

Perform complex queries with JSON data

Note In this reading you can see the steps involved in the process of performing complex queries with JSON data.

We now need to unpack the sales order line-item data that is contained within an array embedded in the details column of our SalesOrders table to access the unit and revenue information that it contains. Luckily, Spark SQL has two functions designed to assist us with this problem:

- `explode()`, which separates the elements of array into multiple rows and uses the default column name `col` for elements of the array.
- `posexplode()`, which separates the elements of array into multiple rows with positions, and uses the column name `pos` for position, and `col` for elements of the array.

Note There are similar `explode()` and `posexplode()` functions in PySpark.

Firstly, let's remind ourselves what the data we are interested in looks like in the current SalesOrders table:

1. Paste the below code into a **new cell (A)**, click the **run cell** button.

```
%%sql
SELECT _id.string as SalesOrderId, details
FROM SalesOrders
LIMIT 10
```

1
2
3
4

Microsoft Azure | Synapse Analytics | synapseinkadventureworks

Synapse live | Validate all | Publish all

Cosmos DB Notebook

Attach to adventurespark | Language: PySpark (Python) | Preview Features

Ready

```

1 %%sql
2 SELECT _id.string as SalesOrderId, details
3 FROM SalesOrders
4 LIMIT 10
5

```

Command executed in 8s 129ms

Job execution Succeeded Spark 2 executors 16 cores

View in monitoring | Open Spark UI

View: Table | Chart

SalesOrderId	details
FCD4CA63-4B85-4FE1-AAD9-CE81E2951421	▶ [{"schema":{"name":"array","dataType":"elementTyp..."
FD0E241C-9862-4E65-8BF0-89E6688A3FA8	▶ [{"schema":{"name":"array","dataType":"elementTyp..."
FACBCB8A-6F65-4750-BA2E-DDEB346138BC	▶ [{"schema":{"name":"array","dataType":"elementTyp..."
FB155795-9C37-4EE1-8F3E-42C7787F4187	▶ [{"schema":{"name":"array","dataType":"elementTyp..."
F9C5558A-1793-45CD-9DD0-CF41D7808EF7	▶ [{"schema":{"name":"array","dataType":"elementTyp..."
FA218538-3538-4DB5-93EE-3AEDD01608F0	▶ [{"schema":{"name":"array","dataType":"elementTyp..."
F88E4221-185A-4418-9D7A-D09196C90E2A	▶ [{"schema":{"name":"array","dataType":"elementTyp..."
F7E7E441-977F-48FD-802E-AE4D6FCA8868	▶ [{"schema":{"name":"array","dataType":"elementTyp..."
F6C9FE0F-30B1-431B-9549-40EDA5802CAA	▶ [{"schema":{"name":"array","dataType":"elementTyp..."
F7248012-9F3D-4498-8C44-8F82853867BF	▶ [{"schema":{"name":"array","dataType":"elementTyp..."

Using SparkSQL query to explore data in a notebook

Here we can see our details column contains array data that needs to be expanded. Each element of the embedded array represents an individual line item of the sales order parent record the ID of which is contained in the `_id` column of the SalesOrders table.

We can expand this array using the `explode()` function:

2. Paste the below code into a **new cell (B)**, and click the **run cell** button.

```

%%sql
SELECT _id.string as SalesOrderId, explode(details.array)
FROM SalesOrders
LIMIT 10

```

1
2
3
4

The screenshot shows a Microsoft Azure Synapse Analytics notebook interface. The top navigation bar includes 'Microsoft Azure', 'Synapse Analytics', and 'synapselinkadventureworks'. The notebook is titled 'Cosmos DB Notebook'. The code cell contains the following SQL query:

```
1 %%sql
2 SELECT _id.string as SalesOrderId, explode(details.array)
3 FROM SalesOrders
4 LIMIT 10
```

The query has been executed, and the results are displayed in a table view. The table has two columns: 'SalesOrderId' and 'col'. The 'col' column contains JSON arrays representing the exploded details. The status bar indicates 'Job execution Succeeded' and 'Spark 2 executors 16 cores'.

SalesOrderId	col
F9C5558A-1793-45CD-9DD0-CF41D7B08EF7	["schema":{"name":"object","dataType":{"name":"s..."
F9C5558A-1793-45CD-9DD0-CF41D7B08EF7	["schema":{"name":"object","dataType":{"name":"s..."
FB155795-9C37-4EE1-8F3E-42C7787F4187	["schema":{"name":"object","dataType":{"name":"s..."
FB155795-9C37-4EE1-8F3E-42C7787F4187	["schema":{"name":"object","dataType":{"name":"s..."
FACBC88A-6F65-4750-BA2E-DEB3461388C	["schema":{"name":"object","dataType":{"name":"s..."
FD0E241C-9862-4E65-8BF0-89E6688A3FA8	["schema":{"name":"object","dataType":{"name":"s..."
FD0E241C-9862-4E65-8BF0-89E6688A3FA8	["schema":{"name":"object","dataType":{"name":"s..."
FCD4CA63-4B85-4FE1-AAD9-CE81E2951421	["schema":{"name":"object","dataType":{"name":"s..."
FCD4CA63-4B85-4FE1-AAD9-CE81E2951421	["schema":{"name":"object","dataType":{"name":"s..."
FCD4CA63-4B85-4FE1-AAD9-CE81E2951421	["schema":{"name":"object","dataType":{"name":"s..."

Exploding data in a notebook

We see that we get a new row for each element of the array, so each SalesOrderId in the SalesOrders table is now represented by one or more rows of line item detail, however we are unable to identify the line item row uniquely (without making some assumptions about other properties of the document, which is always dangerous).

So, let's try using the `posexplode()` function, by performing the following step:

3. Paste the below code into a **new cell (C)**, and click the **run cell** button.

```
%%sql
SELECT _id.string as SalesOrderId, posexplode(details.array)
FROM SalesOrders
LIMIT 10
```

1
2
3
4

Microsoft Azure | Synapse Analytics | synapseinkadventureworks

Synapse live | Validate all | Publish all

Cosmos DB Notebook

Attach to: adventurespark | Language: PySpark (Python) | Preview Features

Ready

```

1 %%sql
2 SELECT _id.string as SalesOrderId, posexplode(details.array)
3 FROM SalesOrders
4 LIMIT 10
5

```

Command executed in 6s 103ms

Job execution Succeeded Spark 2 executors 16 cores

View in monitoring | Open Spark UI

View: Table | Chart

SalesOrderId	pos	col
FCD4CA63-4B85-4FE1-AAD9-CE81E2951421	0	["schema":{"name":"object","dataType":{"name":"s..."
FCD4CA63-4B85-4FE1-AAD9-CE81E2951421	1	["schema":{"name":"object","dataType":{"name":"s..."
FCD4CA63-4B85-4FE1-AAD9-CE81E2951421	2	["schema":{"name":"object","dataType":{"name":"s..."
FD0E241C-9862-4E65-8BF0-89E6688A3FA8	0	["schema":{"name":"object","dataType":{"name":"s..."
FD0E241C-9862-4E65-8BF0-89E6688A3FA8	1	["schema":{"name":"object","dataType":{"name":"s..."
FACBCB8A-6F65-4750-BA2E-DD0B3461388C	0	["schema":{"name":"object","dataType":{"name":"s..."
FB155795-9C37-4EE1-8F3E-42C7787F4187	0	["schema":{"name":"object","dataType":{"name":"s..."
FB155795-9C37-4EE1-8F3E-42C7787F4187	1	["schema":{"name":"object","dataType":{"name":"s..."
F9C5558A-1793-45CD-9DD0-CF41D7B08EF7	0	["schema":{"name":"object","dataType":{"name":"s..."
F9C5558A-1793-45CD-9DD0-CF41D7B08EF7	1	["schema":{"name":"object","dataType":{"name":"s..."

Posexplode data in a notebook

Notice that we again get back a new row for each element of the array, so each SalesOrderId in the SalesOrders table is now represented by one or more rows of line item detail. However, the pos column now uniquely identifies the item detail and the order in which it appears in the array. We also note that the pos value is 0 based, and Adventure Works by convention numbers its line items starting at 1, so we will need to fix that as we use this data.

Let's create a SalesOrderDetailsView that encapsulates this sales order line item information for future use:

4. Paste the below code into a **new cell (D)**, and click the **run cell** button.

```

%%sql
CREATE OR REPLACE TEMPORARY VIEW SalesOrderDetailsView
AS
SELECT Ax.SalesOrderId,

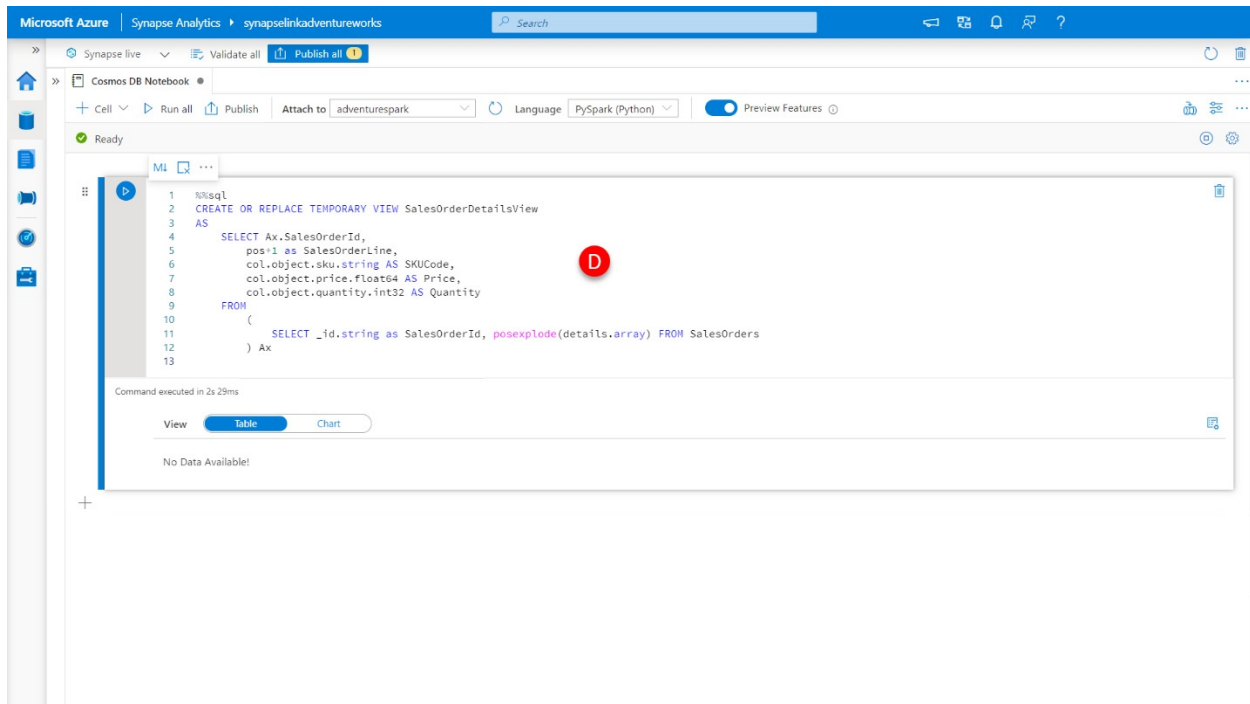
```

1
2
3
4
5
6
7
8
9
10
11
12

```

pos+1 as SalesOrderLine,
col.object.sku.string AS SKUCode,
col.object.price.float64 AS Price,
col.object.quantity.int32 AS Quantity
FROM
(
    SELECT _id.string as SalesOrderId, posexplode(details.array) FROM Sa
lesOrders
) Ax

```



Create a SalesOrderDetailsView-temporary-view

In this view, we have wrapped the inner subquery that uses the `posexplode()` function to extract the content of the array with some additional projection specifically projecting the `col.object.sku.string`, `col.object.price.float64` and `col.object.quantity.int32` values as `SKUCode`, `Price` and `Quantity` respectively, remembering to include their respective data type suffixes when accessing their values.

We have also incremented the `pos` value by 1 before projecting it into the `SalesOrderLine` column of the result set to consider the Adventure Works uses order line numbers starting at 1 by convention.

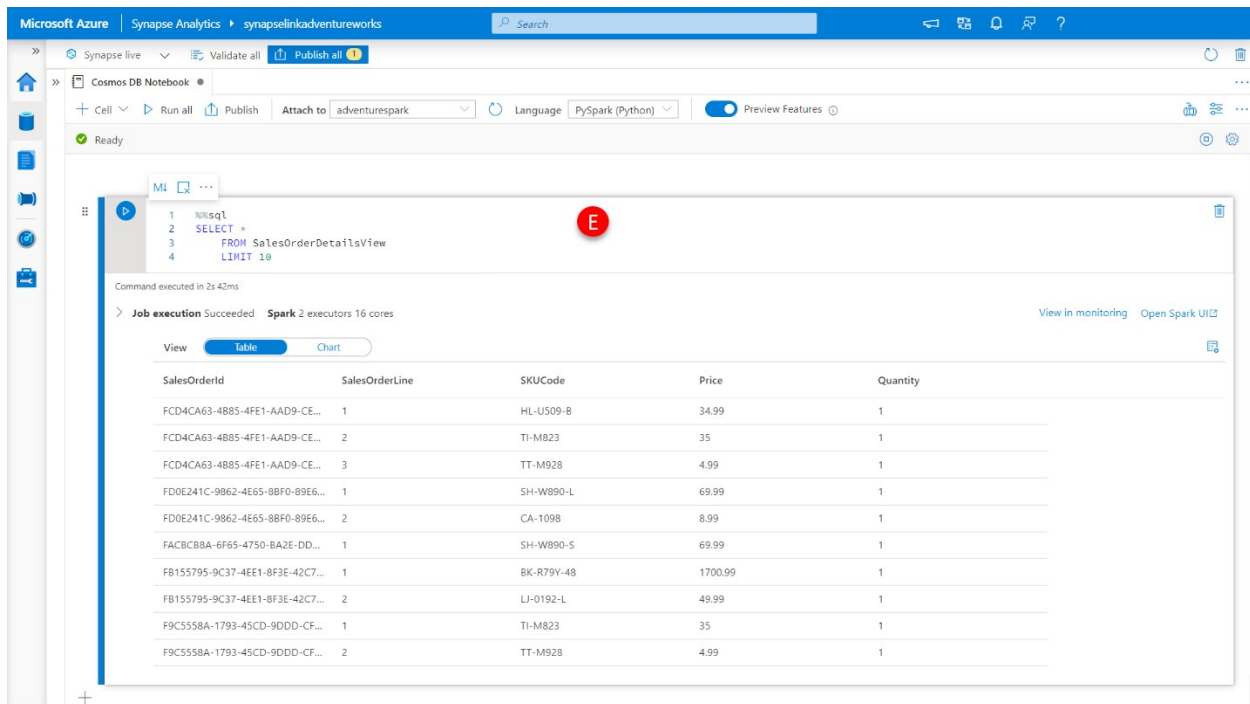
If we want to see the result of this view, we can:

5. Paste the below code into a **new cell (E)**, and click the **run cell** button.

```
SELECT *
```

1
2
3

FROM SalesOrderDetailsView
LIMIT 10



Microsoft Azure Synapse Analytics - synapseinkadventureworks

Ready

```
1 SELECT
2 FROM SalesOrderDetailsView
3 LIMIT 10
```

Command executed in 2s 42ms

Job execution Succeeded Spark 2 executors 16 cores

View in monitoring Open Spark UI

SalesOrderId	SalesOrderLine	SKUCode	Price	Quantity
FCD4CA63-4B85-4FE1-AAD9-CE...	1	HL-U509-B	34.99	1
FCD4CA63-4B85-4FE1-AAD9-CE...	2	TI-M823	35	1
FCD4CA63-4B85-4FE1-AAD9-CE...	3	TT-M928	4.99	1
FD0E241C-9862-4E65-8BF0-89E6...	1	SH-W890-L	69.99	1
FD0E241C-9862-4E65-8BF0-89E6...	2	CA-1098	8.99	1
FACBCB8A-6F65-4750-BA2E-DD...	1	SH-W890-S	69.99	1
FB155795-9C37-4EE1-8F3E-42C7...	1	BK-R79Y-48	1700.99	1
FB155795-9C37-4EE1-8F3E-42C7...	2	LJ-0192-L	49.99	1
F9C5558A-1793-45CD-9DDD-CF...	1	TI-M823	35	1
F9C5558A-1793-45CD-9DDD-CF...	2	TT-M928	4.99	1

Query the SalesOrderDetailsView-temporary-view

And as you can see, we now get back a result set with each sales order line individually represented in a row with the SalesOrderId and SalesOrderLine uniquely identifying it along with the associated SKUCode, price, and quantity information.

If we want to validate the schema of the SalesOrderDetailsView, we can:

6. Paste the below code into a **new cell (F)**, and click the **run cell** button.

1

DESCRIBE SalesOrderDetailsView

The screenshot shows the Microsoft Azure Synapse Analytics interface. At the top, the breadcrumb navigation indicates the path: Synapse Analytics > synapselinkadventureworks. Below this, the notebook environment is titled 'Cosmos DB Notebook'. The interface includes a toolbar with options like 'Cell', 'Run all', 'Publish', 'Attach to', 'Language' (set to PySpark (Python)), and 'Preview Features'. The notebook content shows a SQL command executed in 2s 40ms:

```
1 --sql
2 DESCRIBE SalesOrderDetailsView
```

The results are displayed in a table view, showing the schema of the SalesOrderDetailsView. The table has three columns: col_name, data_type, and comment.

col_name	data_type	comment
SalesOrderId	string	null
SalesOrderLine	int	null
SKUCode	string	null
Price	double	null
Quantity	int	null

Using the describe function

As you can see the data types for the SKUCode, Price and Quantity were automatically mapped back to the underlying data types contained within the array without the need to manually infer data types for these columns.

Now that we have our sales order details information in an easy-to-use format, we should easily be able to extract the insights Adventure Works set out to gather. This is the subject of the next item.