

# Import JSON data into SQL Server

 [sqlshack.com/import-json-data-into-sql-server](https://sqlshack.com/import-json-data-into-sql-server)

Rajendra Gupta

January 17, 2020

100% **free** SQL tools



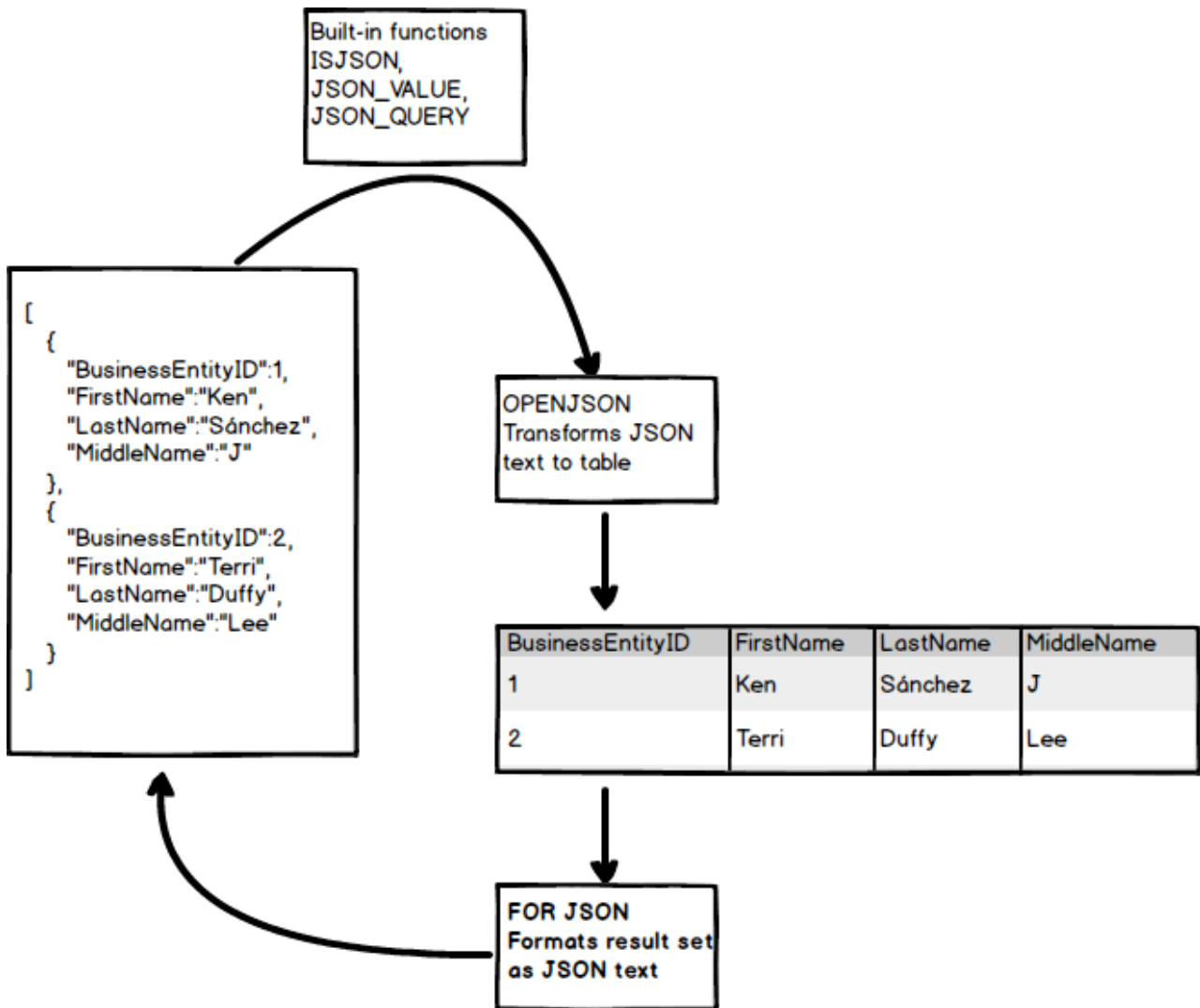
This article explores the process of JSON data import in the SQL Server table using T-SQL and SSIS.

## Introduction

Java Script Object Notation is an accessible data exchange format. Many applications support JSON format data nowadays. We can use JSON data for unstructured data such as log files and NoSQL databases. SQL Server also supports JSON format data import and export for exchanging data with different data sources and applications.

We can use SQL Server FOR XML PATH or FOR XML Auto clause in the SQL Server query, and it retrieves the results into the JSON format. We have explored this in the article [How to import/export JSON data using SQL Server 2016](#).

In the following image, we can see built-in functions for converting text into JSON and vice-versa:



## Azure Data Studio JSON format output

We can also save results in JSON format without specifying any parameter in Azure Data Studio. Let's execute a query in the **AdventureWorks** sample database in Azure Data Studio:

```
1 SELECT TOP (10) [FirstName],
2             [MiddleName],
3             [LastName],
4             [JobTitle],
5             [PhoneNumber],
6             [PhoneNumberType],
7             [EmailAddress],
8             [EmailPromotion]
9 FROM [AdventureWorks].[HumanResources].[vEmployee];
```

We get the output in the tabular format as shown below:

Run Cancel Disconnect Change Connection AdventureWorks Explain Enable SQLCMD

```

1 SELECT TOP (10) [FirstName],
2 [MiddleName],
3 [LastName],
4 [JobTitle],
5 [PhoneNumber],
6 [PhoneNumberType],
7 [EmailAddress],
8 [EmailPromotion]
9 FROM [AdventureWorks].[HumanResources].[vEmployee];

```

Results Messages

	FirstName	MiddleName	LastName	JobTitle	PhoneNumber	PhoneNumberType	EmailAddress	EmailPromotion
1	Ken	J	Sánchez	Chief Executive Officer	697-555-0142	Cell	ken@adventure-works.com	0
2	Terri	Lee	Duffy	Vice President of Engineer...	819-555-0175	Work	terri@adventure-works.com	1
3	Roberto	NULL	Tamburello	Engineering Manager	212-555-0187	Cell	roberto@adventure-works.c...	0
4	Rob	NULL	Walters	Senior Tool Designer	612-555-0100	Cell	rob@adventure-works.com	0
5	Gail	A	Erickson	Design Engineer	849-555-0139	Cell	gail@adventure-works.com	0
6	Jossef	H	Goldberg	Design Engineer	122-555-0189	Work	jossef@adventure-works.com	0
7	Dylan	A	Miller	Research and Development M...	181-555-0156	Work	dylan@adventure-works.com	2
8	Diane	L	Margheim	Research and Development E...	815-555-0138	Cell	diane@adventure-works.com	0
9	Gigi	N	Matthew	Research and Development E...	185-555-0186	Cell	gigi@adventure-works.com	0
10	Michael	NULL	Raheem	Research and Development M...	330-555-2568	Work	michael@adventure-works.c...	2

In the above screenshot, we have following output formats:

- Save as CSV
- Save as Excel
- Save as JSON
- Save as XML
- Chart
- Visualizer

Run Cancel Disconnect Change Connection AdventureWorks Explain Enable SQLCMD

```

1 SELECT TOP (10) [FirstName],
2 [MiddleName],
3 [LastName],
4 [JobTitle],
5 [PhoneNumber],
6 [PhoneNumberType],
7 [EmailAddress],
8 [EmailPromotion]
9 FROM [AdventureWorks].[HumanResources].[vEmployee];

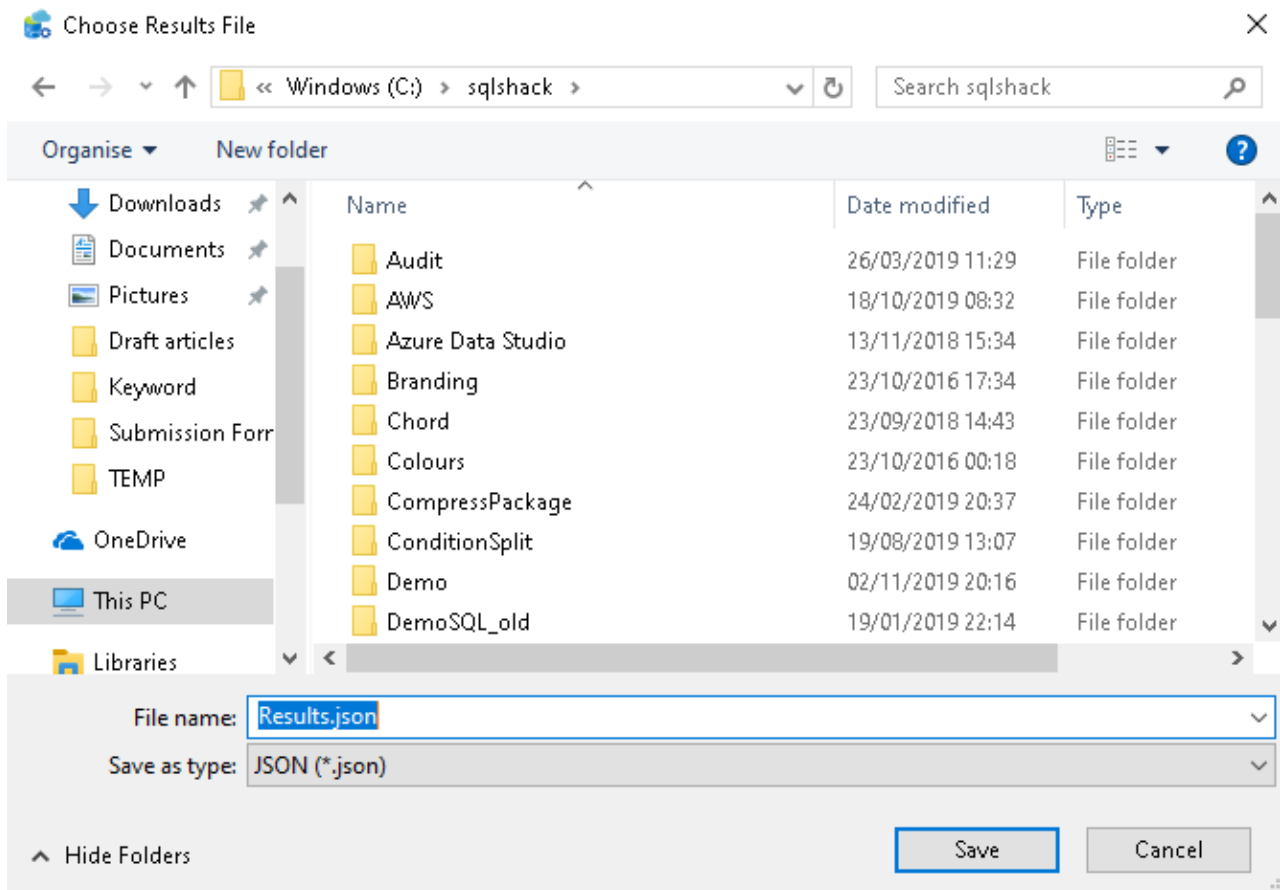
```

Results Messages

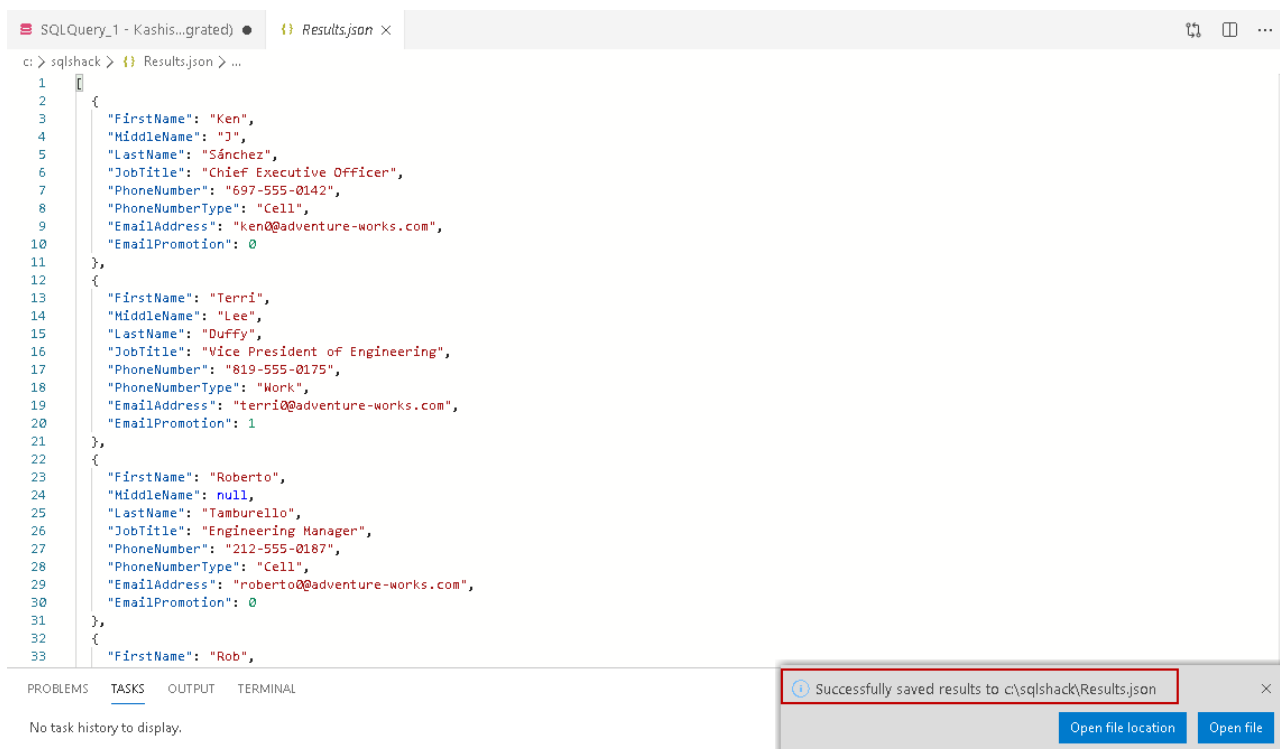
	FirstName	MiddleName	LastName	JobTitle	PhoneNumber	PhoneNumberType	EmailAddress	EmailPromotion
1	Ken	J	Sánchez	Chief Executive Officer	697-555-0142	Cell	ken@adventure-works.com	0
2	Terri	Lee	Duffy	Vice President of Engineer...	819-555-0175	Work	terri@adventure-works.com	1
3	Roberto	NULL	Tamburello	Engineering Manager	212-555-0187	Cell	roberto@adventure-works.c...	0
4	Rob	NULL	Walters	Senior Tool Designer	612-555-0100	Cell	rob@adventure-works.com	0
5	Gail	A	Erickson	Design Engineer	849-555-0139	Cell	gail@adventure-works.com	0
6	Jossef	H	Goldberg	Design Engineer	122-555-0189	Work	jossef@adventure-works.com	0
7	Dylan	A	Miller	Research and Development M...	181-555-0156	Work	dylan@adventure-works.com	2
8	Diane	L	Margheim	Research and Development E...	815-555-0138	Cell	diane@adventure-works.com	0
9	Gigi	N	Matthew	Research and Development E...	185-555-0186	Cell	gigi@adventure-works.com	0
10	Michael	NULL	Raheem	Research and Development M...	330-555-2568	Work	michael@adventure-works.c...	2

Save as JSON

Click on **Save as JSON**, and it asks you for specifying the directory and file name for JSON file:



Click on **Save**, and it shows the JSON format output:



Let's focus on a single row output. In the output, we can see key-value pair for each:

```
c: > sqlshack > {} Results.json > ...
1  [
2    {
3      "FirstName": "Ken",
4      "MiddleName": "J",
5      "LastName": "Sánchez",
6      "JobTitle": "Chief Executive Officer",
7      "PhoneNumber": "697-555-0142",
8      "PhoneNumberType": "Cell",
9      "EmailAddress": "ken0@adventure-works.com",
10     "EmailPromotion": 0
11   },
```

Now, we will take the opposite approach.

## JSON data import in SQL Server

We require JSON data import into the SQL Server table from the .json file created earlier.

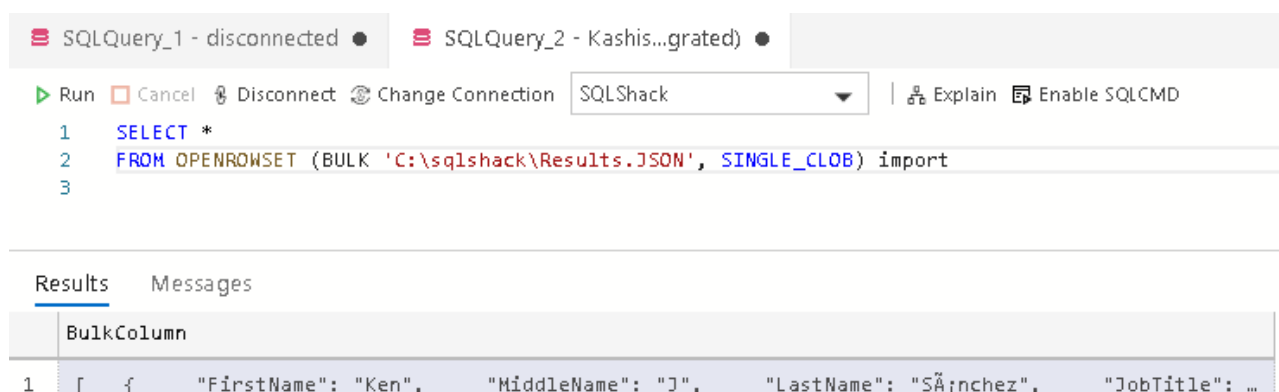
### Step 1: Import file using OPENROWSET

The first step is to load the JSON file content in a table. We can use the table value function **OPENROWSET** for reading data from a file and return a table in the output. This table contains a single column and loads entire file data into it.

Specify the complete file path in the OPENROWSET function:

```
1  SELECT *
2  FROM OPENROWSET (BULK 'C:\sqlshack\Results.JSON', SINGLE_CLOB) as
   import
```

It returns the JSON file output and contains the column **BulkColumn** as shown below:



The screenshot shows the SQL Server Enterprise Manager interface. At the top, there are two query windows: 'SQLQuery\_1 - disconnected' and 'SQLQuery\_2 - Kashis...grated'. Below them is a toolbar with buttons for 'Run', 'Cancel', 'Disconnect', 'Change Connection', and a dropdown menu set to 'SQLShack'. To the right of the toolbar are buttons for 'Explain' and 'Enable SQLCMD'. The query editor shows the following SQL code:

```
1  SELECT *
2  FROM OPENROWSET (BULK 'C:\sqlshack\Results.JSON', SINGLE_CLOB) import
3
```

Below the query editor, there are two tabs: 'Results' and 'Messages'. The 'Results' tab is active, showing a table with one column named 'BulkColumn'. The table contains one row of data, which is a JSON object:

	BulkColumn
1	[ { "FirstName": "Ken", "MiddleName": "J", "LastName": "Sánchez", "JobTitle": ...

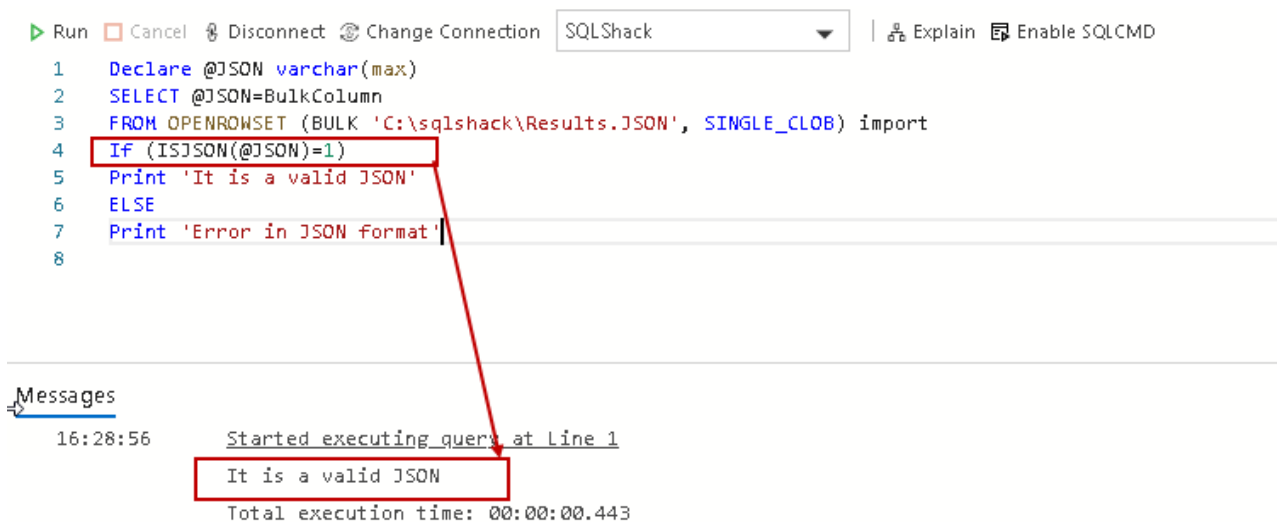
Starting from SQL Server 2016, we have a function **ISJSON** for validating the JSON format. It checks the JSON stored in the @JSON having data type Varchar(max).

In the following query, the @JSON variable reads the BulkColumn from the imported file, and further, we pass this variable into the function ISJSON() for checking the

syntax. It returns value 1 if JSON is a valid structure:

```
1 Declare @JSON varchar(max)
2 SELECT @JSON=BulkColumn
3 FROM OPENROWSET (BULK 'C:\sqlshack\Results.JSON', SINGLE_CLOB)
4 import
5 If (ISJSON(@JSON)=1)
6 Print 'It is a valid JSON'
7 ELSE
  Print 'Error in JSON format'
```

In the following screenshot, verify that the JSON structure is valid:



## Step 2: Convert JSON output from the variable into SQL Server tables

We use **OPENJSON()** function for converting the JSON output from a variable into a tabular format. We get the output in different rows and columns, and later, it can be inserted into SQL Server tables.

Let's run the following command without specifying any additional parameter. By default, it returns all key-value pairs at the first level in the output:

```
1 Declare @JSON varchar(max)
2 SELECT @JSON=BulkColumn
3 FROM OPENROWSET (BULK 'C:\sqlshack\Results.JSON', SINGLE_CLOB)
4 import
5 SELECT *
  FROM OPENJSON (@JSON)
```

Run	Cancel	Disconnect	Change Connection	SQLShack	Explain	Enable SQLCMD
1	Declare @JSON varchar(max)					
2	SELECT @JSON=BulkColumn					
3	FROM OPENROWSET (BULK 'C:\sqlshack\Results.JSON', SINGLE_CLOB) import					
4	SELECT *					
5	FROM OPENJSON (@JSON)					

Results		Messages	
	key	value	type
1	0	{ "FirstName": "Ken", "MiddleName": "J", "LastName": "S��nchez", ...	5
2	1	{ "FirstName": "Terri", "MiddleName": "Lee", "LastName": "Duffy", ...	5
3	2	{ "FirstName": "Roberto", "MiddleName": null, "LastName": "Tamburello"...	5
4	3	{ "FirstName": "Rob", "MiddleName": null, "LastName": "Walters", ...	5
5	4	{ "FirstName": "Gail", "MiddleName": "A", "LastName": "Erickson", ...	5
6	5	{ "FirstName": "Jossef", "MiddleName": "H", "LastName": "Goldberg", ...	5
7	6	{ "FirstName": "Dylan", "MiddleName": "A", "LastName": "Miller", ...	5
8	7	{ "FirstName": "Diane", "MiddleName": "L", "LastName": "Margheim", ...	5
9	8	{ "FirstName": "Gigi", "MiddleName": "N", "LastName": "Matthew", ...	5
10	9	{ "FirstName": "Michael", "MiddleName": null, "LastName": "Raheem", ...	5

In this output, we can see the following columns:

- **Key:** We can consider it as a row number in a table. In JSON format, it is a combination of key-value pair enclosed in the curly brackets {}
- **Value:** It is the property value. In this case, we can see it is a combination of the column name and its value
- **Type:** It is the type of object such as String, Boolean, number, array or object

We want the output like a query output in a tabular format. We can use the WITH clause in the above query along with the column definition. It is like defining a table with appropriate column names and their data types. Once we use a WITH clause, OPENJSON reads the JSON object array and converts the individual value as per specified data type and prints in the output.

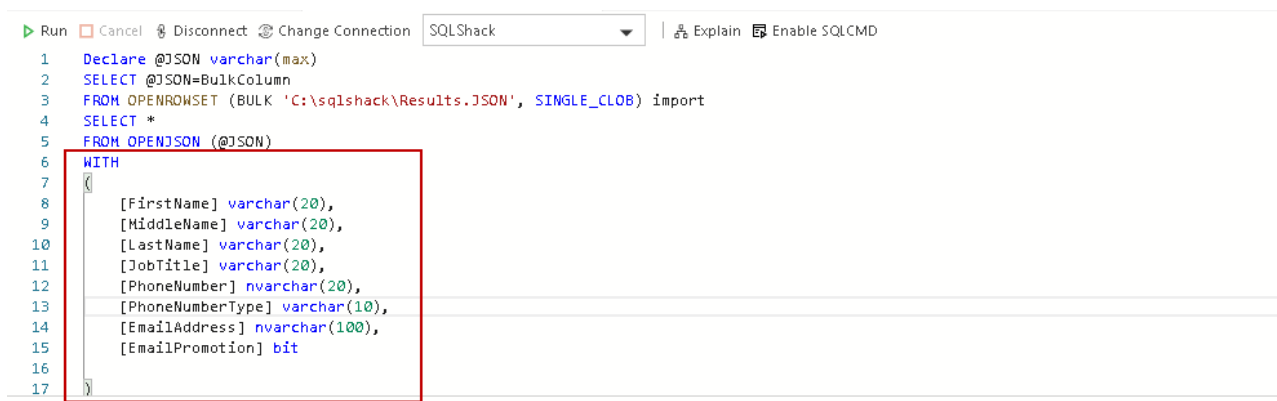
In the following query, we specified all output columns with appropriate data types:

```

1  Declare @JSON varchar(max)
2  SELECT @JSON=BulkColumn
3  FROM OPENROWSET (BULK 'C:\sqlshack\Results.JSON', SINGLE_CLOB)
4  import
5  SELECT *
6  FROM OPENJSON (@JSON)
7  WITH
8  (
9      [FirstName] varchar(20),
10     [MiddleName] varchar(20),
11     [LastName] varchar(20),
12     [JobTitle] varchar(20),
13     [PhoneNumber] nvarchar(20),
14     [PhoneNumberType] varchar(10),
15     [EmailAddress] nvarchar(100),
16     [EmailPromotion] bit
17 )

```

In this output, we get the results similar to query output we get earlier in this article:



The screenshot shows a SQL query in the Enterprise Manager interface. The query is as follows:

```

1  Declare @JSON varchar(max)
2  SELECT @JSON=BulkColumn
3  FROM OPENROWSET (BULK 'C:\sqlshack\Results.JSON', SINGLE_CLOB) import
4  SELECT *
5  FROM OPENJSON (@JSON)
6  WITH
7  (
8      [FirstName] varchar(20),
9      [MiddleName] varchar(20),
10     [LastName] varchar(20),
11     [JobTitle] varchar(20),
12     [PhoneNumber] nvarchar(20),
13     [PhoneNumberType] varchar(10),
14     [EmailAddress] nvarchar(100),
15     [EmailPromotion] bit
16 )

```

The results of the query are displayed in a table with the following columns: FirstName, MiddleName, LastName, JobTitle, PhoneNumber, PhoneNumberType, EmailAddress, and EmailPromotion.

	FirstName	MiddleName	LastName	JobTitle	PhoneNumber	PhoneNumberType	EmailAddress	EmailPromotion
1	Ken	J	Sánchez	Chief Executive Offi	697-555-0142	Cell	ken0@adventure-works.com	0
2	Terri	Lee	Duffy	Vice President of En	819-555-0175	Work	terri0@adventure-works.com	1
3	Roberto	NULL	Tamburello	Engineering Manager	212-555-0187	Cell	roberto0@adventure-works.c...	0
4	Rob	NULL	Walters	Senior Tool Designer	612-555-0100	Cell	rob0@adventure-works.com	0
5	Gail	A	Erickson	Design Engineer	849-555-0139	Cell	gail0@adventure-works.com	0
6	Jossef	H	Goldberg	Design Engineer	122-555-0189	Work	jossef0@adventure-works.com	0
7	Dylan	A	Miller	Research and Develop	181-555-0156	Work	dylan0@adventure-works.com	1
8	Diane	L	Margheim	Research and Develop	815-555-0138	Cell	diane1@adventure-works.com	0
9	Gigi	N	Matthew	Research and Develop	185-555-0186	Cell	gigi0@adventure-works.com	0
10	Michael	NULL	Raheem	Research and Develop	330-555-2568	Work	michael6@adventure-works.c...	1

Ideally, we should specify the similar data type that we use in the table else it might cause issues due to data type conversion errors:



```

1  Declare @JSON varchar(max)
2  SELECT @JSON=BulkColumn
3  FROM OPENROWSET (BULK 'C:\sqlshack\Results.JSON', SINGLE_CLOB) import
4  SELECT *
5  FROM OPENJSON (@JSON)
6  WITH
7  (
8      [FirstName] varchar(20),
9      [MiddleName] varchar(20),
10     [LastName] varchar(20),
11     [JobTitle] int,
12     [PhoneNumber] nvarchar(20),
13     [PhoneNumberType] varchar(10),
14     [EmailAddress] nvarchar(100),
15     [EmailPromotion] bit
16 )
17

```

Results Messages

17:00:11 Started executing query at Line 1

Msg 245, Level 16, State 1, Line 4  
Conversion failed when converting the nvarchar value 'Chief Executive Officer' to data type int.

Total execution time: 00:00:00.020

We can skip any columns as well in the WITH clause. For example, let's say we do not want **EmailPromotion** column in the output; however, it is available in the JSON file. If we do not specify any data type in the WITH clause, SQL Server removes this column from the output:

```

1  Declare @JSON varchar(max)
2  SELECT @JSON=BulkColumn
3  FROM OPENROWSET (BULK 'C:\sqlshack\Results.JSON', SINGLE_CLOB)
4  import
5  SELECT *
6  FROM OPENJSON (@JSON)
7  WITH
8  (
9      [FirstName] varchar(20),
10     [MiddleName] varchar(20),
11     [LastName] varchar(20),
12     [JobTitle] varchar(20),
13     [PhoneNumber] nvarchar(20),
14     [PhoneNumberType] varchar(10),
15     [EmailAddress] nvarchar(100)
16 )

```

In this output, we do not see the **EmailPromotion** column even if it is available in the JSON:

```

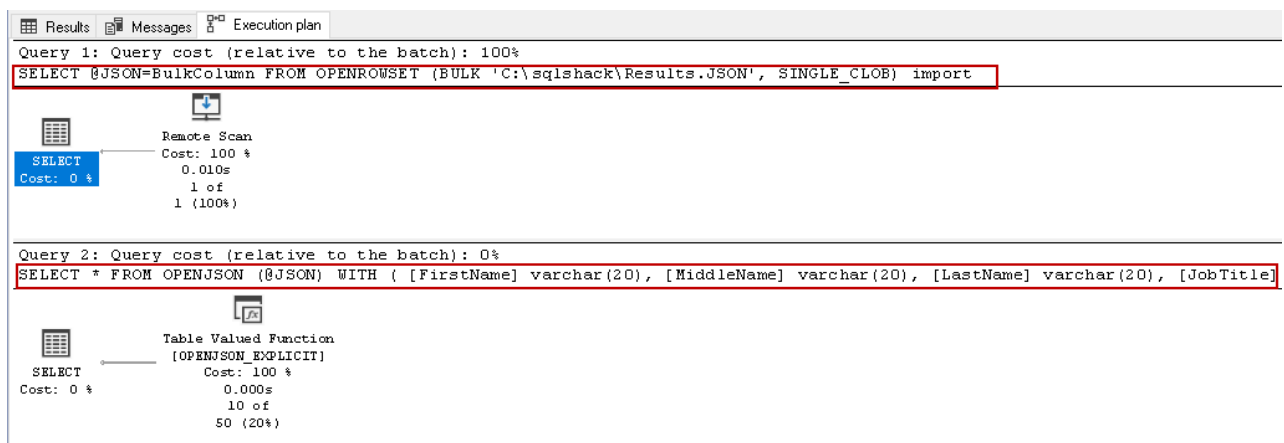
Run Cancel Disconnect Change Connection SQLShack Explain Enable SQLCMD
1 Declare @JSON varchar(max)
2 SELECT @JSON=BulkColumn
3 FROM OPENROWSET (BULK 'C:\sqlshack\Results.JSON', SINGLE_CLOB) import
4 SELECT *
5 FROM OPENJSON (@JSON)
6 WITH
7 (
8     [FirstName] varchar(20),
9     [MiddleName] varchar(20),
10    [LastName] varchar(20),
11    [JobTitle] varchar(20),
12    [PhoneNumber] nvarchar(20),
13    [PhoneNumberType] varchar(10),
14    [EmailAddress] nvarchar(100)
15 )

```

	FirstName	MiddleName	LastName	JobTitle	PhoneNumber	PhoneNumberType	EmailAddress
1	Ken	J	Sánchez	Chief Executive Offi	697-555-0142	Cell	ken0@adventure-works.com
2	Terri	Lee	Duffy	Vice President of En	819-555-0175	Work	terri0@adventure-works.com
3	Roberto	NULL	Tamburello	Engineering Manager	212-555-0187	Cell	roberto0@adventure-works.c...
4	Rob	NULL	Walters	Senior Tool Designer	612-555-0100	Cell	rob0@adventure-works.com
5	Gail	A	Erickson	Design Engineer	849-555-0139	Cell	gail0@adventure-works.com
6	Jossef	H	Goldberg	Design Engineer	122-555-0189	Work	jossef0@adventure-works.com
7	Dylan	A	Miller	Research and Develop	181-555-0156	Work	dylan0@adventure-works.com
8	Diane	L	Margheim	Research and Develop	815-555-0138	Cell	diane1@adventure-works.com
9	Gigi	N	Matthew	Research and Develop	185-555-0186	Cell	gigi0@adventure-works.com
10	Michael	NULL	Raheem	Research and Develop	330-555-2568	Work	michael16@adventure-works.c...

## View actual execution plan for JSON data import

In SSMS, click on **Actual Execution Plan** and execute the previous SQL statement. It gives the following plan:



In this plan, we can note that it uses a table-valued function for OPENJSON. Let's hover the cursor over it and view the details:

Query 1: Query cost (re:)  
SELECT @JSON=BulkColumn

Query 2: Query cost (re:)  
SELECT \* FROM OPENJSON

Table Valued Function

Table valued function.

Physical Operation	Table Valued Function
Logical Operation	Table Valued Function
Actual Execution Mode	Row
Estimated Execution Mode	Row
Actual Number of Rows	10
Actual Number of Batches	0
Estimated Operator Cost	0.0000502 (100%)
Estimated I/O Cost	0
Estimated CPU Cost	0.0000502
Estimated Subtree Cost	0.0000502
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows	50
Estimated Row Size	189 B
Actual Rebinds	1
Actual Rewinds	0
Node ID	0

Object  
[OPENJSON\_EXPLICIT]

Output List  
[OPENJSON\_EXPLICIT].FirstName,  
[OPENJSON\_EXPLICIT].MiddleName,  
[OPENJSON\_EXPLICIT].LastName,  
[OPENJSON\_EXPLICIT].JobTitle,  
[OPENJSON\_EXPLICIT].PhoneNumber,  
[OPENJSON\_EXPLICIT].PhoneNumberType,  
[OPENJSON\_EXPLICIT].EmailAddress,  
[OPENJSON\_EXPLICIT].EmailPromotion

Query executed successfully.

In the above execution plan, note the following:

- It uses object [OPENJSON\_EXPLICIT] for fetching the columns and their values
- We see a difference between the estimated number of rows (50) and an actual number of rows (10) as query optimizer unable to estimates the correct number of rows for JSON input

Let's import another JSON file having 290 records and view the actual execution plan. In this execution plan also, it estimates the number of rows 50 in comparison with the actual 290 rows. It seems SQL Server uses a fixed estimate of 50 rows for JSON data import:

Results Messages Execution plan

Query 1: Query cost (relative to query plan) 1.0000000  
 SELECT @JSON=BulkColumn FROM OPENJSON (@JSON, '\$.BulkColumn') WITH (BulkColumn NVARCHAR(100))

Remote Scan  
 Cost: 100 %  
 0.085s  
 1 of 1 (100%)

Query 2: Query cost (relative to query plan) 1.0000000  
 SELECT \* FROM OPENJSON (@JSON, '\$.BulkColumn') WITH (BulkColumn NVARCHAR(100))

Table Valued Function  
 Cost: 100 %  
 0.004s  
 290 of 50 (580%)

Table Valued Function  
 Table valued function.

Physical Operation	Table Valued Function
Logical Operation	Table Valued Function
Actual Execution Mode	Row
Estimated Execution Mode	Row
Actual Number of Rows	290
Actual Number of Batches	0
Estimated Operator Cost	0.0000502 (100%)
Estimated I/O Cost	0
Estimated CPU Cost	0.0000502
Estimated Subtree Cost	0.0000502
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows	50
Estimated Row Size	189 B
Actual Rebinds	1
Actual Rewinds	0
Node ID	0

Object  
 [OPENJSON\_EXPLICIT]

Output List  
 [OPENJSON\_EXPLICIT].FirstName,  
 [OPENJSON\_EXPLICIT].MiddleName,  
 [OPENJSON\_EXPLICIT].LastName,  
 [OPENJSON\_EXPLICIT].JobTitle,  
 [OPENJSON\_EXPLICIT].PhoneNumber,  
 [OPENJSON\_EXPLICIT].PhoneNumberType,  
 [OPENJSON\_EXPLICIT].EmailAddress,  
 [OPENJSON\_EXPLICIT].EmailPromotion

Query executed successfully.

Ready

## Import JSON file data into the SQL Server table

We can use the similar t-SQL described earlier for inserting data in an existing table. We can also create a table using the SQL SELECT INTO statement similar to a relational DB table. The following query, imports data from results.json, creates a JSON table automatically using SQL SELECT INTO statement and inserts data into it:

```

1  Declare @JSON varchar(max)
2  SELECT @JSON=BulkColumn
3  FROM OPENROWSET (BULK 'C:\sqlshack\Results.JSON', SINGLE_CLOB)
4  import
5  SELECT * INTO JSONTable
6  FROM OPENJSON (@JSON)
7  WITH
8  (
9      [FirstName] varchar(20),
10     [MiddleName] varchar(20),
11     [LastName] varchar(20),
12     [JobTitle] varchar(20),
13     [PhoneNumber] nvarchar(20),
14     [PhoneNumberType] varchar(10),
15     [EmailAddress] nvarchar(100)
16 )

```

We can verify the records using the following SELECT statement:

```

1  Select * from JSONTable

```

□

## SSIS package for importing JSON data into SQL Server table

---

We can use SQL Server integration package for inserting JSON into the SQL Server table as well. You should have basic knowledge of SSIS packages. If you are not much aware of it, I would recommend reading [SQLShack](#) articles.

I am using Visual Studio 2019 with SSDT package in this article.

Open Visual Studio 2019 and create a new integration service project in a suitable directory. In this package, drag Data Flow Task in the control flow task and rename it to **Data import from the JSON file**:

□

Double-click on this data import from the JSON file task and it moves the package in the data flow tab. In the data flow tab, drag the OLE DB Source and OLE DB destination as shown below:

□

Rename the source and destination as per below:

- OLE DB Source -> JSON File data
- OLE DB Destination -> SQL Server table

□

Double-click on the JSON file data and in the OLE DB source editor, do the following tasks:

1. **OLE DB connection Manager:** Specify SQL Server instance details in the connection manager
2. **Data access mode:** We will use the SQL query specified above section. From the drop-down menu, select the data access mode as **SQL Command**
3. **SQL Command Text:** In this section, paste the query we specified earlier for JSON data import. Let me paste the query again for easiness

```
1  Declare @JSON varchar(max)
2  SELECT @JSON=BulkColumn
3  FROM OPENROWSET (BULK 'C:\sqlshack\Results.JSON',
4  SINGLE_CLOB) import
5  SELECT * FROM OPENJSON (@JSON)
6  WITH
7  (
8      [FirstName] varchar(20),
9      [MiddleName] varchar(20),
10     [LastName] varchar(20),
11     [JobTitle] varchar(20),
12     [PhoneNumber] nvarchar(20),
13     [PhoneNumberType] varchar(10),
14     [EmailAddress] nvarchar(100),
15     [EmailPromotion] bit
16 )
```

□

Click on **Columns** and verify the source columns that we want to populate in the destination table:

□

Click **OK**, and our configuration for OLE DB source is complete:

□

Now configure SQL Server table for the destination table in the SQL instance:

□

Click on **Mappings** and verify the source, destination column mappings:

□

Click **OK**, and our SSIS package configuration is complete. You do not see any error icon on any task:

□

Click on **Start**. It quickly processes the JSON file and inserts data into the SQL table. From the output, we see it transferred 10 rows from the JSON file to SQL table:

□

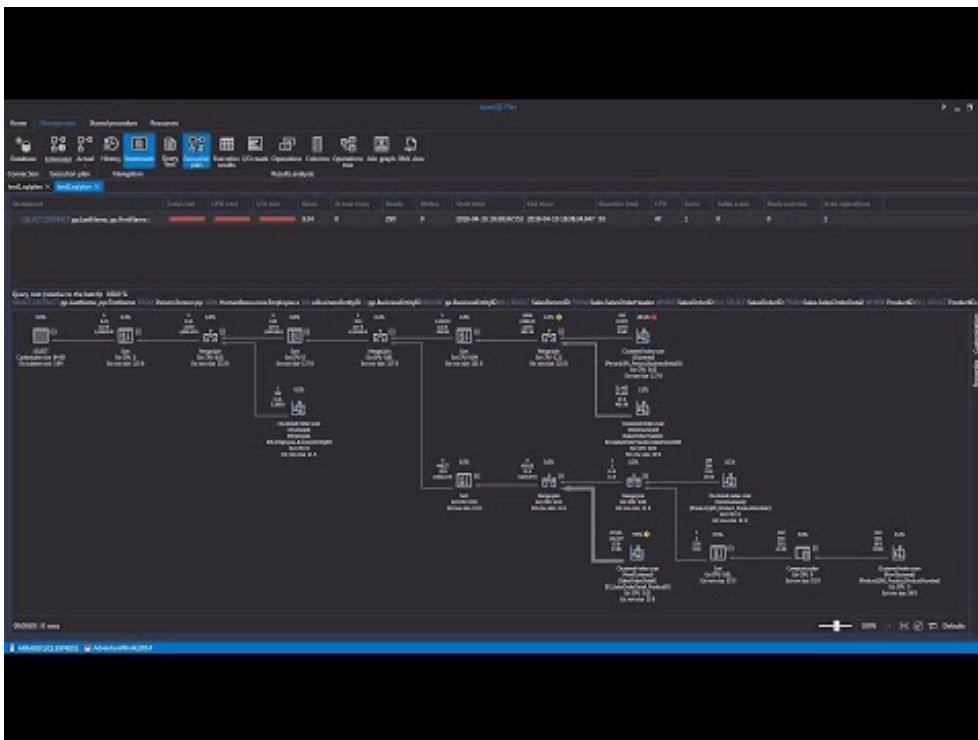
Now, view the records using a SELECT statement:

## Conclusion

In this article, we explored importing JSON file into SQL Server tables using OPENROWSET and OPENJSON function. We also used an SSIS package for JSON data import. You should practice this as per your data requirements.

## See more

Check out ApexSQL Plan, to [view SQL execution plans](#) for free, including comparing plans, stored procedure performance profiling, missing index details, lazy profiling, wait times, plan execution history and more



Watch Video At: [https://youtu.be/Po1\\_Zd11HXo](https://youtu.be/Po1_Zd11HXo)

**FREE SQL query plan analysis and optimization**



## Rajendra Gupta

Rajendra has 8+ years of experience in database administration having a passion for database performance optimization, monitoring, and high availability and disaster recovery technologies, learning new things, new features.

While working as a Senior consultant DBA for big customers and having certified with MCSA SQL 2012, he likes to share knowledge on various blogs.

He can be reached at  
[rajendra.gupta16@gmail.com](mailto:rajendra.gupta16@gmail.com)

[View all posts by Rajendra Gupta](#)

