON string.

- Use **JSON_QUERY** to extract an object or an array from a JSON string.
- Use the **ISJSON** function to test whether a string contains valid JSON.
- Use the **JSON_MODIFY** function to change a value in a JSON string.

Example

In the following example, the query uses both relational and JSON data (stored in the jsonCol column) from a table:

```
SELECT Name,Surname,
 JSON_VALUE(jsonCol,'$.info.address.PostCode') AS PostCode,
 JSON_VALUE(jsonCol,'$.info.address."Address Line 1"')+' '
  +JSON_VALUE(jsonCol,'$.info.address."Address Line 2"') AS Address,
 JSON_QUERY(jsonCol,'$.info.skills') AS Skills
FROM PeopleCollection
WHERE ISJSON(jsonCol)>0
 AND JSON_VALUE(jsonCol,'$.info.address.Town')='Belgrade'
 AND Status='Active'
ORDER BY JSON_VALUE(jsonCol,'$.info.address.PostCode')
```

Applications and tools see no difference between the values taken from scalar table columns and the values taken from JSON columns. You can use values from JSON text in any part of a Transact-SQL query (including WHERE, ORDER BY, or GROUP BY clauses, window aggregates, and so on). JSON functions use JavaScript-like syntax for referencing values inside JSON text.+

For more info, see Validate, Query, and Change JSON Data with Built-in Functions (SQL Server), JSON_VALUE (Transact-SQL), and JSON_QUERY (Transact-SQL).

Change JSON values

If you have to modify parts of JSON text, you can use the **JSON_MODIFY** function to update the value of a property in a JSON string and return the updated JSON string. The following example updates the value of a property in a variable that contains JSON.

```
DECLARE @jsonInfo NVARCHAR(MAX)
```

```
SET @jsonInfo=JSON_MODIFY(@jsonInfo,'$.info.address[0].town','London')
```

Convert JSON collections to a rowset

You don't need a custom query language to query JSON in SQL Server. To query JSON data, you can use standard T-SQL. If you have to create a query or report on JSON data, you can easily convert JSON data to rows and columns by calling the **OPENJSON** rowset function. For more info, see Convert JSON Data to Rows and Columns with OPENJSON (SQL Server). +

The following example calls **OPENJSON** and transforms the array of objects stored in the `@json` variable to a rowset that can be queried with a standard SQL **SELECT** statement:

```
DECLARE @json NVARCHAR(MAX)
SET @json =
N'[
     { "id" : 2,"info": { "name": "John", "surname": "Smith" }, "age": 25 },
     { "id" : 5,"info": { "name": "Jane", "surname": "Smith" }, "dob": "2005-11-04T12:00:00" }
 ]'

SELECT *
FROM OPENJSON(@json)
  WITH (id int 'strict $.id',
       firstName nvarchar(50) '$.info.name', lastName nvarchar(50) '$.info.surname',
       age int, dateOfBirth datetime2 '$.dob')
```

Results

id	firstName	lastName	age	dateOfBirth
2	John	Smith	25	
5	Jane	Smith		2005-11-04T12:00:00

OPENJSON transforms the array of JSON objects into a table in which each object is represented as one row, and key/value pairs are returned as cells. The output observes the following rules.+

- **OPENJSON** converts JSON values to the types specified in the **WITH** clause.
- **OPENJSON** can handle both flat key/value pairs and nested, hierarchically organized objects.
- You don't have to return all the fields contained in the JSON text.
- **OPENJSON** returns NULL values if JSON values don't exist.
- You can optionally specify a path after the type specification to reference a nested property or to reference a property by a different name.
- The optional **strict** prefix in the path specifies that values for the specified properties must exist in the JSON text.

For more info, see Convert JSON Data to Rows and Columns with OPENJSON (SQL Server) and OPENJSON (Transact-SQL). +

Convert SQL Server data to JSON or export JSON

Format SQL Server data or the results of SQL queries as JSON by adding the **FOR JSON** clause to a **SELECT** statement. Use FOR JSON to delegate the formatting of JSON output from your client applications to SQL Server. For more info, see Format Query Results as JSON with FOR JSON (SQL Server).

The following example uses PATH mode with the FOR JSON clause.

```
SELECT id, firstName AS "info.name", lastName AS "info.surname", age, dateOfBirth as dob
FROM People
FOR JSON PATH
```

The **FOR JSON** clause formats SQL results as JSON text that can be provided to any app that understands JSON. The PATH option uses dot-separated aliases in the SELECT clause to nest objects in the query results.

Results


```
[{
   "id": 2,
   "info": {
      "name": "John",
      "surname": "Smith"
   },
   "age": 25
}, {
   "id": 5,
   "info": {
      "name": "Jane",
      "surname": "Smith"
   },
   "dob": "2005-11-04T12:00:00"
}]
```


For more info, see Format Query Results as JSON with FOR JSON (SQL Server) and FOR Clause (Transact-SQL).

Combine relational and JSON data

SQL Server provides a hybrid model for storing and processing both relational and JSON data using standard Transact-SQL language. You can organize collections of your JSON documents in tables, establish relationships between them, combine strongly-typed scalar columns stored in tables with flexible key/value pairs stored in JSON columns, and query both scalar and JSON values in one or more tables using full Transact-SQL.

JSON text is typically stored in varchar or nvarchar columns and is indexed as plain text. Any SQL Server feature or component that supports text supports JSON, so there are almost no constraints on interaction between JSON and other SQL Server features. You can store JSON in In-memory or Temporal tables, you can apply Row-Level Security predicates on JSON text, and so on.+

If you have pure JSON workloads where you want to use a query language that's customized for the processing of JSON documents, consider Microsoft Azure DocumentDB.

Here are some use cases that show how you can use the built-in JSON support in SQL Server .

Return data from a SQL Server table formatted as JSON

If you have a web service that takes data from the database layer and returns it in JSON format, or JavaScript frameworks or libraries that accept data formatted as JSON, you can format JSON output directly in a SQL query. Instead of writing code or including a library to convert tabular query results and then serialize objects to JSON format, you can use FOR JSON to delegate the JSON formatting to SQL Server.

For example, you might want to generate JSON output that's compliant with the OData specification. The web service expects a request and response in the following format.

- Request: `/Northwind/Northwind.svc/Products(1)?$select=ProductID,ProductName`
- Response:

  `{"@odata.context":"http://services.odata.org/V4/Northwind/Northwind.svc/$metadata#Products(ProductID,ProductName)/$entity","ProductID":1,"ProductName":"Chai"}`

  This OData URL represents a request for the ProductID and ProductName columns for the product with id 1. You can use **FOR JSON** to format the output as expected in SQL Server.

```
SELECT
'http://services.odata.org/V4/Northwind/Northwind.svc/$metadata#Products(ProductID,ProductName)/$entity'
 AS '@odata.context',
 ProductID, Name as ProductName
FROM Production.Product
WHERE ProductID = 1
FOR JSON AUTO
```

The output of this query is JSON text that's fully compliant with OData spec. Formatting and escaping are handled by SQL Server. SQL Server can also format query results in any format such as OData JSON or GeoJSON - for more info, see Returning spatial data in GeoJSON format. +

Analyze JSON data with SQL queries

If you have to filter or aggregate JSON data for reporting purposes, you can use **OPENJSON** to transform JSON to relational format. Then use standard Transact-SQL and built-in functions to prepare the reports.

```
SELECT Tab.Id, SalesOrderJsonData.Customer, SalesOrderJsonData.Date
FROM   SalesOrderRecord AS Tab
       CROSS APPLY
   OPENJSON (Tab.json, N'$.Orders.OrdersArray')
        WITH (
          Number   varchar(200) N'$.Order.Number',
          Date     datetime     N'$.Order.Date',
          Customer varchar(200) N'$.AccountNumber',
          Quantity int          N'$.Item.Quantity'
        )
 AS SalesOrderJsonData
WHERE JSON_VALUE(Tab.json, '$.Status') = N'Closed'
ORDER BY JSON_VALUE(Tab.json, '$.Group'), Tab.DateModified
```

Both standard table columns and values from JSON text can be used in the same query. You can add indexes on the `JSON_VALUE(Tab.json, '$.Status')` expression to improve performance of query. For more info, see Index JSON data.+

Import JSON data into SQL Server tables

If you have to load JSON data from an external service into SQL Server, you can use **OPENJSON** to import the data into SQL Server instead of parsing the data in the application layer.

```
DECLARE @jsonVariable NVARCHAR(MAX)

SET @jsonVariable = N'[
     {
      "Order": {
       "Number":"SO43659",
       "Date":"2011-05-31T00:00:00"
      },
      "AccountNumber":"AW29825",
      "Item": {
       "Price":2024.9940,
       "Quantity":1
      }
```

```
    },
    {
     "Order": {
      "Number":"SO43661",
      "Date":"2011-06-01T00:00:00"
     },
     "AccountNumber":"AW73565",
     "Item": {
      "Price":2024.9940,
      "Quantity":3
     }
    }
]'

INSERT INTO SalesReport
SELECT SalesOrderJsonData.*
FROM OPENJSON (@jsonVariable, N'$.Orders.OrdersArray')
        WITH (
          Number   varchar(200) N'$.Order.Number',
          Date     datetime     N'$.Order.Date',
          Customer varchar(200) N'$.AccountNumber',
          Quantity int          N'$.Item.Quantity'
        )
 AS SalesOrderJsonData;
```

The content of the JSON variable can be provided by an external REST service, sent as a parameter from a client-side JavaScript framework, or loaded from external files. You can easily insert, update or merge results from JSON text into a SQL Server table. For more info about this scenario, see the following blog posts.+

- Importing JSON data in SQL Server
- Upsert JSON documents in SQL Server 2016
- Loading GeoJSON data into SQL Server 2016.

Load JSON files into SQL Server

Information stored in files can be formatted as standard JSON or Line-Delimited JSON. SQL Server can import the contents of JSON files, parse it using the **OPENJSON** or **JSON_VALUE** functions, and load it into tables. +

- If your JSON documents are stored in local files, on shared network drives, or in Azure File Storage locations that can be accessed by SQL Server, you can use bulk import to load your JSON data into SQL Server. For more info about this scenario, see Importing JSON files into SQL Server using OPENROWSET (BULK).
- If your line-delimited JSON files are stored in Azure Blob Storage or the Hadoop file system, you can use Polybase to load JSON text, parse it in Transact-SQL code, and load it into tables.

Test drive built-in JSON support

Test drive built-in JSON support with the AdventureWorks sample database. To get the AdventureWorks sample database, download at least the database file and the samples and scripts file from here. After you restore the sample database to an instance of SQL Server 2016, unzip the samples file and open the "JSON Sample Queries procedures views and indexes.sql" file from the JSON folder. Run the scripts in this file to reformat some existing data as JSON data, run sample queries and reports over the JSON data, index the JSON data, and import and export JSON.

Here's what you can do with the scripts included in the file.

1. Denormalize the existing schema to create columns of JSON data.
   a. Store information from SalesReasons, SalesOrderDetails, SalesPerson, Customer, and other tables that contain information related to sales order into JSON columns in the SalesOrder_json table.
   b. Store information from EmailAddresses/PersonPhone tables into the Person_json table as arrays of JSON objects.
2. Create procedures and views that query JSON data.
3. Index JSON data – create indexes on JSON properties and full-text indexes.
4. Import and export JSON – create and run procedures that export the content of the Person and the SalesOrder tables as JSON results, and import and update the Person and the SalesOrder tables using JSON input.
5. Run query examples – run some queries that call the stored procedures and views created in steps 2 and 4.

6. Clean up scripts – don't run this part if you want to keep the stored procedures and views created in steps 2 and 4.

References

1. JSON support in SQL Server 2016: https://www.simple-talk.com/sql/learn-sql-server/json-support-in-sql-server-2016/
2. JSON Data in SQL 2016: https://blogs.technet.microsoft.com/dataplatforminsider/2016/01/05/json-in-sql-server-2016-part-1-of-4/
3. JSON Data (SQL Server): https://docs.microsoft.com/en-us/sql/relational-databases/json/json-data-sql-server
4. AdventureWorks Databases and Scripts for SQL Server 2016 CTP3: https://www.microsoft.com/en-us/download/details.aspx?id=49502
5. Importing JSON files in SQL Server: https://blogs.msdn.microsoft.com/sqlserverstorageengine/2015/10/07/bulk-importing-json-files-into-sql-server/
6. Importing files using OPENROWSET(BULK): https://blogs.msdn.microsoft.com/sqlserverstorageengine/2015/10/07/bulk-importing-json-files-into-sql-server/