CHAPTER 8

Shredding JSON Data

In Chapter 7, I discussed how we can convert relational data into JSON documents, but what if we had to shred a JSON document (just as you learned to shred XML documents in Chapter 4) into a relational dataset? We could achieve this by using the `OPENJSON()` function. The `OPENJSON()` function accepts a single JSON document as an input parameter and outputs a tabular result set. The `OPENJSON()` function can be called either with or without specifying an explicit schema for the result set. `OPENJSON()` also supports the use of JSON path expressions. This chapter will examine each of these options.

OPENJSON() with Default Schema

In order to understand how the `OPENJSON()` function works with the default schema, let's examine the CustomFields column of the Application.People table in the WideWorldImporters database. The query in Listing 8-1, returns the `PersonID` (the primary key of the table), the FullName column, and the CustomFields column, which contains a JSON document.

Tip Unlike the other data types discussed in this book, a JSON data type has not actually been created in SQL Server 2017. Instead, JSON documents are stored in NVARCHAR columns, and JSON-aware functions are called against the data, to parse and interact with the JSON.

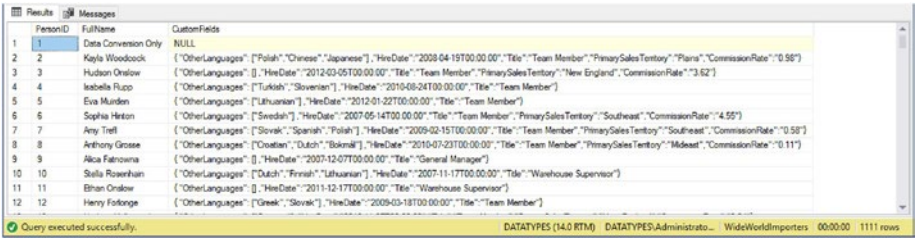
P. A. Carter, *SQL Server Advanced Data Types*,
https://doi.org/10.1007/978-1-4842-3901-8_8

Listing 8-1. Inspecting the Application.Person Table

```
USE WideWorldImporters
GO

SELECT
      PersonID
        , FullName
        , CustomFields
FROM Application.People ;
```

You will notice, from the partial result set shown in Figure 8-1, that the CustomFields column contains a JSON document specifying each person's properties, such as their hire data (for staff), languages spoken, and their title.



	PersonID	FullName	CustomFields
1	1	Data Conversion Only	NULL
2	2	Kayla Woodcock	{"OtherLanguages":["Polish","Chinese","Japanese"],"HireDate":"2008-04-19T00:00:00","Title":"Team Member","PrimarySalesTerritory":"Plains","CommissionRate":"0.98"}
3	3	Hudson Onslow	{"OtherLanguages":[],"HireDate":"2012-03-05T00:00:00","Title":"Team Member","PrimarySalesTerritory":"New England","CommissionRate":"3.62"}
4	4	Isabella Rupp	{"OtherLanguages":["Turkish","Slovenian"],"HireDate":"2010-08-24T00:00:00","Title":"Team Member"}
5	5	Eva Murden	{"OtherLanguages":["Lithuanian"],"HireDate":"2012-01-22T00:00:00","Title":"Team Member"}
6	6	Sophia Hinton	{"OtherLanguages":["Swedish"],"HireDate":"2007-05-14T00:00:00","Title":"Team Member","PrimarySalesTerritory":"Southeast","CommissionRate":"4.55"}
7	7	Amy Trefl	{"OtherLanguages":["Slovak","Spanish","Polish"],"HireDate":"2009-02-15T00:00:00","Title":"Team Member","PrimarySalesTerritory":"Southeast","CommissionRate":"0.58"}
8	8	Anthony Grosse	{"OtherLanguages":["Croatian","Dutch","Bokmål"],"HireDate":"2010-07-23T00:00:00","Title":"Team Member","PrimarySalesTerritory":"Mideast","CommissionRate":"0.11"}
9	9	Alica Fatnowna	{"OtherLanguages":[],"HireDate":"2007-12-07T00:00:00","Title":"General Manager"}
10	10	Stella Rosenhain	{"OtherLanguages":["Dutch","Finnish","Lithuanian"],"HireDate":"2007-11-17T00:00:00","Title":"Warehouse Supervisor"}
11	11	Ethan Onslow	{"OtherLanguages":[],"HireDate":"2011-12-17T00:00:00","Title":"Warehouse Supervisor"}
12	12	Henry Forlonge	{"OtherLanguages":["Greek","Slovak"],"HireDate":"2009-03-18T00:00:00","Title":"Team Member"}

Figure 8-1. *Results of inspecting the Application.Person table*

If we wanted to use `OPENJSON()` to shred the details of a specific user, we would first have to pass the JSON document into a variable, before passing the variable into the `OPENJSON()` function. This technique is demonstrated in Listing 8-2, which returns the custom fields for Anthony Grosse.

Listing 8-2. Shredding a Single JSON Document

```
USE WideWorldImporters
GO

DECLARE @CustomFields NVARCHAR(MAX) ;
```

```
SET @CustomFields = (
        SELECT CustomFields
        FROM Application.People
        WHERE FullName = 'Anthony Grosse'
) ;

SELECT *
FROM OPENJSON(@CustomFields) ;
```

The results of this script are illustrated in Figure 8-2.

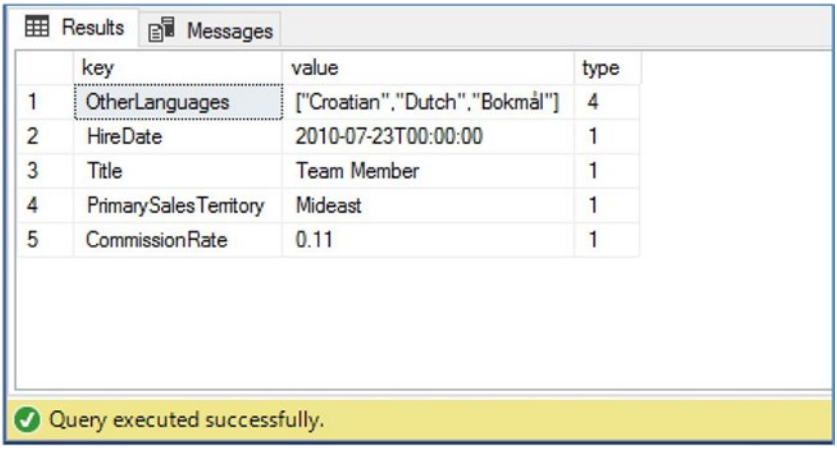


Figure 8-2. *Results of shredding a single JSON document*

The key column contains the name of the name/value pair; the value column contains the value of the name/value pair; and the type column indicates the data type. Table 8-1 details the data types that can be returned.

Table 8-1. *Data Types*

Data Type ID	Data Type
1	String
2	Number
3	Boolean
4	Array
5	Object

Shredding a Column

But what if we wanted to shred an entire column? The `OPENJSON()` function only accepts a single JSON object, so we could not pass in values from multiple rows. Instead, we would have to use the `OUTER APPLY` operator against the table.

The `OUTER APPLY` operator applies a function to every row in a result set. If the function returns a `NULL` value, the row will be included in the result set. This contrasts with the `CROSS APPLY` operator, which also applies a function to every row in a result set but omits the row, if the applied function returns `NULL`.

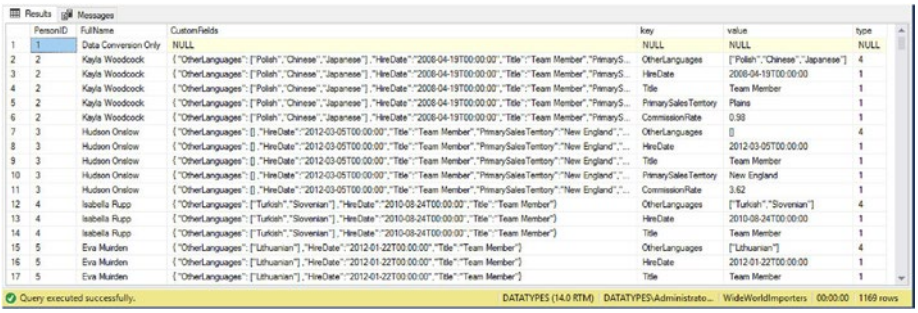
The query in Listing 8-3 demonstrates how to use the `OUTER APPLY` operator against the Application.People table, to shred the `CustomFields` document.

Listing 8-3. Using `OUTER APPLY` with `OPENJSON()`

```
USE WideWorldImporters
GO

SELECT PersonID, FullName, CustomFields, JSON.*
FROM Application.People
OUTER APPLY OPENJSON(CustomFields) JSON ;
```

Partial results from this query are shown in Figure 8-3.



	PersonID	FullName	CustomFields	key	value	type
1	1	Data Conversion Only	NULL	NULL	NULL	NULL
2	2	Kayla Woodcock	{"OtherLanguages":["Polish","Chinese","Japanese"],"HireDate":"2008-04-19T00:00:00","Title":"Team Member","PrimaryS...	OtherLanguages	["Polish","Chinese","Japanese"]	4
3	2	Kayla Woodcock	{"OtherLanguages":["Polish","Chinese","Japanese"],"HireDate":"2008-04-19T00:00:00","Title":"Team Member","PrimaryS...	HireDate	2008-04-19T00:00:00	1
4	2	Kayla Woodcock	{"OtherLanguages":["Polish","Chinese","Japanese"],"HireDate":"2008-04-19T00:00:00","Title":"Team Member","PrimaryS...	Title	Team Member	1
5	2	Kayla Woodcock	{"OtherLanguages":["Polish","Chinese","Japanese"],"HireDate":"2008-04-19T00:00:00","Title":"Team Member","PrimaryS...	PrimarySalesTerritory	Plains	1
6	2	Kayla Woodcock	{"OtherLanguages":["Polish","Chinese","Japanese"],"HireDate":"2008-04-19T00:00:00","Title":"Team Member","PrimaryS...	CommissionRate	0.98	1
7	3	Hudson Onslow	{"OtherLanguages":[],"HireDate":"2012-03-05T00:00:00","Title":"Team Member","PrimarySalesTerritory":"New England","...	OtherLanguages	[]	4
8	3	Hudson Onslow	{"OtherLanguages":[],"HireDate":"2012-03-05T00:00:00","Title":"Team Member","PrimarySalesTerritory":"New England","...	HireDate	2012-03-05T00:00:00	1
9	3	Hudson Onslow	{"OtherLanguages":[],"HireDate":"2012-03-05T00:00:00","Title":"Team Member","PrimarySalesTerritory":"New England","...	Title	Team Member	1
10	3	Hudson Onslow	{"OtherLanguages":[],"HireDate":"2012-03-05T00:00:00","Title":"Team Member","PrimarySalesTerritory":"New England","...	PrimarySalesTerritory	New England	1
11	3	Hudson Onslow	{"OtherLanguages":[],"HireDate":"2012-03-05T00:00:00","Title":"Team Member","PrimarySalesTerritory":"New England","...	CommissionRate	3.62	1
12	4	Isabella Rupp	{"OtherLanguages":["Turkish","Slovenian"],"HireDate":"2010-08-24T00:00:00","Title":"Team Member"}	OtherLanguages	["Turkish","Slovenian"]	4
13	4	Isabella Rupp	{"OtherLanguages":["Turkish","Slovenian"],"HireDate":"2010-08-24T00:00:00","Title":"Team Member"}	HireDate	2010-08-24T00:00:00	1
14	4	Isabella Rupp	{"OtherLanguages":["Turkish","Slovenian"],"HireDate":"2010-08-24T00:00:00","Title":"Team Member"}	Title	Team Member	1
15	5	Eva Murden	{"OtherLanguages":["Lithuanian"],"HireDate":"2012-01-22T00:00:00","Title":"Team Member"}	OtherLanguages	["Lithuanian"]	4
16	5	Eva Murden	{"OtherLanguages":["Lithuanian"],"HireDate":"2012-01-22T00:00:00","Title":"Team Member"}	HireDate	2012-01-22T00:00:00	1
17	5	Eva Murden	{"OtherLanguages":["Lithuanian"],"HireDate":"2012-01-22T00:00:00","Title":"Team Member"}	Title	Team Member	1

Figure 8-3. *Results of using* `OUTER APPLY` *with* `OPENJSON()`

You will notice that the results from the Application.Person table are duplicated for each row returned from the `OPENJSON()` function. This is known as a Cartesian product.

Tip If we had used `CROSS APPLY` instead of `OUTER APPLY`, the results for `PersonID 1` would have been omitted.

To turn this data into columns, to avoid rows being duplicated, you could use the `PIVOT` operator. The `PIVOT` operator works by rotating unique values from a column into separate columns. This could also be described as changing rows to columns. It will then perform aggregations on remaining columns, as required. The same could be achieved by using multiple `CASE` statements, but the `PIVOT` operator is far more efficient.

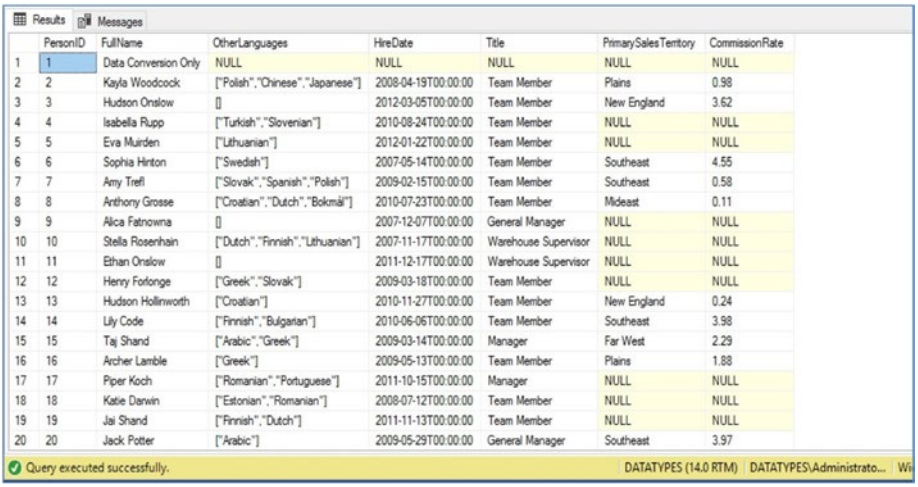
The syntax of the `PIVOT` operator has an outer query, followed by two subqueries. The first subquery contains the base query, while the second contains the pivot specification. Because our values are often textual, and aggregation isn't appropriate, we will use the `MAX()` aggregate function. This is demonstrated with the query in Listing 8-4.

Listing 8-4. Using PIVOT with OPENJSON()

```
USE WideWorldImporters
GO

SELECT
          PersonID
        , FullName
        , [OtherLanguages]
        , [HireDate]
        , [Title]
        , [PrimarySalesTerritory]
        , [CommissionRate]
FROM (
SELECT
          PersonID
        , FullName
        , JSON.[Key] AS JSONName
        , JSON.value AS JSONValue
FROM Application.People
OUTER APPLY OPENJSON(CustomFields) JSON
) Src
PIVOT
(
MAX(JSONValue)
FOR JSONName IN ([OtherLanguages], [HireDate], [Title],
[PrimarySalesTerritory], [CommissionRate])
) pvt ;
```

The partial results of this query can be seen in Figure 8-4.



Results Messages

	PersonID	FullName	OtherLanguages	HireDate	Title	PrimarySalesTerritory	CommissionRate
1	1	Data Conversion Only	NULL	NULL	NULL	NULL	NULL
2	2	Kayla Woodcock	["Polish","Chinese","Japanese"]	2008-04-19T00:00:00	Team Member	Plains	0.98
3	3	Hudson Onslow	[]	2012-03-05T00:00:00	Team Member	New England	3.62
4	4	Isabella Rupp	["Turkish","Slovenian"]	2010-08-24T00:00:00	Team Member	NULL	NULL
5	5	Eva Muirden	["Lithuanian"]	2012-01-22T00:00:00	Team Member	NULL	NULL
6	6	Sophia Hinton	["Swedish"]	2007-05-14T00:00:00	Team Member	Southeast	4.55
7	7	Amy Trefl	["Slovak","Spanish","Polish"]	2009-02-15T00:00:00	Team Member	Southeast	0.58
8	8	Anthony Grosse	["Croatian","Dutch","Bokmål"]	2010-07-23T00:00:00	Team Member	Mideast	0.11
9	9	Alica Fatnowna	[]	2007-12-07T00:00:00	General Manager	NULL	NULL
10	10	Stella Rosenhain	["Dutch","Finnish","Lithuanian"]	2007-11-17T00:00:00	Warehouse Supervisor	NULL	NULL
11	11	Ethan Onslow	[]	2011-12-17T00:00:00	Warehouse Supervisor	NULL	NULL
12	12	Henry Forlonge	["Greek","Slovak"]	2009-03-18T00:00:00	Team Member	NULL	NULL
13	13	Hudson Hollinworth	["Croatian"]	2010-11-27T00:00:00	Team Member	New England	0.24
14	14	Lily Code	["Finnish","Bulgarian"]	2010-06-06T00:00:00	Team Member	Southeast	3.98
15	15	Taj Shand	["Arabic","Greek"]	2009-03-14T00:00:00	Manager	Far West	2.29
16	16	Archer Lamble	["Greek"]	2009-05-13T00:00:00	Team Member	Plains	1.88
17	17	Piper Koch	["Romanian","Portuguese"]	2011-10-15T00:00:00	Manager	NULL	NULL
18	18	Katie Darwin	["Estonian","Romanian"]	2008-07-12T00:00:00	Team Member	NULL	NULL
19	19	Jai Shand	["Finnish","Dutch"]	2011-11-13T00:00:00	Team Member	NULL	NULL
20	20	Jack Potter	["Arabic"]	2009-05-29T00:00:00	General Manager	Southeast	3.97

Query executed successfully. DATATYPES (14.0 RTM) DATATYPES\Administrato... Wi

Figure 8-4. *Results of using* `PIVOT` *with* `OPENJSON()`

The limitation of using this approach is that you must know the name of each key in the JSON document before writing the query. If any key names are missed, or added later, the data will not appear in the result set. This can be particularly challenging, as JSON documents cannot be bound to a schema.

Dynamic Shredding Based on Document Content

The way to resolve the issue of not knowing the document contents at development time is to use a dynamic `PIVOT`. This involves using dynamic SQL to define the current list of JSON keys to pivot before the query is run. This technique is demonstrated in Listing 8-5.

Tip `QUOTENAME()` is a system function that delimits a value by wrapping it in square brackets.

Listing 8-5. Using Dynamic PIVOT with OPENJSON()

```
DECLARE @Columns NVARCHAR(MAX) ;
DECLARE @SQL NVARCHAR(MAX) ;

SET @Columns = '';

SELECT @Columns += ', p.' + QUOTENAME(JSONName)
FROM (
SELECT DISTINCT
        JSON.[Key] AS JSONName
FROM Application.People p
CROSS APPLY OPENJSON(CustomFields) JSON
) AS cols ;

SET @SQL =
'SELECT
          PersonID
        , FullName
        , ' + STUFF(@Columns, 1, 2, '') + '
FROM
(
  SELECT
          PersonID
        , FullName
        , JSON.[Key] AS JSONName
        , JSON.value AS JSONValue
FROM Application.People
OUTER APPLY OPENJSON(CustomFields) JSON
) AS src
PIVOT
```

```
(
  MAX(JSONValue) FOR JSONName IN ('
  + STUFF(REPLACE(@Columns, ', p.[', ',['), 1, 1, '')
  + ')
) AS p ;' ;

EXEC (@SQL) ;
```

OPENJSON( ) with Explicit Schema

When using `OPENJSON()` with an explicit schema, you are able to provide control over the format of the result set that is returned. Instead of a three-column result set, a column will be returned for every column that you have specified in the `WITH` clause. You can also specify the data type of each column. These data types are T-SQL data types, not JSON data types, so types such as `DATE` or `DECIMAL` can be specified. For example, consider the script in Listing 8-6.

Listing 8-6. Using `OPENJSON()` with an Explicit Schema

```
DECLARE @CustomFields NVARCHAR(MAX) ;

SET @CustomFields =
(
SELECT
        CustomFields
FROM Application.People
WHERE PersonID = 2
) ;

SELECT *
FROM OPENJSON(@CustomFields)
WITH (
```

```
          HireDate DATETIME2
        , Title NVARCHAR(50)
        , PrimarySalesTerritory NVARCHAR(50)
        , CommissionRate DECIMAL(5,2)
) ;
```

This query returns the results shown in Figure 8-5.

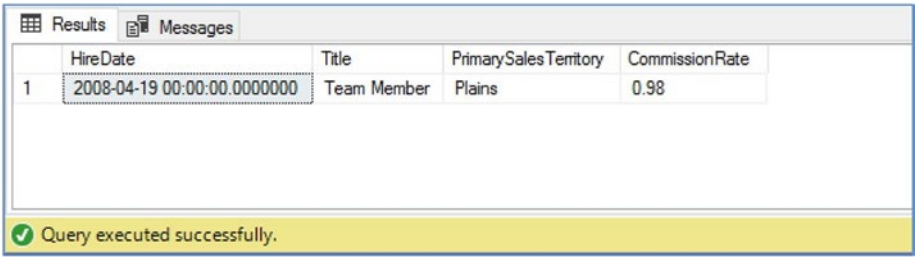


Figure 8-5. *Results of using* `OPENJSON()` *with an explicit schema*

A slight complexity arises when one of the columns returned is a JSON object. For example, consider the query in Listing 8-7, which adds the OtherLanguages column to the query. As there is no specific JSON data type, we will use `NVARCHAR(MAX)`, as it can be stored as `NVARCHAR(MAX)` in a table.

Listing 8-7. Adding a JSON Column

```
DECLARE @CustomFields NVARCHAR(MAX) ;

SET @CustomFields =
(
SELECT
        CustomFields
FROM Application.People
WHERE PersonID = 2
) ;
```

```
SELECT *
FROM OPENJSON(@CustomFields)
WITH (
          OtherLanguages NVARCHAR(MAX)
        , HireDate DATETIME2
        , Title NVARCHAR(50)
        , PrimarySalesTerritory NVARCHAR(50)
        , CommissionRate DECIMAL(5,2)
) ;
```

This query returns the results shown in Figure 8-6.

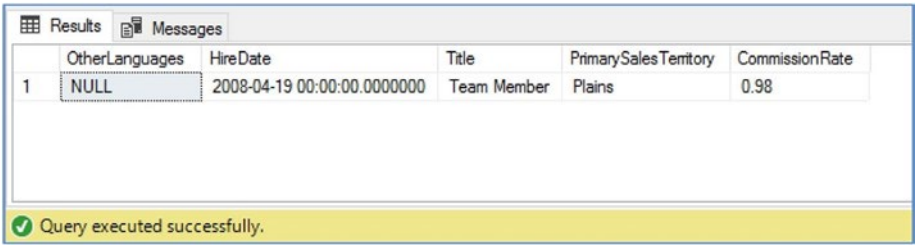


Figure 8-6. *Results of adding a JSON column*

So why has the OtherLanguages column returned `NULL`? We know that the column exists, and that it contains data for `PersonID 2`, owing to the previous examples in this chapter. When returning a JSON object from `OPENJSON()`, we must use additional syntax in the `WITH` clause, to specify that the `NVARCHAR` actually represents a JSON object, as demonstrated in Listing 8-8.

Listing 8-8. Correctly Returning a JSON Array or Object

```
DECLARE @CustomFields NVARCHAR(MAX) ;

SET @CustomFields =
(
```

```
SELECT
        CustomFields
FROM Application.People
WHERE PersonID = 2
) ;

SELECT *
FROM OPENJSON(@CustomFields)
WITH (
          OtherLanguages NVARCHAR(MAX) AS JSON
        , HireDate DATETIME2
        , Title NVARCHAR(50)
        , PrimarySalesTerritory NVARCHAR(50)
        , CommissionRate DECIMAL(5,2)
) ;
```

The script will now return the results that we expect, as shown in Figure 8-7.

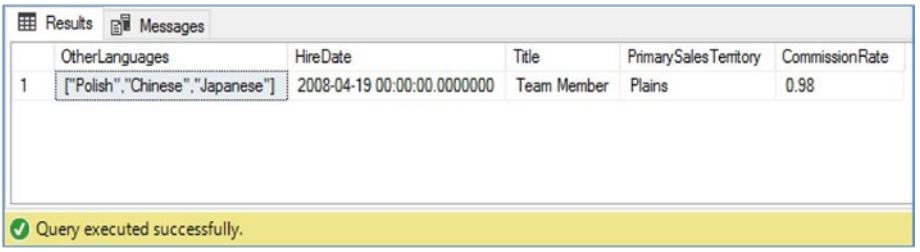


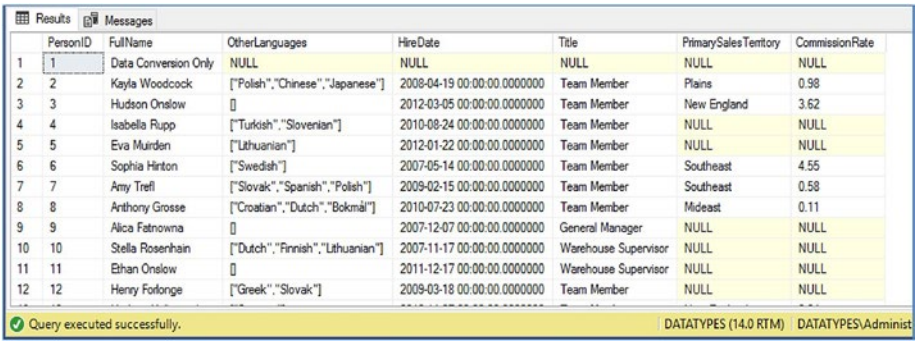
Figure 8-7. *Correctly returning JSON data*

When you must shred multiple rows, an explicit schema can also be specified, when using the `OUTER APPLY` operator. Remember that the `OUTER APPLY` operator will not remove rows that return `NULL` values, in the way that `CROSS APPLY` does. This is demonstrated in Listing 8-9.

Listing 8-9. Using an Explicit Schema with `OUTER APPLY`

```
SELECT
          PersonID
        , FullName
        , JSON.*
FROM Application.People
OUTER APPLY OPENJSON(CustomFields)
                                WITH (
                                        OtherLanguages
                                        NVARCHAR(MAX) AS JSON
                                      , HireDate DATETIME2
                                      , Title NVARCHAR(50)
                                      , PrimarySalesTerritory
                                      NVARCHAR(50)
                                      , CommissionRate
                                      DECIMAL(5,2)
                                ) JSON ;
```

As you can see from the partial results in Figure 8-8, some of the need-to-pivot data has been eliminated. The issue remains, however, that you must know every possible key in the JSON document before the query is written. Therefore, if there is not a discrete set of possible values, you may still be required to use dynamic SQL.



Results | Messages

	PersonID	FullName	OtherLanguages	HireDate	Title	PrimarySalesTerritory	CommissionRate
1	1	Data Conversion Only	NULL	NULL	NULL	NULL	NULL
2	2	Kayla Woodcock	["Polish","Chinese","Japanese"]	2008-04-19 00:00:00.0000000	Team Member	Plains	0.98
3	3	Hudson Onslow	[]	2012-03-05 00:00:00.0000000	Team Member	New England	3.62
4	4	Isabella Rupp	["Turkish","Slovenian"]	2010-08-24 00:00:00.0000000	Team Member	NULL	NULL
5	5	Eva Muirden	["Lithuanian"]	2012-01-22 00:00:00.0000000	Team Member	NULL	NULL
6	6	Sophia Hinton	["Swedish"]	2007-05-14 00:00:00.0000000	Team Member	Southeast	4.55
7	7	Amy Trefl	["Slovak","Spanish","Polish"]	2009-02-15 00:00:00.0000000	Team Member	Southeast	0.58
8	8	Anthony Grosse	["Croatian","Dutch","Bokmål"]	2010-07-23 00:00:00.0000000	Team Member	Mideast	0.11
9	9	Alica Fatnowna	[]	2007-12-07 00:00:00.0000000	General Manager	NULL	NULL
10	10	Stella Rosenhain	["Dutch","Finnish","Lithuanian"]	2007-11-17 00:00:00.0000000	Warehouse Supervisor	NULL	NULL
11	11	Ethan Onslow	[]	2011-12-17 00:00:00.0000000	Warehouse Supervisor	NULL	NULL
12	12	Henry Forlonge	["Greek","Slovak"]	2009-03-18 00:00:00.0000000	Team Member	NULL	NULL

Query executed successfully. DATATYPES (14.0 RTM) DATATYPES\Administ

Figure 8-8. *Results of using an explicit schema with OUTER APPLY*

OPENJSON( ) with Path Expressions

As well as the use of explicit schema, `OPENJSON()` also supports JSON path expressions. A path expression allows you to reference specific properties within a JSON document. For example, consider the JSON document in Listing 8-10.

Tip You may recognize this document, as we created it in Chapter 7.

Listing 8-10. Sales Orders with Root Node

```
{
   "SalesOrders": [
      {
         "OrderID": 72646,
         "CustomerID": 1060,
         "SalespersonPersonID": 14,
         "OrderDate": "2016-05-18"
      },
```

```
        {
            "OrderID": 72738,
            "CustomerID": 1060,
            "SalespersonPersonID": 14,
            "OrderDate": "2016-05-19"
        },
        {
            "OrderID": 72916,
            "CustomerID": 1060,
            "SalespersonPersonID": 6,
            "OrderDate": "2016-05-20"
        },
        {
            "OrderID": 73081,
            "CustomerID": 1060,
            "SalespersonPersonID": 8,
            "OrderDate": "2016-05-24"
        }
    ]
}
```

If we used a basic `OPENJSON()` statement against this document, it would return the entire `SalesOrders` array, as partially shown in Figure 8-9.

Figure 8-9. *Results of basic `OPENJSON()`*

If we were to use a PATH statement, however, we could choose to only return the *n*th item in this array. This would drastically alter the results set, as OPENJSON() would be able to map each item within the array element to a relational column, meaning that a row for each key within the element would be returned, instead of a single row containing a JSON document, as shown in Figure 8-10, which contains the results of shredding the first array element (OrderID 72646).

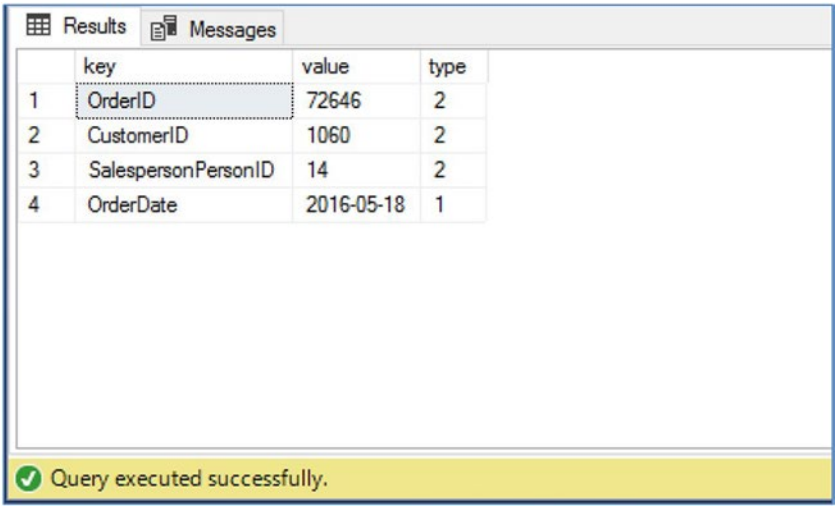


Figure 8-10. *Results of shredding a single array element*

So, let's look at how we can get to this result. First, we must understand that path expressions can be run in one of two modes: strict or lax. If you run a path expression in lax mode, and the path expression contains an error, OPENJSON() will "eat the error" and return an empty result set. If you use strict mode, however, if the path expression contains an error, OPENJSON() will throw an error message.

We now must understand the elements of the path itself. First, we use a $ to specify the context, followed by dot-separated, nested key names. Finally, we specify the array element number in square brackets. So, to produce the results in Figure 8-10, we would use the query in Listing 8-11.

Listing 8-11. Using Path Expressions to Return a Single Array Element

```
DECLARE @JSON NVARCHAR(MAX) ;

SET @JSON = '{
   "SalesOrders": [
      {
         "OrderID": 72646,
         "CustomerID": 1060,
         "SalespersonPersonID": 14,
         "OrderDate": "2016-05-18"
      },
      {
         "OrderID": 72738,
         "CustomerID": 1060,
         "SalespersonPersonID": 14,
         "OrderDate": "2016-05-19"
      },
      {
         "OrderID": 72916,
         "CustomerID": 1060,
         "SalespersonPersonID": 6,
         "OrderDate": "2016-05-20"
      },
      {
         "OrderID": 73081,
         "CustomerID": 1060,
         "SalespersonPersonID": 8,
         "OrderDate": "2016-05-24"
      }
   ]
}
' ;
```

```
SELECT *
FROM OPENJSON(@JSON, 'lax $.SalesOrders[0]') ;
```

You will notice, in this script, that after passing in the JSON document, we use the lax (or, alternatively, strict) keyword to specify the mode we will use. After a space comes the path expression itself. Here, we start with $, to set the context, and then point to the `SalesOrders` key. We then use square brackets to specify the array element that we wish to use.

Tip JSON path expressions always use zero-base arrays.

Shredding Data into Tables

You can now imagine how simple looping techniques could be used to shred each element within an array. For example, consider the script in Listing 8-12. This script will shred each of the array elements into a temporary table called Orders.

Listing 8-12. Shredding Each Element into a Temporary Table

```
DECLARE @JSON NVARCHAR(MAX) ;

SET @JSON = '{
   "SalesOrders": [
      {
         "OrderID": 72646,
         "CustomerID": 1060,
         "SalespersonPersonID": 14,
         "OrderDate": "2016-05-18"
      },
```

```
      {
         "OrderID": 72738,
         "CustomerID": 1060,
         "SalespersonPersonID": 14,
         "OrderDate": "2016-05-19"
      },
      {
         "OrderID": 72916,
         "CustomerID": 1060,
         "SalespersonPersonID": 6,
         "OrderDate": "2016-05-20"
      },
      {
         "OrderID": 73081,
         "CustomerID": 1060,
         "SalespersonPersonID": 8,
         "OrderDate": "2016-05-24"
      }
   ]
}
' ;

CREATE TABLE #Orders
(
        OrderID                 INT,
        CustomerID              INT,
        SalespersonPersonID     INT,
        OrderDate               DATE
) ;
```

```
DECLARE @ArrayElement INT = 0 ;

DECLARE @path NVARCHAR(MAX) = 'lax $.SalesOrders[' + CAST(
@ArrayElement AS NVARCHAR) + ']' ;

WHILE @ArrayElement <=3
BEGIN
         INSERT INTO #Orders (OrderID, CustomerID,
         SalespersonPersonID, OrderDate)
         SELECT
                  OrderID
                , CustomerID
                , SalespersonPersonID
                , OrderDate
         FROM OPENJSON(@JSON, @Path)
         WITH( OrderID INT, CustomerID INT, SalespersonPersonID
         INT, OrderDate DATE) ;

         SET @ArrayElement = @ArrayElement + 1 ;

         SET @path = 'lax $.SalesOrders[' + CAST(@ArrayElement
         AS NVARCHAR) + ']' ;
END

SELECT * FROM #Orders ;

DROP TABLE #Orders ;
```

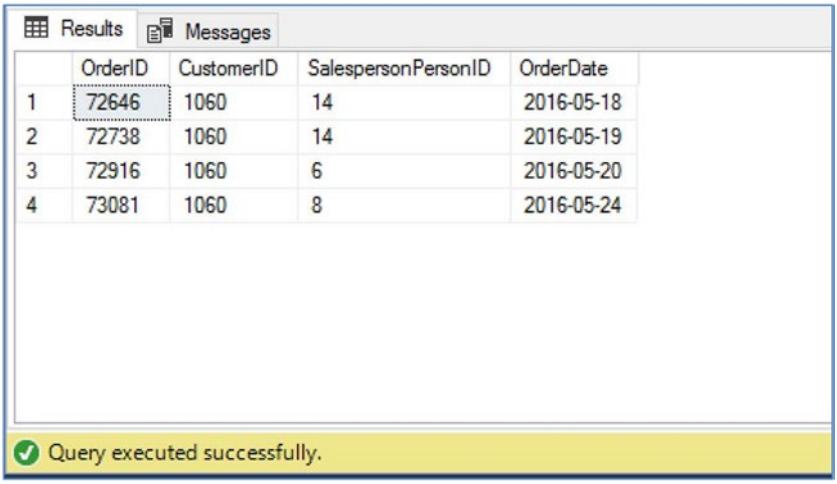


Figure 8-11. *Results of shredding multiple array elements*

The final `SELECT` statement in this script produces the results illustrated in Figure 8-11.

Caution While I have used a `WHILE` loop in this example, I have done so only because it provides a clear and easy example of how path expressions can be used. I would never use a `WHILE` loop or `CURSOR` in production code. There is always a way to achieve the same results, using a set-based approach.

Summary

JSON data can be shredded into tabular results sets by using the `OPENJSON()` function. `OPENJSON()` can be used either with or without an explicit

schema. When a schema is not explicitly defined, `OPENJSON()`, using a `WITH` clause, returns a standard row set, detailing the key (name), value, and JSON data type ID of each node in the document.

When an explicit schema is supplied, `OPENJSON()` will return a formatted result set, which contains a column for each specified in the `WITH` clause. Using an explicit schema avoids the need to pivot the data when you know every node in the document at development time. If the list of columns is not discrete, however, dynamic SQL will be required, to build a list of possible results before processing.

`OPENJSON()` also supports JSON path expressions. Passing path expressions to the function allows you to navigate to a specific item within an array, meaning that you can shred data to a more granular level. For example, instead of shredding an array of JSON objects into a table, you can use looping methodologies to shred the contents of each array element into relational data.