DOCUMENTO EJEMPLOS DE INSERTAR Y SELECT DATOS DESDE XML

Xml inserter sqlserver

Ejemplo xml formato dataset.

DECLARE @input XML = '<dataset>

<metadata>

<item name="NAME\_LAST" type="xs:string" length="62" />

<item name="NAME\_FIRST" type="xs:string" length="62" />

<item name="NAME\_MIDDLE" type="xs:string" length="32" />

</metadata>

<data>

<row>

<value>SMITH</value>

<value>MARY</value>

<value>N</value>

</row>

<row>

<value>SMITH2</value>

<value>MARY2</value>

<value>N2</value>

</row>

</data>

</dataset>'

INSERT INTO dbo.YourTable(ColName, ColFirstName, ColOther)

SELECT

Name = XCol.value('(value)[1]','varchar(25)'),

FirstName = XCol.value('(value)[2]','varchar(25)'),

OtherValue = XCol.value('(value)[3]','varchar(25)')

FROM

@input.nodes('/dataset/data/row') AS XTbl(XCol)

**Ejemplo III**( xml insertar paises)

declare @XML AS XML

SET @XML='<root> <row><pki\_idpais>6</pki\_idpais><atc\_despais>ARGENTINA</atc\_despais><fki\_idcontinent>1</fki\_idcontinent></row> <row><pki\_idpais>7</pki\_idpais><atc\_despais>CHILE</atc\_despais><fki\_idcontinent>1</fki\_idcontinent></row> <row><pki\_idpais>8</pki\_idpais><atc\_despais>COLOMBIA</atc\_despais><fki\_idcontinent>1</fki\_idcontinent></row> <row><pki\_idpais>9</pki\_idpais><atc\_despais>VENEZUELA</atc\_despais><fki\_idcontinent>1</fki\_idcontinent></row> </root>'

insert into dbo.Dbtb\_Pais (pki\_idpais,atc\_despais,fki\_idcontinent)

select distinct

'pki\_idpais'=x.v.value('pki\_idpais[1]','INT'),

'atc\_despais'=x.v.value('atc\_despais[1]','VARCHAR(50)'),

'fki\_idcontinent'=x.v.value('fki\_idcontinent[1]','INT')

FROM @XML.nodes('/root/row') x(v))

**EJEMPLOS III VISUALIZA LOS DATOS DE LA TABLA PAIS A XML (FOR XML PATH)**

SELECT [pki\_idpais]

,[atc\_despais]

,[fki\_idcontinent]

,[atb\_statuspais]

FROM [JVCIDBSYSTEM].[dbo].[Dbtb\_Pais]

FOR XML PATH;

**RESULTADO**

<row>

<pki\_idpais>1</pki\_idpais>

<atc\_despais>Republica Dominicana</atc\_despais>

<fki\_idcontinent>1</fki\_idcontinent>

<atb\_statuspais>0</atb\_statuspais>

</row>

<row>

<pki\_idpais>2</pki\_idpais>

<atc\_despais>Japon</atc\_despais>

<fki\_idcontinent>2</fki\_idcontinent>

<atb\_statuspais>0</atb\_statuspais>

</row>

2-)( FOR XML RAW, ELEMENTS)

SELECT [pki\_idpais]

,[atc\_despais]

,[fki\_idcontinent]

,[atb\_statuspais]

FROM [JVCIDBSYSTEM].[dbo].[Dbtb\_Pais]

FOR XML RAW, ELEMENTS;

**RESULTADO**

<row>

<pki\_idpais>1</pki\_idpais>

<atc\_despais>Republica Dominicana</atc\_despais>

<fki\_idcontinent>1</fki\_idcontinent>

<atb\_statuspais>0</atb\_statuspais>

</row>

<row>

<pki\_idpais>2</pki\_idpais>

<atc\_despais>Japon</atc\_despais>

<fki\_idcontinent>2</fki\_idcontinent>

<atb\_statuspais>0</atb\_statuspais>

</row>

**CAMBIAR LA FILA row por Cambio**

SELECT [pki\_idpais]

,[atc\_despais]

,[fki\_idcontinent]

,[atb\_statuspais]

FROM [JVCIDBSYSTEM].[dbo].[Dbtb\_Pais]

FOR XML PATH('Cambio');

**RESULTADO**

<Cambio>

<pki\_idpais>1</pki\_idpais>

<atc\_despais>Republica Dominicana</atc\_despais>

<fki\_idcontinent>1</fki\_idcontinent>

<atb\_statuspais>0</atb\_statuspais>

</Cambio>

<Cambio>

<pki\_idpais>2</pki\_idpais>

<atc\_despais>Japon</atc\_despais>

<fki\_idcontinent>2</fki\_idcontinent>

<atb\_statuspais>0</atb\_statuspais>

</Cambio>

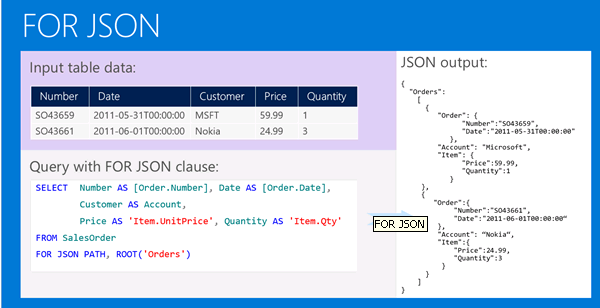
**Dar formato JSON a los resultados de consulta con FOR JSON (SQL Server)**

**ESTE TEMA SE APLICA A:** SQL Server (a partir de 2016)Az

ure SQL DatabaseAzure SQL Data Warehouse Almacenamiento de datos paralelos

Dé formato JSON a los resultados de las consultas o exporte datos de SQL Server como JSON mediante la adición de la cláusula **FOR JSON** a una instrucción **SELECT**. Use la **FOR JSON** cláusula para simplificar las aplicaciones cliente delegando el formato de salida JSON desde las aplicaciones cliente para SQL Server.

Cuando se usa la cláusula **FOR JSON** , puede especificar la estructura de la salida de forma explícita o dejar que la estructura de la instrucción SELECT determine la salida.

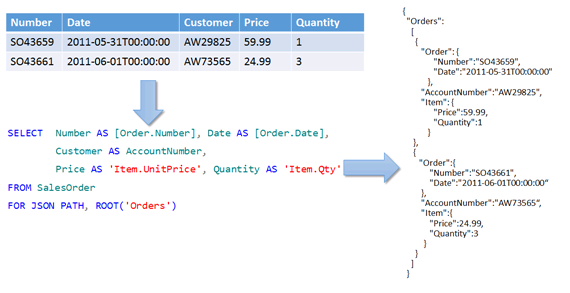
* Use **FOR JSON PATH** para mantener el control total sobre el formato de la salida JSON. Puede crear objetos contenedores y anidar propiedades complejas.
* Use **FOR JSON AUTO** dar formato JSON a la salida automáticamente en función de la estructura de la instrucción SELECT.

Este es un ejemplo de una instrucción **SELECT** con la cláusula **FOR JSON** y su salida.

Mantener el control sobre la salida JSON con FOR JSON PATH

En el modo **PATH** , puede usar la sintaxis de puntos (por ejemplo, 'Item.Price' ) para dar formato a la salida anidada.

Esta es una consulta de ejemplo en la que se usa el modo **PATH** con la cláusula **FOR JSON** . En el ejemplo siguiente también se usa la opción **ROOT** para especificar un elemento raíz con nombre.



Más información

Para obtener más información y ejemplos, vea [formato de salida JSON anidada con el modo PATH ( SQL Server )](https://docs.microsoft.com/es-es/sql/relational-databases/json/format-nested-json-output-with-path-mode-sql-server).

Para ver la sintaxis y el uso, consulte [Cláusula FOR (Transact-SQL)](https://docs.microsoft.com/es-es/sql/t-sql/queries/select-for-clause-transact-sql).

Permitir que el control de la instrucción SELECT del resultado JSON con FOR JSON AUTO

En el modo **AUTO** , la estructura de la instrucción SELECT determina el formato de la salida JSON. Los valores **null** no se incluyen en la salida de forma predeterminada. Puede usar la opción **INCLUDE\_NULL\_VALUES** para cambiar este comportamiento.

Esta es una consulta de ejemplo en la que se usa el modo **AUTO** con la cláusula **FOR JSON** .

**Consulta:**

SQLCopiar

SELECT name, surname

FROM emp

FOR JSON AUTO

**Resultado**

JSONCopiar

[{

"name": "John"

}, {

"name": "Jane",

"surname": "Doe"

}]

Más información

Para obtener más información y ejemplos, vea [formato de salida JSON automáticamente con el modo AUTO ( SQL Server )](https://docs.microsoft.com/es-es/sql/relational-databases/json/format-json-output-automatically-with-auto-mode-sql-server).

Para ver la sintaxis y el uso, consulte [Cláusula FOR (Transact-SQL)](https://docs.microsoft.com/es-es/sql/t-sql/queries/select-for-clause-transact-sql).

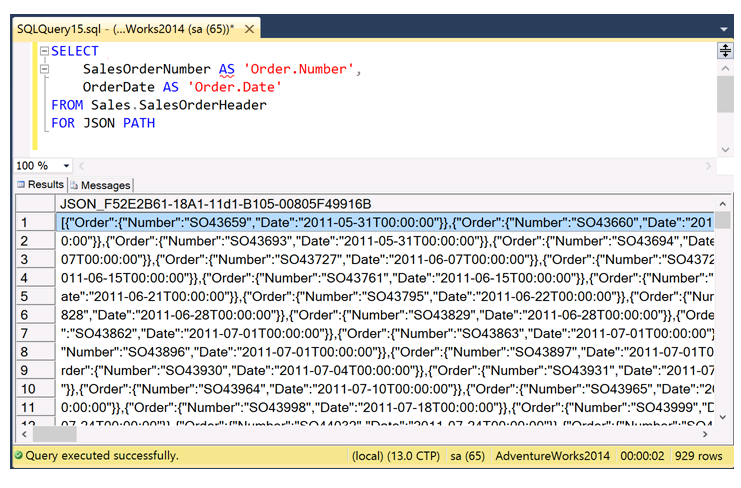
**Controlar otras opciones de salida JSON**

Controlar la salida de la **FOR JSON** cláusula mediante el uso de las siguientes opciones adicionales.

* **RAÍZ**. Para agregar un solo elemento de nivel superior a la salida JSON, especifique la opción **ROOT** . Si no especifica esta opción, la salida JSON no tiene un elemento raíz. Para obtener más información, vea [Agregar un nodo raíz a la salida JSON con la opción ROOT (SQL Server)](https://docs.microsoft.com/es-es/sql/relational-databases/json/add-a-root-node-to-json-output-with-the-root-option-sql-server).
* **INCLUDE\_NULL\_VALUES**. Para incluir valores NULL en la salida JSON, especifique la opción **INCLUDE\_NULL\_VALUES** . Si no especifica esta opción, la salida no incluye las propiedades JSON para los valores NULL en los resultados de la consulta. Para obtener más información, consulte [incluir valores Null en la salida de JSON con la opción INCLUDE\_NULL\_VALUES ( SQL Server )](https://docs.microsoft.com/es-es/sql/relational-databases/json/include-null-values-in-json-include-null-values-option).
* **WITHOUT\_ARRAY\_WRAPPER**. Para quitar los corchetes que rodean la salida JSON de la cláusula **FOR JSON** de manera predeterminada, especifique la opción **WITHOUT\_ARRAY\_WRAPPER** . Utilice esta opción para generar un objeto JSON único como salida de un sola fila de resultados. Si no especifica esta opción, la salida JSON es un formato como una matriz, es decir, está delimitado por corchetes. Para obtener más información, vea [Quitar corchetes de la salida JSON con la opción WITHOUT\_ARRAY\_WRAPPER (SQL Server)](https://docs.microsoft.com/es-es/sql/relational-databases/json/remove-square-brackets-from-json-without-array-wrapper-option).

**Salida de la cláusula FOR JSON**

La salida de la cláusula **FOR JSON** tiene las siguientes características.

1. El conjunto de resultados contiene una sola columna.
   * Un conjunto de resultados pequeño puede contener una sola fila.
   * Un conjunto de resultados grande divide la cadena JSON larga en varias filas.
     + De forma predeterminada, SQL Server Management Studio (SSMS) concatena los resultados en una sola fila cuando el valor de salida es **resultados a cuadrícula**. La barra de estado SSMS muestra el recuento de filas real.
     + Otras aplicaciones cliente pueden requerir código para concatenar los resultados extensa combinando el contenido de varias filas. Para obtener un ejemplo de este código en una aplicación de C#, vea [salida Use FOR JSON en una aplicación de cliente de C#](https://docs.microsoft.com/en-us/sql/relational-databases/json/use-for-json-output-in-sql-server-and-in-client-apps-sql-server#use-for-json-output-in-a-c-client-app).
2. Se da formato a los resultados como una matriz de objetos JSON.
   * El número de elementos de la matriz JSON es igual al número de filas en los resultados de la instrucción SELECT (antes de que se aplica la cláusula FOR JSON).
   * Cada fila de los resultados de la instrucción SELECT (antes de que se aplica la cláusula FOR JSON) se convierte en un objeto JSON independiente en la matriz.
   * Cada columna de los resultados de la instrucción SELECT (antes de la cláusula se aplica consulta JSON FOR) se convierte en una propiedad del objeto JSON.
3. Tanto los nombres de las columnas como sus valores van acompañados de un carácter de escape, como reza la sintaxis de JSON. Para obtener más información, vea [Cómo FOR JSON inserta caracteres de escape en los caracteres especiales y caracteres de control (SQL Server)](https://docs.microsoft.com/es-es/sql/relational-databases/json/how-for-json-escapes-special-characters-and-control-characters-sql-server).

Ejemplo

Este es un ejemplo que muestra cómo el **FOR JSON** cláusula da formato al resultado JSON.

**Resultados de la consulta**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| **A** | **B** | **C** | **D** |
| 10 | 11 | 12 | X |
| 20 | 21 | 22 | S |
| 30 | 31 | 32 | Z |

**Salida JSON**

JSONCopiar

[{

"A": 10,

"B": 11,

"C": 12,

"D": "X"

}, {

"A": 20,

"B": 21,

"C": 22,

"D": "Y"

}, {

"A": 30,

"B": 31,

"C": 32,

"D": "Z"

}]

**Formato de salida JSON anidada con el modo PATH (SQL Server)**

Para mantener el control total sobre la salida de la cláusula **FOR JSON**, especifique la opción**PATH**.

El modo**PATH** le permite crear objetos contenedores y anidar propiedades complejas. Se da formato a los resultados como una matriz de objetos JSON.

La alternativa es usar la opción **AUTO** para aplicar formato a la salida automáticamente en función de la estructura de la instrucción **SELECT**.

* Para más información sobre la opción **AUTO**, vea [Aplicar formato a la salida JSON automáticamente con el modo AUTO](https://docs.microsoft.com/es-es/sql/relational-databases/json/format-json-output-automatically-with-auto-mode-sql-server).
* Para obtener información general de ambas opciones, vea [Format Query Results as JSON with FOR JSON](https://docs.microsoft.com/es-es/sql/relational-databases/json/format-query-results-as-json-with-for-json-sql-server) (Aplicar formato JSON a los resultados de la consulta con FOR JSON).

Estos son algunos ejemplos de la cláusula **FOR JSON** con la opción **PATH** . Aplique formato a los resultados anidados utilizando nombres de columna separados por puntos o consultas anidadas, tal como se muestra en los ejemplos siguientes. De forma predeterminada los valores null no se incluyen en la salida de **FOR JSON**.

Ejemplo: nombres de columna separados por puntos

La siguiente consulta da formato a las cinco primeras filas de la tabla AdventureWorks Person como JSON.

La cláusula FOR JSON PATH usa el alias de columna o el nombre de columna para determinar el nombre de clave en la salida de JSON. Si un alias que contiene puntos, la opción PATH crea objetos anidados.

**Consulta**

SQLCopiar

SELECT TOP 5

BusinessEntityID As Id,

FirstName, LastName,

Title As 'Info.Title',

MiddleName As 'Info.MiddleName'

FROM Person.Person

FOR JSON PATH

**Resultado**

JSONCopiar

[{

"Id": 1,

"FirstName": "Ken",

"LastName": "Sánchez",

"Info": {

"MiddleName": "J"

}

}, {

"Id": 2,

"FirstName": "Terri",

"LastName": "Duffy",

"Info": {

"MiddleName": "Lee"

}

}, {

"Id": 3,

"FirstName": "Roberto",

"LastName": "Tamburello"

}, {

"Id": 4,

"FirstName": "Rob",

"LastName": "Walters"

}, {

"Id": 5,

"FirstName": "Gail",

"LastName": "Erickson",

"Info": {

"Title": "Ms.",

"MiddleName": "A"

}

}]

Ejemplo: varias tablas

Si se hace referencia a más de una tabla en una consulta, FOR JSON PATH anida cada columna mediante su alias. La siguiente consulta crea un objeto JSON por par (OrderHeader, OrderDetails) combinado en la consulta.

**Consulta**

SQLCopiar

SELECT TOP 2 SalesOrderNumber AS 'Order.Number',

OrderDate AS 'Order.Date',

UnitPrice AS 'Product.Price',

OrderQty AS 'Product.Quantity'

FROM Sales.SalesOrderHeader H

INNER JOIN Sales.SalesOrderDetail D

ON H.SalesOrderID = D.SalesOrderID

FOR JSON PATH

**Resultado**

JSONCopiar

[{

"Order": {

"Number": "SO43659",

"Date": "2011-05-31T00:00:00"

},

"Product": {

"Price": 2024.9940,

"Quantity": 1

}

}, {

"Order": {

"Number": "SO43659"

},

"Product": {

"Price": 2024.9940

}

}]

Para aplicar formato a la salida de la cláusula **FOR JSON** automáticamente en función de la estructura de la instrucción **SELECT**, especifique la opción **AUTO**.

Con la opción **AUTO** , el formato de la salida JSON se determina automáticamente según el orden de las columnas en la lista SELECT y sus tablas de origen. No se puede cambiar este formato.

La alternativa consiste en usar la opción **PATH** para mantener el control sobre la salida.

* Para obtener más información sobre la opción **PATH**, consulte [Format Nested JSON Output with PATH Mode](https://docs.microsoft.com/es-es/sql/relational-databases/json/format-nested-json-output-with-path-mode-sql-server) (Aplicar formato a la salida JSON anidada con el modo PATH).
* Para obtener información general sobre ambas opciones, consulte [Format Query Results as JSON with FOR JSON](https://docs.microsoft.com/es-es/sql/relational-databases/json/format-query-results-as-json-with-for-json-sql-server) (Aplicar formato a los resultados de la consulta como JSON con FOR JSON).

Una consulta que usa la opción **FOR JSON AUTO** debe tener una cláusula **FROM** .

Estos son algunos ejemplos de la cláusula **FOR JSON** con la opción **AUTO** .

Ejemplos

**Consulta 1**

Los resultados de la cláusula FOR JSON AUTO son similares a los de FOR JSON PATH cuando se usa solo una tabla en la consulta. En este caso, FOR JSON AUTO no crea objetos anidados. La única diferencia es que FOR JSON AUTO genera alias separados por puntos (por ejemplo, Info.MiddleName en el siguiente ejemplo) como claves con puntos, no como objetos anidados.

SELECT TOP 5

BusinessEntityID As Id,

FirstName, LastName,

Title As 'Info.Title',

MiddleName As 'Info.MiddleName'

FROM Person.Person

FOR JSON AUTO

**Resultado 1**

JSONCopiar

[{

"Id": 1,

"FirstName": "Ken",

"LastName": "Sánchez",

"Info.MiddleName": "J"

}, {

"Id": 2,

"FirstName": "Terri",

"LastName": "Duffy",

"Info.MiddleName": "Lee"

}, {

"Id": 3,

"FirstName": "Roberto",

"LastName": "Tamburello"

}, {

"Id": 4,

"FirstName": "Rob",

"LastName": "Walters"

}, {

"Id": 5,

"FirstName": "Gail",

"LastName": "Erickson",

"Info.Title": "Ms.",

"Info.MiddleName": "A"

}]

**Consulta 2**

Al unir tablas, las columnas de la primera tabla se generan como propiedades del objeto raíz. Las columnas de la segunda tabla se generan como propiedades de un objeto anidado. El nombre de tabla o alias de la segunda tabla (por ejemplo, D en el ejemplo siguiente) se usa como el nombre de la matriz anidada.

SQLCopiar

SELECT TOP 2 SalesOrderNumber,

OrderDate,

UnitPrice,

OrderQty

FROM Sales.SalesOrderHeader H

INNER JOIN Sales.SalesOrderDetail D

ON H.SalesOrderID = D.SalesOrderID

FOR JSON AUTO

**Resultado 2**

JSONCopiar

[{

"SalesOrderNumber": "SO43659",

"OrderDate": "2011-05-31T00:00:00",

"D": [{

"UnitPrice": 24.99,

"OrderQty": 1

}]

}, {

"SalesOrderNumber": "SO43659",

"D": [{

"UnitPrice": 34.40

}, {

"UnitPrice": 134.24,

"OrderQty": 5

}]

}]

**Consulta 3**  
En lugar de usar FOR JSON AUTO, puede anidar una subconsulta FOR JSON PATH en la instrucción SELECT, como se muestra en el ejemplo siguiente. Este ejemplo produce el mismo resultado que el ejemplo anterior.

SQLCopiar

SELECT TOP 2

SalesOrderNumber,

OrderDate,

(SELECT UnitPrice, OrderQty

FROM Sales.SalesOrderDetail AS D

WHERE H.SalesOrderID = D.SalesOrderID

FOR JSON PATH) AS D

FROM Sales.SalesOrderHeader AS H

FOR JSON PATH

**Resultado 3**

JSONCopiar

[{

"SalesOrderNumber": "SO43659",

"OrderDate": "2011-05-31T00:00:00",

"D": [{

"UnitPrice": 24.99,

"OrderQty": 1

}]

}, {

"SalesOrderNumber": "SO4390",

"D": [{

"UnitPrice": 24.99

}]

}]

Para agregar un solo elemento de nivel superior a la salida JSON de la cláusula **FOR JSON** , especifique la opción **ROOT** .

Si no especifica la opción **ROOT** , la salida JSON no incluye un elemento raíz.

Ejemplos

En la tabla siguiente se muestra la salida de la cláusula **FOR JSON** con y sin la opción **ROOT** .

En los ejemplos de la tabla siguiente se asume que el argumento *RootName* opcional está vacío. Si proporciona un nombre para el elemento raíz, este valor reemplaza el valor **root** en los ejemplos.

Sin la opción **ROOT**

JSONCopiar

{

<<json properties>>

}

JSONCopiar

[

<<json array elements>>

]

Con la opción **ROOT**

JSONCopiar

{

"root": {

<<json properties>>

}

}

JSONCopiar

{

"root": [

<< json array elements >>

]

}

Este es otro ejemplo de una cláusula **FOR JSON** con la opción **ROOT** . En este ejemplo se especifica un valor para el argumento *RootName* opcional.

**Query**

SELECT TOP 5

BusinessEntityID As Id,

FirstName, LastName,

Title As 'Info.Title',

MiddleName As 'Info.MiddleName'

FROM Person.Person

FOR JSON PATH, ROOT('info')

**Resultado**

JSONCopiar

{

"info": [{

"Id": 1,

"FirstName": "Ken",

"LastName": "Sánchez",

"Info": {

"MiddleName": "J"

}

}, {

"Id": 2,

"FirstName": "Terri",

"LastName": "Duffy",

"Info": {

"MiddleName": "Lee"

}

}, {

"Id": 3,

"FirstName": "Roberto",

"LastName": "Tamburello"

}, {

"Id": 4,

"FirstName": "Rob",

"LastName": "Walters"

}, {

"Id": 5,

"FirstName": "Gail",

"LastName": "Erickson",

"Info": {

"Title": "Ms.",

"MiddleName": "A"

}

}]

}

**Resultado (sin raíz)**

JSONCopiar

[{

"Id": 1,

"FirstName": "Ken",

"LastName": "Sánchez",

"Info": {

"MiddleName": "J"

}

}, {

"Id": 2,

"FirstName": "Terri",

"LastName": "Duffy",

"Info": {

"MiddleName": "Lee"

}

}, {

"Id": 3,

"FirstName": "Roberto",

"LastName": "Tamburello"

}, {

"Id": 4,

"FirstName": "Rob",

"LastName": "Walters"

}, {

"Id": 5,

"FirstName": "Gail",

"LastName": "Erickson",

"Info": {

"Title": "Ms.",

"MiddleName": "A"

}

}]

Para incluir valores NULL en la salida JSON de la cláusula **FOR JSON** , especifique la opción **INCLUDE\_NULL\_VALUES** .

Si no especifica la opción **INCLUDE\_NULL\_VALUES** , la salida JSON no incluye propiedades para valores NULL en los resultados de consulta.

Ejemplos

En el ejemplo siguiente se muestra la salida de la cláusula **FOR JSON** con y sin la opción **INCLUDE\_NULL\_VALUES** .

| Sin la opción **INCLUDE\_NULL\_VALUES** | Con la opción **INCLUDE\_NULL\_VALUES** |
| --- | --- |
| { "name": "John", "surname": "Doe" } | { "name": "John", "surname": "Doe", "age": null, "phone": null } |

Este es otro ejemplo de una cláusula **FOR JSON** con la opción **INCLUDE\_NULL\_VALUES** .

**Query**

SQLCopiar

SELECT name, surname

FROM emp

FOR JSON AUTO, INCLUDE\_NULL\_VALUES

**Resultado**

JSONCopiar

[{

"name": "John",

"surname": null

}, {

"name": "Jane",

"surname": "Doe"

}]

Para quitar los corchetes que rodean la salida JSON de la cláusula **FOR JSON** de manera predeterminada, especifique la opción **WITHOUT\_ARRAY\_WRAPPER** . Utilice esta opción con un sola fila de resultados para generar un objeto JSON único como salida en lugar de una matriz con un único elemento.

Si utiliza esta opción con un resultado de varias filas, el resultado no es un valor JSON válido debido a varios elementos y los corchetes que faltan.

Ejemplo (resultados de fila única)

En el ejemplo siguiente se muestra la salida de la cláusula **FOR JSON** con y sin la opción **WITHOUT\_ARRAY\_WRAPPER** .

**Consulta**

SQLCopiar

SELECT 2015 as year, 12 as month, 15 as day

FOR JSON PATH, WITHOUT\_ARRAY\_WRAPPER

**Resultado** con la opción **WITHOUT\_ARRAY\_WRAPPER**

JSONCopiar

{

"year": 2015,

"month": 12,

"day": 15

}

**Resultado** (predeterminado) sin el **WITHOUT\_ARRAY\_WRAPPER** opción

JSONCopiar

[{

"year": 2015,

"month": 12,

"day": 15

}]

Ejemplo (resultados de varias filas)

Este es otro ejemplo de una cláusula **FOR JSON** con y sin la opción **WITHOUT\_ARRAY\_WRAPPER** . Este ejemplo genera un resultado de varias filas. El resultado no es un valor JSON válido debido a varios elementos y los corchetes que faltan.

**Consulta**

SELECT TOP 3 SalesOrderNumber, OrderDate, Status

FROM Sales.SalesOrderHeader

ORDER BY ModifiedDate

FOR JSON PATH, WITHOUT\_ARRAY\_WRAPPER

**Resultado** con la opción **WITHOUT\_ARRAY\_WRAPPER**

{

"SalesOrderNumber": "SO43662",

"OrderDate": "2011-05-31T00:00:00",

"Status": 5

}, {

"SalesOrderNumber": "SO43661",

"OrderDate": "2011-05-31T00:00:00",

"Status": 5

}, {

"SalesOrderNumber": "SO43660",

"OrderDate": "2011-05-31T00:00:00",

"Status": 5

}

**Resultado** (predeterminado) sin el **WITHOUT\_ARRAY\_WRAPPER** opción

[{

"SalesOrderNumber": "SO43662",

"OrderDate": "2011-05-31T00:00:00",

"Status": 5

}, {

"SalesOrderNumber": "SO43661",

"OrderDate": "2011-05-31T00:00:00",

"Status": 5

}, {

"SalesOrderNumber": "SO43660",

"OrderDate": "2011-05-31T00:00:00",

"Status": 5

}]

La cláusula **FOR JSON** usa las siguientes reglas para convertir tipos de datos SQL Server en tipos JSON en la salida JSON.

| Categoría | Tipo de datos de SQL Server | Tipo de datos JSON |
| --- | --- | --- |
| Tipos de carácter y cadena | Char, nchar, varchar, nvarchar | string |
| Tipos numéricos | int, bigint, float, decimal, numeric | number |
| Tipo de bit | bit | Booleano (true o false) |
| Tipos de fecha y hora | date, datetime, datetime2, time, datetimeoffset | string |
| Tipos binarios | varbinary, binary, image, timestamp, rowversion | Cadena codificada en BASE64 |
| Tipos CLR | Geometry, geography, otros tipos CLR | No compatible. Estos tipos devuelven un error.  En la instrucción SELECT, use CAST o convertir, o usar una propiedad CLR o un método, para convertir los datos de origen a un tipo de datos de SQL Server que se puede convertir correctamente a un tipo JSON. Por ejemplo, utilice **STAsText()** para el tipo de geometría o use **ToString()** para cualquier tipo CLR. El tipo del valor de salida JSON, a continuación, se deriva el tipo de valor devuelto de la conversión que se aplican en la instrucción SELECT. |
| Otros tipos | uniqueidentifier, money | string |

En este tema se describe cómo la cláusula **FOR JSON** de una instrucción **SELECT** de SQL Server inserta caracteres de escape en caracteres especiales y representa caracteres de control en la salida JSON.

##### **Importante**

En esta página se describe la compatibilidad integrada con JSON en Microsoft SQL Server. Para obtener información general sobre la inserción de caracteres de escape y la codificación en JSON, vea la sección 2.5 de JSON RFC: <http://www.ietf.org/rfc/rfc4627.txt>.

## Secuencia de caracteres de escape en los caracteres especiales

Si los datos de origen contienen caracteres especiales, la cláusula **FOR JSON** inserta caracteres de escape en ellos en la salida de JSON con \, tal y como se muestra en la tabla siguiente. Esta secuencia de escape se produce tanto en los nombres de propiedades como en sus valores.

| **Carácter especial** | **Salida con escape** |
| --- | --- |
| Comillas (") | \" |
| Barra diagonal inversa (\) | \\ |
| Barra diagonal (/) | \/ |
| Retroceso | \b |
| Avance de página | \f |
| Nueva línea | \n |
| Retorno de carro | \r |
| Tabulación horizontal | \t |

## Caracteres de control

Si los datos de origen contienen caracteres de control, la cláusula **FOR JSON** los codifica en la salida de JSON en formato \u<code>, tal y como se muestra en la tabla siguiente.

| **Carácter de control** | **Salida codificada** |
| --- | --- |
| CHAR(0) | \u0000 |
| CHAR(1) | \u0001 |
| … | … |
| CHAR(31) | \u001f |

## Ejemplo

Se trata de un ejemplo de la salida de **FOR JSON** en datos de origen que incluye caracteres especiales y caracteres de control.

Consulta:

SQLCopiar

SELECT

'VALUE\ /

"' as [KEY\/"],

CHAR(0) as '0',

CHAR(1) as '1',

CHAR(31) as '31'

FOR JSON PATH

Resultado:

JSONCopiar

{

"KEY\\\t\/\"": "VALUE\\\t\/\r\n\"",

"0": "\u0000",

"1": "\u0001",

"31": "\u001f"

}

Los siguientes ejemplos muestran algunas de las formas de usar el **FOR JSON** su JSON y la cláusula output en SQL Server o en aplicaciones cliente.

Uso de salidas FOR JSON en variables de SQL Server

La salida de la cláusula FOR JSON es de tipo nvarchar (max), por lo que puede asignar a cualquier variable, como se muestra en el ejemplo siguiente.

DECLARE @x NVARCHAR(MAX) = (SELECT TOP 10 \* FROM Sales.SalesOrderHeader FOR JSON AUTO)

Uso de salidas FOR JSON en funciones definidas por el usuario de SQL Server

Puede crear funciones definidas por el usuario que formatean conjuntos de resultados como JSON y devuelven esta salida JSON. En el ejemplo siguiente se crea una función definida por el usuario que recupera algunas filas de detalle de pedido de ventas y las formatea como una matriz JSON.

CREATE FUNCTION GetSalesOrderDetails(@salesOrderId int)

RETURNS NVARCHAR(MAX)

AS

BEGIN

RETURN (SELECT UnitPrice, OrderQty

FROM Sales.SalesOrderDetail

WHERE SalesOrderID = @salesOrderId

FOR JSON AUTO)

END

Puede utilizar esta función en un lote o una consulta, como se muestra en el ejemplo siguiente.

DECLARE @x NVARCHAR(MAX) = dbo.GetSalesOrderDetails(43659)

PRINT dbo.GetSalesOrderDetails(43659)

SELECT TOP 10

H.\*, dbo.GetSalesOrderDetails(H.SalesOrderId) AS Details

FROM Sales.SalesOrderHeader H

Combinación de datos primarios y secundarios en una sola tabla

En el ejemplo siguiente, cada conjunto de filas secundarias se formatea como una matriz JSON. La matriz JSON se convierte en el valor de la columna de detalles de la tabla primaria.

SQLCopiar

SELECT TOP 10 SalesOrderId, OrderDate,

(SELECT TOP 3 UnitPrice, OrderQty

FROM Sales.SalesOrderDetail D

WHERE H.SalesOrderId = D.SalesOrderID

FOR JSON AUTO) AS Details

INTO SalesOrder

FROM Sales.SalesOrderHeader H

Actualización de los datos en columnas JSON

En el ejemplo siguiente se muestra que se puede actualizar el valor de una columna que contiene texto JSON.

SQLCopiar

UPDATE SalesOrder

SET Details =

(SELECT TOP 1 UnitPrice, OrderQty

FROM Sales.SalesOrderDetail D

WHERE D.SalesOrderId = SalesOrder.SalesOrderId

FOR JSON AUTO

Uso de salidas FOR JSON en una aplicación cliente de C#

En el ejemplo siguiente se muestra cómo recuperar la salida JSON de una consulta en un objeto StringBuilder en una aplicación cliente de C#. Se asume que la variable queryWithForJson contiene el texto de una instrucción SELECT con una cláusula FOR JSON.

C#Copiar

var queryWithForJson = "SELECT ... FOR JSON";

var conn = new SqlConnection("<connection string>");

var cmd = new SqlCommand(queryWithForJson, conn);

conn.Open();

var jsonResult = new StringBuilder();

var reader = cmd.ExecuteReader();

if (!reader.HasRows)

{

jsonResult.Append("[]");

}

else

{

while (reader.Read())

{

jsonResult.Append(reader.GetValue(0).ToString());

}

}

La función de conjunto de filas **OPENJSON** convierte texto JSON en un conjunto de filas y columnas. Utilice **OPENJSON** para ejecutar consultas SQL en colecciones de JSON o para importar texto JSON en tablas de SQL Server.

La función **OPENJSON** toma un objeto JSON o una colección de objetos JSON y los transforma en una o varias filas. De forma predeterminada, la función **OPENJSON** devuelve la siguiente información.

* Desde un objeto JSON, todos los pares de clave-valor que encuentra en el primer nivel.
* Desde una matriz JSON, todos los elementos de la matriz con sus índices.

Si lo desea, agregue una cláusula **WITH** para especificar el esquema de las filas que devuelve la función **OPENJSON**. Este esquema explícito define la estructura de la salida.

Uso de OPENJSON sin un esquema explícito para la salida

Cuando se usa la función **OPENJSON** sin proporcionar un esquema explícito para los resultados: es decir, sin una cláusula **WITH** después de OPENJSON, la función devuelve una tabla con las siguientes tres columnas.

1. El nombre de la propiedad en el objeto de entrada (o el índice del elemento en la matriz de entrada).
2. El valor de la propiedad o el elemento de matriz.
3. El tipo (por ejemplo, cadena, número, booleano, matriz u objeto).

Cada propiedad del objeto JSON o cada elemento de la matriz se devuelve como una fila independiente.

En el siguiente ejemplo rápido se usa **OPENJSON** con el esquema predeterminado y se devuelve una fila por cada propiedad del objeto JSON.

**Ejemplo**

DECLARE @json NVARCHAR(MAX)

SET @json='{"name":"John","surname":"Doe","age":45,"skills":["SQL","C#","MVC"]}';

SELECT \*

FROM OPENJSON(@json);

**Resultado**

| Key | value | tipo |
| --- | --- | --- |
| name | John | 1 |
| surname | Doe | 1 |
| age | 45 | 2 |
| skills | ["SQL","C#","MVC"] | 4 |

Más información

Para obtener más información y ejemplos, vea [Uso de OPENJSON con el esquema predeterminado (SQL Server)](https://docs.microsoft.com/es-es/sql/relational-databases/json/use-openjson-with-the-default-schema-sql-server).

Para ver la sintaxis y el uso, consulte [OPENJSON (Transact-SQL)](https://docs.microsoft.com/es-es/sql/t-sql/functions/openjson-transact-sql).

Uso de OPENJSON con un esquema explícito para la salida

Cuando se especifica el esquema de los resultados con la cláusula **WITH** de la función **JSON**, esta devuelve una tabla con las columnas definidas en la cláusula **WITH**. En la cláusula **WITH** se pueden especificar un conjunto de columnas de salida, sus tipos y las rutas de acceso de las propiedades de origen de JSON de cada valor de salida. **OPENJSON** iterará por la matriz de objetos JSON, leerá el valor en la ruta de acceso especificada para cada columna y convertirá el valor al tipo especificado.

A continuación, se muestra un ejemplo rápido que usa **OPENJSON** con un esquema para los resultados especificados de forma explícita.

**Ejemplo**

SQLCopiar

DECLARE @json NVARCHAR(MAX)

SET @json =

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

}

}

]'

SELECT \* FROM

OPENJSON ( @json )

WITH (

Number varchar(200) '$.Order.Number' ,

Date datetime '$.Order.Date',

Customer varchar(200) '$.AccountNumber',

Quantity int '$.Item.Quantity'

)

**Resultado**

| Number | Date | Customer | Cantidad |
| --- | --- | --- | --- |
| SO43659 | 2011-05-31T00:00:00 | AW29825 | 1 |
| SO43661 | 2011-06-01T00:00:00 | AW73565 | 3 |

Esta función devuelve los elementos de una matriz JSON y les da formato.

* Por cada elemento de la matriz JSON, **OPENJSON** genera una nueva fila en la tabla de salida. Los dos elementos de la matriz JSON se convierten en dos filas en la tabla devuelta.
* Para cada columna especificada mediante la sintaxis colName type json\_path, la función **OPENJSON** convierte los valores que encuentra en los elementos de cada matriz en la ruta de acceso especificada, los convierte al tipo especificado y rellena una celda de la tabla de salida. En este ejemplo, se toman los valores de la columna Date de cada objeto en una ruta de acceso$.Order.Date y se convierten en valores de fecha y hora.

Una vez que transforma una colección de JSON en un conjunto de filas con **OPENJSON**, puede ejecutar cualquier consulta SQL en los datos devueltos o insertarlos en una tabla.

OPENJSON requiere el nivel de compatibilidad 130

La función **OPENJSON** solo está disponible en el **nivel de compatibilidad 130**. Si el nivel de compatibilidad de la base de datos es inferior a 130, SQL Server no podrá encontrar ni ejecutar la función **OPENJSON** . Hay otras funciones integradas de JSON que sí están disponibles en todos los niveles de compatibilidad. Puede comprobar el nivel de compatibilidad en la vista sys.databases o en las propiedades de la base de datos.

Puede cambiar un nivel de compatibilidad de la base de datos mediante el comando siguiente:  
ALTER DATABASE <DatabaseName> SET COMPATIBILITY\_LEVEL = 130

En los ejemplos de esta página se usa el siguiente texto de JSON, que contiene un elemento complejo.

DECLARE @jsonInfo NVARCHAR(MAX)

SET @jsonInfo=N'{

"info":{

"type":1,

"address":{

"town":"Bristol",

"county":"Avon",

"country":"England"

},

"tags":["Sport", "Water polo"]

},

"type":"Basic"

}'

Validar texto JSON mediante la función ISJSON

La función **ISJSON** prueba si una cadena contiene un valor JSON válido.

En el ejemplo siguiente se devuelve el texto JSON si la columna contiene un valor JSON válido.

SELECT id,json\_col

FROM tab1

WHERE ISJSON(json\_col)>0

Para obtener más información, vea [ISJSON (Transact-SQL)](https://docs.microsoft.com/es-es/sql/t-sql/functions/isjson-transact-sql).

Extraer un valor de texto JSON mediante la función JSON\_VALUE

La función **JSON\_VALUE** extrae un valor escalar de una cadena JSON.

En el ejemplo siguiente se extrae el valor de una propiedad JSON en una variable local.

SET @town=JSON\_VALUE(@jsonInfo,'$.info.address.town')

Extraer un objeto o una matriz de texto JSON mediante la función JSON\_QUERY

La función **JSON\_QUERY** extrae un objeto o una matriz de una cadena JSON.

En el ejemplo siguiente se muestra cómo devolver un fragmento de JSON en los resultados de la consulta.

SQLCopiar

SELECT FirstName,LastName,JSON\_QUERY(jsonInfo,'$.info.address') AS Address

FROM Person.Person

ORDER BY LastName

Comparación de JSON\_VALUE y JSON\_QUERY

La diferencia clave entre **JSON\_VALUE** y **JSON\_QUERY** es que **JSON\_VALUE** devuelve un valor escalar, mientras que **JSON\_QUERY** devuelve un objeto o una matriz.

Observe el siguiente ejemplo de texto JSON.

{

"a": "[1,2]",

"b": [1, 2],

"c": "hi"

}

En este ejemplo de texto JSON, los miembros de datos "a" y "c" son valores de cadena, mientras que el miembro de datos "b" es una matriz.**JSON\_VALUE** y **JSON\_QUERY** devuelven los resultados siguientes:

| Query | **JSON\_VALUE** devuelve | **JSON\_QUERY** devuelve |
| --- | --- | --- |
| **$** | NULL o error | { "a": "[1,2]", "b": [1,2], "c":"hi"} |
| **$.a** | [1,2] | NULL o error |
| **$.b** | NULL o error | [1,2] |
| **$.b[0]** | 1 | NULL o error |
| **$.c** | hi | NULL o error |

Probar JSON\_VALUE y JSON\_QUERY con la base de datos de ejemplo AdventureWorks

Pruebe las funciones integradas que se describen en este tema ejecutando los ejemplos siguientes con la base de datos de ejemplo AdventureWorks, que contiene datos JSON. Para obtener la base de datos de ejemplo AdventureWorks, [haga clic aquí](http://www.microsoft.com/en-us/download/details.aspx?id=49502).

En los ejemplos siguientes, la columna Info de la tabla SalesOrder\_json contiene texto JSON.

Ejemplo 1: devolver columnas estándar y datos JSON

La consulta siguiente devuelve columnas y valores estándar y relacionales de una columna JSON.

SELECT SalesOrderNumber,OrderDate,Status,ShipDate,Status,AccountNumber,TotalDue,

JSON\_QUERY(Info,'$.ShippingInfo') ShippingInfo,

JSON\_QUERY(Info,'$.BillingInfo') BillingInfo,

JSON\_VALUE(Info,'$.SalesPerson.Name') SalesPerson,

JSON\_VALUE(Info,'$.ShippingInfo.City') City,

JSON\_VALUE(Info,'$.Customer.Name') Customer,

JSON\_QUERY(OrderItems,'$') OrderItems

FROM Sales.SalesOrder\_json

WHERE ISJSON(Info)>0

Ejemplo 2: agregar y filtrar valores JSON

La consulta siguiente agrega subtotales por nombre de cliente (almacenados en JSON) y estado (almacenado en una columna normal). A continuación, filtra los resultados por ciudad (almacenados en JSON) y OrderDate (almacenados en una columna normal).

SQLCopiar

DECLARE @territoryid INT;

DECLARE @city NVARCHAR(32);

SET @territoryid=3;

SET @city=N'Seattle';

SELECT JSON\_VALUE(Info,'$.Customer.Name') AS Customer,Status,SUM(SubTotal) AS Total

FROM Sales.SalesOrder\_json

WHERE TerritoryID=@territoryid

AND JSON\_VALUE(Info,'$.ShippingInfo.City')=@city

AND OrderDate>'1/1/2015'

GROUP BY JSON\_VALUE(Info,'$.Customer.Name'),Status

HAVING SUM(SubTotal)>1000

Actualizar valores de propiedad en texto JSON mediante la función JSON\_MODIFY

La función **JSON\_MODIFY** actualiza el valor de una propiedad en una cadena JSON y devuelve la cadena JSON actualizada.

En el siguiente ejemplo se actualiza el valor de una propiedad de una variable que contiene JSON.

SET @info=JSON\_MODIFY(@jsonInfo,"$.info.address[0].town",'London')

Usar expresiones de ruta de acceso JSON para hacer referencia a las propiedades de objetos JSON.

Al llamar a las siguientes funciones, hay que proporcionar una expresión de ruta de acceso.

* Cuando se llama a **OPENJSON** para crear una vista relacional de los datos JSON. Para obtener más información, vea [OPENJSON (Transact-SQL)](https://docs.microsoft.com/es-es/sql/t-sql/functions/openjson-transact-sql).
* Cuando se llama a **JSON\_VALUE** para extraer un valor de texto JSON. Para obtener más información, vea [JSON\_VALUE (Transact-SQL)](https://docs.microsoft.com/es-es/sql/t-sql/functions/json-value-transact-sql).
* Cuando se llama a **JSON\_QUERY** para extraer un objeto JSON o una matriz. Para obtener más información, vea [JSON\_QUERY (Transact-SQL)](https://docs.microsoft.com/es-es/sql/t-sql/functions/json-query-transact-sql).
* Cuando se llama a **JSON\_MODIFY** para actualizar el valor de una propiedad en una cadena JSON. Para obtener más información, vea [JSON\_MODIFY (Transact-SQL)](https://docs.microsoft.com/es-es/sql/t-sql/functions/json-modify-transact-sql).

Partes de una expresión de ruta de acceso

Una expresión de ruta de acceso tiene dos componentes.

1. Opcional [modo path](https://docs.microsoft.com/es-es/sql/relational-databases/json/json-path-expressions-sql-server#PATHMODE), con un valor de **lax** o **estricta**.
2. La [ruta de acceso](https://docs.microsoft.com/es-es/sql/relational-databases/json/json-path-expressions-sql-server#PATH) en sí.

Path mode

Al principio de la expresión de ruta de acceso, puede optar por declarar el modo de la ruta de acceso con las palabras clave **lax** o **strict**. El valor predeterminado es **lax**.

* En **lax** modo, la función devuelve valores vacíos si la expresión de ruta de acceso contiene un error. Por ejemplo, si se solicita el valor **$.name**, y el texto JSON no contiene un **nombre** clave, la función devuelve null, pero no genera un error.
* En **estricta** modo, la función genera un error si la expresión de ruta de acceso contiene un error.

La consulta siguiente especifica explícitamente lax modo en la expresión de ruta de acceso.

DECLARE @json NVARCHAR(MAX)

SET @json=N'{ ... }'

SELECT \* FROM OPENJSON(@json, N'lax $.info')

Path

Después de declarar (opcionalmente) el modo de ruta de acceso, se especifica la ruta de acceso.

* El signo de dólar ($) representa el elemento de contexto.
* La ruta de acceso de propiedad es un conjunto de pasos de ruta de acceso. Los pasos de ruta de acceso pueden contener los siguientes elementos y operadores.
  + Nombres de clave. Por ejemplo, $.name y $."first name". Si el nombre de clave comienza por un signo de dólar o contiene caracteres especiales (como espacios), insértelo entre comillas.
  + Elementos de matriz. Por ejemplo, $.product[3]. Las matrices tienen una base cero.
  + El operador de punto (.) indica un miembro de un objeto. Por ejemplo, en $.people[1].surname, surname es un elemento secundario de people.

Ejemplos

Los ejemplos de esta sección hacen referencia al siguiente texto JSON.

{

"people": [{

"name": "John",

"surname": "Doe"

}, {

"name": "Jane",

"surname": null,

"active": true

}]

}

En la siguiente tabla se muestran algunos ejemplos de expresiones de ruta de acceso.

| Expresión de ruta de acceso | Valor | |
| --- | --- | --- |
| $.people[0].name | John |
| $.people[1] | { "name": "Jane", "surname": null, "active": true } | |
| $.people[1].surname | null | |
| $ | { "people": [ { "name": "John", "surname": "Doe" }, { "name": "Jane", "surname": null, "active": true } ] } | |

Control de las rutas de acceso duplicadas por las funciones integradas

Si el texto JSON contiene propiedades duplicadas: por ejemplo, dos claves con el mismo nombre en el mismo nivel - el **JSON\_VALUE** y **JSON\_QUERY** funciones devuelven solo el primer valor que coincide con la ruta de acceso. Para analizar un objeto JSON que contiene las claves duplicadas y devolver todos los valores, utilice **OPENJSON**, tal y como se muestra en el ejemplo siguiente.

SQLCopiar

DECLARE @json NVARCHAR(MAX)

SET @json=N'{"person":{"info":{"name":"John", "name":"Jack"}}}'

SELECT value

FROM OPENJSON(@json,'$.person.info')

En este tema se describe cómo importar archivos JSON a SQL Server. Actualmente hay una gran cantidad de documentos JSON almacenados en archivos. Información de registro de aplicaciones en archivos JSON, sensores generan información que se almacena en archivos JSON y así sucesivamente. Es importante ser capaz de leer los datos JSON almacenados en archivos, cargar los datos en SQL Server y analizarlos.

Importar un documento JSON en una sola columna

**OPENROWSET(BULK)** es una función con valores de tabla que puede leer datos de cualquier archivo que se encuentre en la unidad local o la red, si SQL Server tiene acceso de lectura a esa ubicación. Devuelve una tabla con una sola columna con el contenido del archivo. Hay varias opciones que puede utilizar con la función OPENROWSET(BULK), como pueden ser los separadores. Pero en el caso más simple, solamente puede cargar todo el contenido de un archivo como un valor de texto. (Este valor grande único se conoce como un objeto grande de carácter único o SINGLE\_CLOB).

Este es un ejemplo de la función **OPENROWSET(BULK)** que lee el contenido de un archivo JSON y lo devuelve al usuario como un valor único:

SQLCopiar

SELECT BulkColumn

FROM OPENROWSET (BULK 'C:\JSON\Books\book.json', SINGLE\_CLOB) as j

OPENJSON(BULK) lee el contenido del archivo y lo devuelve en BulkColumn.

También puede cargar el contenido del archivo en una variable local o en una tabla, como se muestra en el ejemplo siguiente:

SQLCopiar

-- Load file contents into a variable

SELECT @json = BulkColumn

FROM OPENROWSET (BULK 'C:\JSON\Books\book.json', SINGLE\_CLOB) as j

-- Load file contents into a table

SELECT BulkColumn

INTO #temp

FROM OPENROWSET (BULK 'C:\JSON\Books\book.json', SINGLE\_CLOB) as j

Después de cargar el contenido del archivo JSON, puede guardar el texto JSON en una tabla.

Importar varios documentos JSON

Puede usar el mismo enfoque para cargar un conjunto de archivos JSON desde el sistema de archivos en una variable local de uno en uno.Supongamos que los archivos se llaman book<index>.json.

SQLCopiar

DECLARE @i INT = 1

DECLARE @json AS NVARCHAR(MAX)

WHILE(@i < 10)

BEGIN

SET @file = 'C:\JSON\Books\book' + cast(@i AS VARCHAR(5)) + '.json';

SELECT @json = BulkColumn FROM OPENROWSET (BULK (@file), SINGLE\_CLOB) AS j

SELECT \* FROM OPENJSON(@json) AS json

-- Optionally, save the JSON text in a table.

SET @i = @i + 1 ;

END

Importar documentos JSON desde Azure File Storage

También puede utilizar OPENROWSET como se describió anteriormente para leer archivos JSON de otras ubicaciones de archivos que puede tener acceso SQL Server. Por ejemplo, Azure File Storage admite el protocolo SMB. Como resultado, puede asignar una unidad virtual local al recurso compartido de Azure File Storage mediante el procedimiento siguiente:

1. Cree una cuenta de almacenamiento de archivos (por ejemplo, mystorage), un recurso compartido de archivos (por ejemplo, sharejson) y una carpeta de archivos en Azure File Storage mediante Azure Portal o Azure PowerShell.
2. Cargue algunos archivos JSON en el recurso compartido de almacenamiento de archivos.
3. Cree una regla de firewall de salida en el Firewall de Windows en el equipo que permite el puerto 445. Tenga en cuenta que su proveedor de servicio de Internet puede bloquear este puerto. Si recibe un error DNS (error 53) en el paso siguiente, es que no ha abierto el puerto 445 o su ISP lo bloquea.
4. Montar el recurso compartido de almacenamiento de archivos de Azure como una unidad local (por ejemplo T:).

Aquí se muestra la sintaxis de comando:

net use [drive letter] \\[storage name].file.core.windows.net\[share name] /u:[storage account name] [storage account access key]

Este es un ejemplo que asigna la letra de unidad local T: al recurso compartido de almacenamiento de archivos de Azure:

net use t: \\mystorage.file.core.windows.net\sharejson /u:myaccount hb5qy6eXLqIdBj0LvGMHdrTiygkjhHDvWjUZg3Gu7bubKLg==

Puede encontrar la clave de la cuenta de almacenamiento y la clave de acceso de la cuenta de almacenamiento principal o secundaria en la sección Claves de Configuración en Azure Portal.

1. Ahora puede acceder los archivos JSON desde el recurso compartido de almacenamiento de archivos de Azure mediante el uso de la unidad asignada, como se muestra en el ejemplo siguiente:

SELECT book.\* FROM

OPENROWSET(BULK N't:\books\books.json', SINGLE\_CLOB) AS json

CROSS APPLY OPENJSON(BulkColumn)

WITH( id nvarchar(100), name nvarchar(100), price float,

pages\_i int, author nvarchar(100)) AS book

Para más información sobre Azure File Storage, vea [File Storage](https://azure.microsoft.com/en-us/services/storage/files/).

Importar documentos JSON desde Azure Blob Storage

Puede cargar archivos directamente en la base de datos de SQL de Azure desde el almacenamiento de blobs de Azure con el comando T-SQL BULK INSERT o la función OPENROWSET.

En primer lugar, cree un origen de datos externo, como se muestra en el ejemplo siguiente.

SQLCopiar

CREATE EXTERNAL DATA SOURCE MyAzureBlobStorage

WITH ( TYPE = BLOB\_STORAGE,

LOCATION = 'https://myazureblobstorage.blob.core.windows.net',

CREDENTIAL= MyAzureBlobStorageCredential);

A continuación, ejecute un comando BULK INSERT con la opción DATA\_SOURCE.

SQLCopiar

BULK INSERT Product

FROM 'data/product.dat'

WITH ( DATA\_SOURCE = 'MyAzureBlobStorage');

Para obtener más información y un ejemplo que utiliza OPENROWSET, vea [cargar archivos desde el almacenamiento de blobs de Azure en la base de datos de SQL Azure](https://blogs.msdn.microsoft.com/sqlserverstorageengine/2017/02/23/loading-files-from-azure-blob-storage-into-azure-sql-database/).

Analizar documentos JSON en filas y columnas

En lugar de leer un archivo JSON completo como un valor único, puede que desee analizarlos y devolver los libros en el archivo y sus propiedades en filas y columnas. En el ejemplo siguiente se usa un archivo JSON de [este sitio](https://github.com/tamingtext/book/blob/master/apache-solr/example/exampledocs/books.json) que contiene una lista de libros.

Ejemplo 1

En el ejemplo más simple, solo puede cargar toda la lista desde el archivo.

SQLCopiar

SELECT value

FROM OPENROWSET (BULK 'C:\JSON\Books\books.json', SINGLE\_CLOB) as j

CROSS APPLY OPENJSON(BulkColumn)

Ejemplo 2

OPENROWSET lee un solo valor de texto del archivo, lo devuelve como BulkColumn y lo pasa a la función OPENJSON. OPENJSON recorre en iteración la matriz de objetos JSON de la matriz BulkColumn y devuelve un libro en cada fila, con formato JSON:

Copiar

{"id":"978-0641723445″, "cat":["book","hardcover"], "name":"The Lightning Thief", …

{"id":"978-1423103349″, "cat":["book","paperback"], "name":"The Sea of Monsters", …

{"id":"978-1857995879″, "cat":["book","paperback"], "name":"Sophie’s World : The Greek …

{"id":"978-1933988177″, "cat":["book","paperback"], "name":"Lucene in Action, Second …

Ejemplo 3

La función OPENJSON puede analizar el contenido JSON y transformarlo en una tabla o un conjunto de resultados. En el ejemplo siguiente se carga el contenido, se analiza el JSON cargado y se devuelven los cinco campos como columnas:

SQLCopiar

SELECT book.\*

FROM OPENROWSET (BULK 'C:\JSON\Books\books.json', SINGLE\_CLOB) as j

CROSS APPLY OPENJSON(BulkColumn)

WITH( id nvarchar(100), name nvarchar(100), price float,

pages\_i int, author nvarchar(100)) AS book

En este ejemplo, OPENROWSET(BULK) lee el contenido del archivo y pasa ese contenido a la función OPENJSON con un esquema definido para la salida. OPENJSON hace coincidir las propiedades de los objetos JSON utilizando nombres de columna. Por ejemplo, la propiedad price se devuelve como una columna price y se convierte al tipo de datos float. He aquí los resultados:

| Identificador | Nombre | price | pages\_i | Autor |
| --- | --- | --- | --- | --- |
| 978-0641723445 | The Lightning Thief | 12,5 | 384 | Rick Riordan |
| 978-1423103349 | The Sea of Monsters | 6,49 | 304 | Rick Riordan |
| 978-1857995879 | Sophie’s World : The Greek Philosophers | 3,07 | 64 | Jostein Gaarder |
| 978-1933988177 | Lucene in Action, Second Edition | 30,5 | 475 | Michael McCandless |

Ahora puede devolver esta tabla al usuario o cargar los datos en otra tabla.

En SQL Server 2016, JSON no es un tipo de datos integrado y SQL Server no tiene índices personalizados JSON. Sin embargo, puede optimizar las consultas en documentos JSON, mediante índices estándar.

Índices de base de datos mejoran el rendimiento de las operaciones de filtro y ordenación. Sin ellos, SQL Server debe realizar un examen completo de la tabla cada vez que realice consultas de datos.

Indexación de propiedades JSON mediante columnas calculadas

Al almacenar datos JSON en SQL Server, normalmente querrá filtrar u ordenar resultados de la consulta por uno o varios *propiedades* de los documentos JSON.

Ejemplo

En este ejemplo, suponga que AdventureWorks SalesOrderHeader tabla tiene un Info columna que contiene diversa información en formato JSON sobre pedidos de venta. Por ejemplo, contiene información sobre clientes, vendedores, direcciones de envío y facturación y así sucesivamente. Desea usar los valores de la Info columna para filtrar los pedidos de ventas para un cliente.

Consulta para optimizar

Este es un ejemplo del tipo de consulta que se desea optimizar mediante un índice.

SQLCopiar

SELECT SalesOrderNumber,

OrderDate,

JSON\_VALUE(Info, '$.Customer.Name') AS CustomerName

FROM Sales.SalesOrderHeader

WHERE JSON\_VALUE(Info, '$.Customer.Name') = N'Aaron Campbell'

Índice del ejemplo

Si desea acelerar los filtros o ORDER BY cláusulas en una propiedad de un documento JSON, puede usar los mismos índices que ya está usando en otras columnas. Sin embargo, no se puede *directamente* hacer referencia a propiedades de los documentos JSON.

1. En primer lugar, tendrá que crear una "columna virtual" que devuelva los valores que desea usar para filtrar.
2. Después, debe crear un índice de esa columna virtual.

En el ejemplo siguiente se crea una columna calculada que se puede usar para la indización. A continuación, crea un índice en la nueva columna calculada. Este ejemplo crea una columna que muestra el nombre del cliente, que se almacena en la $.Customer.Name ruta de acceso en los datos JSON.

SQLCopiar

ALTER TABLE Sales.SalesOrderHeader

ADD vCustomerName AS JSON\_VALUE(Info,'$.Customer.Name')

CREATE INDEX idx\_soh\_json\_CustomerName

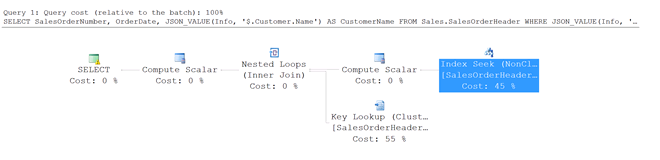
ON Sales.SalesOrderHeader(vCustomerName)

Para obtener más información acerca de la columna calculada

La columna calculada no se conserva. Se calcula solo cuando hay que volver a generar el índice. No ocupa espacio adicional en la tabla.

Es importante que crea la columna calculada con la misma expresión que va a usar en las consultas; en este ejemplo, la expresión es JSON\_VALUE(Info, '$.Customer.Name').

No tiene que volver a escribir las consultas. Si se usan expresiones con la JSON\_VALUE función, como se muestra en la consulta de ejemplo anterior, SQL Server considera que hay una columna calculada equivalente con la misma expresión y aplica un índice si es posible.

Plan de ejecución de este ejemplo

En lugar examinar toda la tabla, SQL Server busca un índice en el índice no agrupado e identifica las filas que satisfacen las condiciones especificadas. A continuación, usa una búsqueda de claves en el SalesOrderHeader tabla para capturar las otras columnas que se hace referencia en la consulta: en este ejemplo, SalesOrderNumber y OrderDate.

Optimizar aún más el índice con columnas incluidas

Puede evitar esta búsqueda adicional en la tabla si agrega las columnas necesarias en el índice. Puede agregar estas columnas como columnas incluidas estándar, tal como se muestra en el ejemplo siguiente, que amplía el CREATE INDEX ejemplo mostrado anteriormente.

SQLCopiar

CREATE INDEX idx\_soh\_json\_CustomerName

ON Sales.SalesOrderHeader(vCustomerName)

INCLUDE(SalesOrderNumber,OrderDate)

En este caso no tiene SQL Server leer datos adicionales desde la SalesOrderHeader tabla, ya que todo lo que necesita se incluye en el índice JSON no agrupado. Se trata de una buena forma de combinar datos JSON de columna en las consultas y crear índices óptimos para la carga de trabajo.

Los índices JSON son índices de intercalación

Una característica importante de los índices basados en datos JSON es que los índices son compatibles con la intercalación. El resultado de la JSON\_VALUE función que se usa cuando se crea la columna calculada es un valor de texto que hereda la intercalación de la expresión de entrada.Por lo tanto, los valores en el índice se ordenan con las reglas de intercalación definidas en las columnas de origen.

Para demostrar esto, en el ejemplo siguiente se crea una tabla de colección simple con una clave principal y el contenido JSON.

SQLCopiar

CREATE TABLE JsonCollection

(

id INT IDENTITY CONSTRAINT PK\_JSON\_ID PRIMARY KEY,

json NVARCHAR(MAX) COLLATE SERBIAN\_CYRILLIC\_100\_CI\_AI

CONSTRAINT [Content should be formatted as JSON]

CHECK(ISJSON(json)>0)

)

*El comando anterior especifica la intercalación del serbio (cirílico) para la columna JSON. En el ejemplo* *siguiente se rellena la tabla y crea un índice en la propiedad name.*

SQLCopiar

INSERT INTO JsonCollection

VALUES

(N'{"name":"Иво","surname":"Андрић"}'),

(N'{"name":"Андрија","surname":"Герић"}'),

(N'{"name":"Владе","surname":"Дивац"}'),

(N'{"name":"Новак","surname":"Ђоковић"}'),

(N'{"name":"Предраг","surname":"Стојаковић"}'),

(N'{"name":"Михајло","surname":"Пупин"}'),

(N'{"name":"Борислав","surname":"Станковић"}'),

(N'{"name":"Владимир","surname":"Грбић"}'),

(N'{"name":"Жарко","surname":"Паспаљ"}'),

(N'{"name":"Дејан","surname":"Бодирога"}'),

(N'{"name":"Ђорђе","surname":"Вајферт"}'),

(N'{"name":"Горан","surname":"Бреговић"}'),

(N'{"name":"Милутин","surname":"Миланковић"}'),

(N'{"name":"Никола","surname":"Тесла"}')

GO

ALTER TABLE JsonCollection

ADD vName AS JSON\_VALUE(json,'$.name')

CREATE INDEX idx\_name

ON JsonCollection(vName)

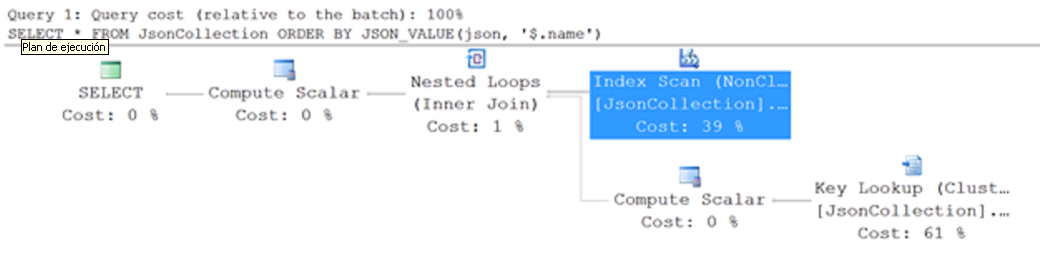
Los comandos anteriores crean un índice estándar en la columna calculada vName, que representa el valor de la JSON $.name propiedad. En la página de códigos del serbio (cirílico), el orden de las letras es 'А', 'Б', 'В', 'Г', 'Д', 'Ђ', 'Е', etc. El orden de los elementos en el índice es compatible con las reglas del serbio (cirílico) porque el resultado de la JSON\_VALUE función hereda la intercalación de la columna de origen. En el ejemplo siguiente se realiza una consulta de esta colección y se ordenan los resultados por nombre.

SQLCopiar

SELECT JSON\_VALUE(json,'$.name'),\*

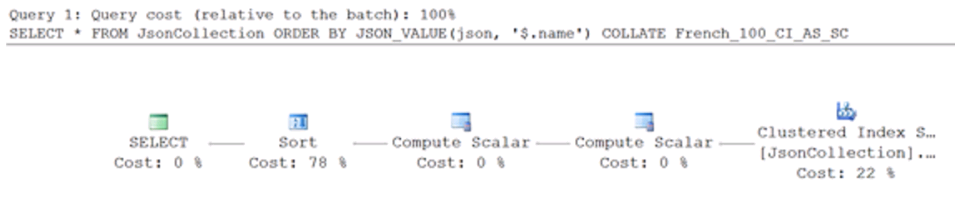
FROM JsonCollection

ORDER BY JSON\_VALUE(json,'$.name')

Si observa el plan de ejecución real, verá que emplea los valores ordenados del índice no agrupado.

Aunque la consulta tiene un ORDER BY cláusula, el plan de ejecución no usa un operador de ordenación. El índice JSON ya está ordenado según reglas del serbio (cirílico). Por lo tanto, SQL Server puede utilizar el índice no agrupado donde los resultados ya están ordenados.

Sin embargo, si cambiamos la intercalación de la ORDER BY expresión — por ejemplo, si colocamos COLLATE French\_100\_CI\_AS\_SC después de la JSON\_VALUE función - obtenemos un plan de ejecución de consulta diferente.



Puesto que el orden de los valores en el índice no cumple las reglas de intercalación del francés, SQL Server no puede utilizar el índice para ordenar los resultados. Por lo tanto, agrega un operador Sort que ordena los resultados mediante las reglas de intercalación del francés.

SQL Server y Azure SQL Database permiten trabajar con texto en formato JSON. Para aumentar el rendimiento de las consultas OLTP que procesan datos JSON, puede almacenar documentos JSON en tablas optimizadas en memoria utilizando las columnas de cadena estándar (tipo NVARCHAR).

Almacenamiento de datos JSON en tablas con optimización para memoria

En el ejemplo siguiente se crea una tabla Product con optimización para memoria con dos columnas: Tags y Data.

SQLCopiar

CREATE SCHEMA xtp;

GO

CREATE TABLE xtp.Product(

ProductID int PRIMARY KEY NONCLUSTERED, --standard column

Name nvarchar(400) NOT NULL, --standard column

Price float, --standard column

Tags nvarchar(400),--json stored in string column

Data nvarchar(4000) --json stored in string column

) WITH (MEMORY\_OPTIMIZED=ON);

Al almacenar datos JSON en tablas con optimización para memoria, se aumenta el rendimiento de consulta gracias a que se aprovecha el acceso a los datos en memoria sin bloqueo.

Optimización de datos JSON con más características en memoria

Las nuevas características disponibles en SQL Server y Azure SQL Database permiten integrar completamente las funcionalidades JSON con las tecnologías existentes de OLTP en memoria. Por ejemplo, puede realizar las siguientes tareas:

* Valide la estructura de los documentos JSON almacenados en tablas con optimización para memoria mediante las restricciones CHECK compiladas de forma nativa.
* Exponga y tipe fuertemente los valores almacenados en documentos JSON con columnas calculadas.
* Indexe los valores de los documentos JSON con índices con optimización para memoria.
* Compile de forma negativa las consultas SQL que usan valores de documentos JSON o formatee los resultados como texto JSON.

Validación de columnas JSON

SQL Server y Azure SQL Database permiten agregar restricciones CHECK compiladas de forma nativa que validan el contenido de los documentos JSON almacenados en una columna de cadena, tal y como se muestra en el ejemplo siguiente.

SQLCopiar

DROP TABLE IF EXISTS xtp.Product;

GO

CREATE TABLE xtp.Product(

ProductID int PRIMARY KEY NONCLUSTERED,

Name nvarchar(400) NOT NULL,

Price float,

Tags nvarchar(400)

CONSTRAINT [Tags should be formatted as JSON]

CHECK (ISJSON(Tags)=1),

Data nvarchar(4000)

) WITH (MEMORY\_OPTIMIZED=ON);

La restricción CHECK compilada de forma nativa puede agregarse en las tablas existentes que contienen columnas JSON:

SQLCopiar

ALTER TABLE xtp.Product

ADD CONSTRAINT [Data should be JSON]

CHECK (ISJSON(Data)=1)

Con las restricciones CHECK de JSON compiladas de forma nativa, puede asegurarse de que el texto JSON almacenado en las tablas con optimización para memoria tiene el formato correcto.

Exposición de valores JSON mediante columnas calculadas

Las columnas calculadas permiten exponer valores del texto JSON y obtener acceso a esos valores sin volver a evaluar las expresiones que capturan un valor del texto JSON y sin volver a analizar la estructura JSON. Los valores expuestos están fuertemente tipados y persisten físicamente en las columnas calculadas. Acceder a los valores JSON mediante columnas calculadas persistentes es más rápido que hacerlo a los valores del documento JSON.

En el ejemplo siguiente se muestra cómo exponer los dos valores siguientes de la columna Data JSON:

* El país donde se fabricó un producto.
* El costo de fabricación del producto.

SQLCopiar

DROP TABLE IF EXISTS xtp.Product;

GO

CREATE TABLE xtp.Product(

ProductID int PRIMARY KEY NONCLUSTERED,

Name nvarchar(400) NOT NULL,

Price float,

Data nvarchar(4000),

MadeIn AS CAST(JSON\_VALUE(Data, '$.MadeIn') as NVARCHAR(50)) PERSISTED,

Cost AS CAST(JSON\_VALUE(Data, '$.ManufacturingCost') as float)

) WITH (MEMORY\_OPTIMIZED=ON);

Las columnas calculadas MadeIn y Cost se actualizan cada vez que se guarda el documento JSON en los cambios de la columna Data.

Indexación de valores en las columnas JSON

SQL Server y Azure SQL Database permiten indexar valores en columnas JSON utilizando índices con optimización para memoria. Se deben exponer los valores JSON que se indexan y se tipan fuertemente utilizando las columnas calculadas, tal y como se muestra en el ejemplo siguiente.

SQLCopiar

DROP TABLE IF EXISTS xtp.Product;

GO

CREATE TABLE xtp.Product(

ProductID int PRIMARY KEY NONCLUSTERED,

Name nvarchar(400) NOT NULL,

Price float,

Data nvarchar(4000),

MadeIn AS CAST(JSON\_VALUE(Data, '$.MadeIn') as NVARCHAR(50)) PERSISTED,

Cost AS CAST(JSON\_VALUE(Data, '$.ManufacturingCost') as float) PERSISTED,

INDEX [idx\_Product\_MadeIn] NONCLUSTERED (MadeIn)

) WITH (MEMORY\_OPTIMIZED=ON)

ALTER TABLE Product

ADD INDEX [idx\_Product\_Cost] NONCLUSTERED HASH(Cost)

WITH (BUCKET\_COUNT=20000)

Los valores de columnas JSON se pueden indexar con los índices NONCLUSTERED y HASH.

* Los índices NONCLUSTERED optimizan las consultas que seleccionan rangos de filas por algún valor JSON u ordenan los resultados por valores JSON.
* Los índices HASH proporcionan un rendimiento óptimo cuando se recupera una sola fila o una serie de filas especificando el valor exacto que se va a buscar.

Compilación nativa de consultas JSON

Finalmente, la compilación nativa de procedimientos, funciones y desencadenadores de Transact-SQL que contienen consultas con funciones JSON mejora el rendimiento de las consultas y reduce los ciclos de CPU necesarios para ejecutar los procedimientos. En el ejemplo siguiente se muestra un procedimiento compilado de forma nativa que utiliza varias funciones: JSON\_VALUE, OPENJSON y JSON\_MODIFY.

SQLCopiar

CREATE PROCEDURE xtp.ProductList(@ProductIds nvarchar(100))

WITH SCHEMABINDING, NATIVE\_COMPILATION

AS BEGIN

ATOMIC WITH (transaction isolation level = snapshot, language = N'English')

SELECT ProductID,Name,Price,Data,Tags, JSON\_VALUE(data,'$.MadeIn') AS MadeIn

FROM xtp.Product

JOIN OPENJSON(@ProductIds)

ON ProductID = value

END;

CREATE PROCEDURE xtp.UpdateProductData(@ProductId int, @Property nvarchar(100), @Value nvarchar(100))

WITH SCHEMABINDING, NATIVE\_COMPILATION

AS BEGIN

ATOMIC WITH (transaction isolation level = snapshot, language = N'English')

UPDATE xtp.Product

SET Data = JSON\_MODIFY(Data, @Property, @Value)

WHERE ProductID = @ProductId;

END

Encuentre respuestas en este artículo a algunas preguntas habituales sobre la compatibilidad integrada de JSON en SQL Server.

Salida de FOR JSON y JSON

¿FOR JSON PATH o FOR JSON AUTO?

**Pregunta.** Deseo crear un resultado de texto JSON de una consulta SQL simple en una sola tabla. FOR JSON PATH y FOR JSON AUTO generan el mismo resultado. ¿Cuál de estas dos opciones debo usar?

**Respuesta.** Use FOR JSON PATH. Aunque no hay ninguna diferencia en la salida JSON, el modo AUTO aplica alguna lógica adicional que comprueba si se deben anidar columnas. Considere la posibilidad de usar PATH como la opción predeterminada.

Creación de una estructura anidada JSON

**Pregunta.** Desea generar texto JSON con varias matrices del mismo nivel. FOR JSON PATH puede crear objetos anidados con rutas de acceso y FOR JSON AUTO crea el nivel de anidamiento adicional para cada tabla. Ni una de estas dos opciones me permite generar el resultado deseado.¿Cómo puedo crear un formato JSON personalizado que no admitan directamente las opciones existentes?

**Respuesta.** Puede crear cualquier estructura de datos agregando consultas FOR JSON como expresiones de columna que devuelvan texto JSON.También puede crear manualmente JSON mediante la función JSON\_QUERY. El ejemplo siguiente muestra estas técnicas.

SQLCopiar

SELECT col1, col2, col3,

(SELECT col11, col12, col13 FROM t11 WHERE t11.FK = t1.PK FOR JSON PATH) as t11,

(SELECT col21, col22, col23 FROM t21 WHERE t21.FK = t1.PK FOR JSON PATH) as t21,

(SELECT col31, col32, col33 FROM t31 WHERE t31.FK = t1.PK FOR JSON PATH) as t31,

JSON\_QUERY('{"'+col4'":"'+col5+'"}' as t41

FROM t1

FOR JSON PATH

Cada resultado de una consulta FOR JSON o la función JSON\_QUERY en las expresiones de columna se formatea como un subobjeto JSON anidado independiente y se incluye en el resultado principal.

Prevención de texto JSON con caracteres de doble escape en el resultado FOR JSON

**Pregunta.** Tengo texto JSON almacenado en una columna de tabla. Quiero incluirlo en el resultado de FOR JSON. Pero FOR JSON convierte todos los caracteres JSON, por lo que obtengo una cadena JSON en lugar de un objeto anidado, tal como se muestra en el ejemplo siguiente.

SQLCopiar

SELECT 'Text' AS myText, '{"day":23}' AS myJson

FOR JSON PATH

Esta consulta genera el siguiente resultado.

JSONCopiar

[{"myText":"Text", "myJson":"{\"day\":23}"}]

¿Cómo puedo evitar este comportamiento? Quiero que se devuelva {"day":23} como un objeto JSON y no como texto con caracteres de escape.

**Respuesta.** Los textos JSON almacenados en una columna de texto o un literal se tratan como cualquier texto. Es decir, tiene acotado por comillas dobles y caracteres de escape. Si desea devolver un objeto JSON sin caracteres de escape, pasa la columna JSON como un argumento a la función JSON\_QUERY, tal como se muestra en el ejemplo siguiente.

SQLCopiar

SELECT col1, col2, col3, JSON\_QUERY(jsoncol1) AS jsoncol1

FROM tab1

FOR JSON PATH

JSON\_QUERY sin su segundo parámetro opcional devuelve solo el primer argumento como resultado. Puesto que JSON\_QUERY devuelve siempre un valor JSON válido, FOR JSON reconoce que este resultado no tiene que ser caracteres de escape.

Texto JSON generado con la cláusula WITHOUT\_ARRAY\_WRAPPER y con caracteres de escape en el resultado FOR JSON

**Pregunta.** Estoy tratando de dar formato a una expresión de columna mediante FOR JSON y la opción WITHOUT\_ARRAY\_WRAPPER.

SQLCopiar

SELECT 'Text' as myText,

(SELECT 12 day, 8 mon FOR JSON PATH, WITHOUT\_ARRAY\_WRAPPER) as myJson

FOR JSON PATH

Parece que al texto que devuelve la consulta FOR JSON se le aplican caracteres de escape como texto sin formato. Esto solo sucede si se especifica WITHOUT\_ARRAY\_WRAPPER. ¿Por qué no se trata como un objeto JSON y se incluye sin caracteres de escape en el resultado?

**Respuesta.** Si especifica la WITHOUT\_ARRAY\_WRAPPER opción en interna FOR JSON, el texto JSON resultante no es necesariamente válido JSON. Por lo tanto, el exterior FOR JSON se da por supuesto que esto es texto sin formato y convierte la cadena. Si está seguro de que el JSON de salida es válido, inclúyalo con la JSON\_QUERY función para promoverlo a correctamente con el formato JSON, tal como se muestra en el ejemplo siguiente.

SQLCopiar

SELECT 'Text' as myText,

JSON\_QUERY((SELECT 12 day, 8 mon FOR JSON PATH, WITHOUT\_ARRAY\_WRAPPER)) as myJson

FOR JSON PATH

Entrada de OPENJSON y JSON

Devolución de un subobjeto JSON anidado a partir de texto JSON con OPENJSON

**Pregunta.** No puedo abrir una matriz de objetos JSON complejos que contiene valores escalares, objetos, matrices y uso de OPENJSON con un esquema explícito. Cuando hago referencia a una clave en la cláusula WITH, se devuelven solo los valores escalares. Los objetos y las matrices se devuelven como valores NULL. ¿Cómo puedo extraer objetos o matrices como objetos JSON?

**Respuesta.** Si desea devolver un objeto o una matriz como una columna, utilice la opción AS JSON en la definición de columna, tal como se muestra en el ejemplo siguiente.

SQLCopiar

SELECT scalar1, scalar2, obj1, obj2, arr1

FROM OPENJSON(@json)

WITH ( scalar1 int,

scalar2 datetime2,

obj1 NVARCHAR(MAX) AS JSON,

obj2 NVARCHAR(MAX) AS JSON,

arr1 NVARCHAR(MAX) AS JSON)

Devolver el valor de texto largo con OPENJSON en lugar de JSON\_VALUE

**Pregunta.** Tengo una clave de descripción en JSON que contiene texto largo. JSON\_VALUE(@json, '$.description') devuelve NULL en lugar de un valor.

**Respuesta.** JSON\_VALUE se ha diseñado para devolver valores escalares de tamaño reducido. Generalmente, la función devuelve NULL en lugar de un error de desbordamiento. Si quiere que se devuelvan valores de mayor tamaño, utilice OPENJSON, que admite valores NVARCHAR(MAX), tal y como se muestra en el ejemplo siguiente.

SQLCopiar

SELECT myText FROM OPENJSON(@json) WITH (myText NVARCHAR(MAX) '$.description')

Gestión de claves duplicadas con OPENJSON en lugar de JSON\_VALUE

**Pregunta.** Tengo claves duplicadas en el texto JSON. JSON\_VALUE devuelve solo la primera clave que se encuentra en la ruta de acceso. ¿Cómo puedo devolver todas las claves que tengan el mismo nombre?

**Respuesta.** Las funciones escalares integradas de JSON devuelven solo la primera aparición del objeto de referencia. Si necesita más de una clave, utilice la función con valores de tabla OPENJSON, tal y como se muestra en el ejemplo siguiente.

SQLCopiar

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

WHERE [key] = 'color'

OPENJSON requiere el nivel de compatibilidad 130

**Pregunta.** Estoy tratando de ejecutar OPENJSON en SQL Server 2016 y obtengo el siguiente error.

Msg 208, Level 16, State 1 ‘Invalid object name OPENJSON’

**Respuesta.** La función OPENJSON solo está disponible en el nivel de compatibilidad 130. Si el nivel de compatibilidad de base de datos es inferior a 130, OPENJSON estará oculto. Hay otras funciones JSON que sí están disponibles en todos los niveles de compatibilidad.

Otras preguntas

Claves de referencia que contienen caracteres no alfanuméricos en texto JSON

**Pregunta.** Tengo caracteres no alfanuméricos en claves de mi texto JSON. ¿Cómo puedo hacer referencia a estas propiedades?

**Respuesta.** Tiene que incluirlas entre comillas en las rutas de acceso JSON. Por ejemplo, JSON\_VALUE(@json, '$."$info"."First Name".value').