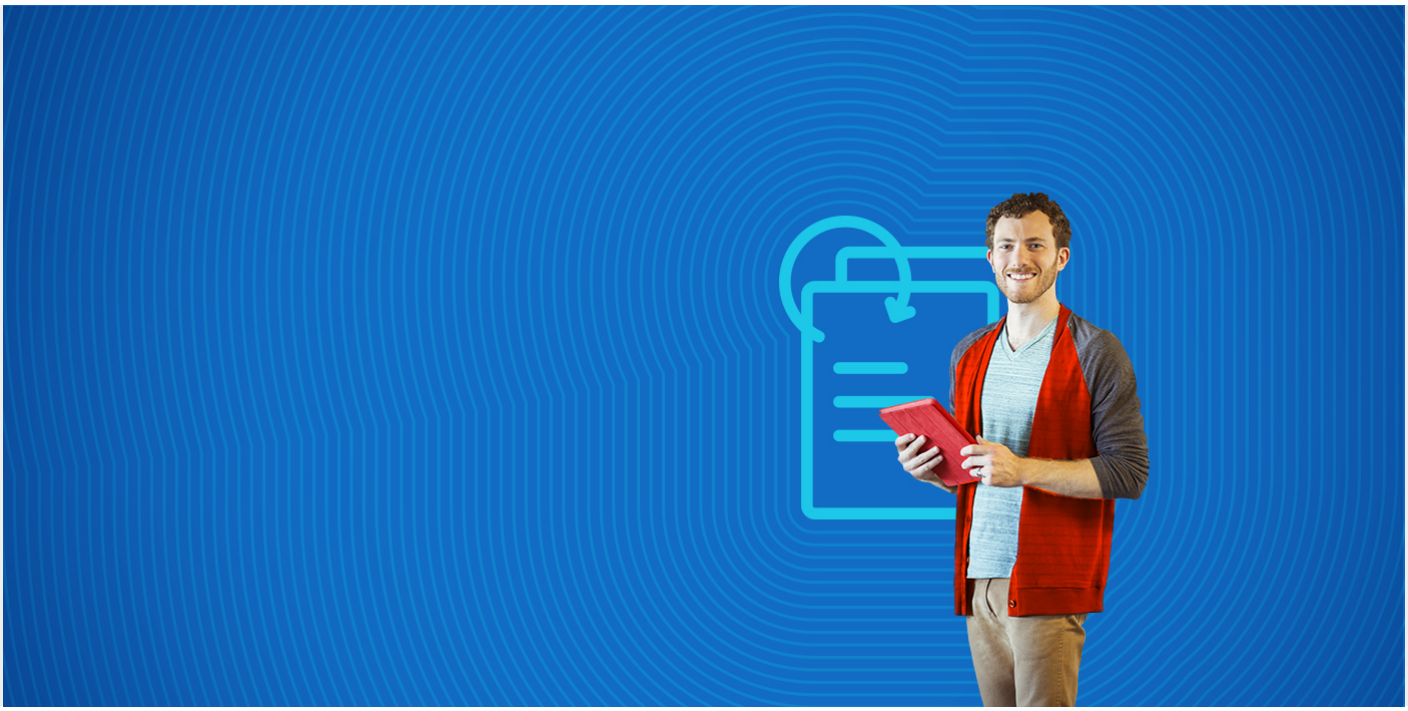
HOW TO QUERY JSON DATA IN SNOWFLAKE



We've already showed you how to create a variant column in a Snowflake table, where **variant** means JSON. In this tutorial, we show how to query those JSON columns.

Create a table with a JSON column

First create a database or use the **inventory** one we created in the last post and then create a table with one column of type **variant**:

```
use database inventory;
create table jsonRecord(jsonRecord variant);
```

Add JSON data to Snowflake

Then, add some data. We will add simple JSON, nested JSON, and JSON arrays (i.e. JSON objects inside brackets []) to show how to query each type. Notice the parse_json() function.

```
INSERT INTO JSONRECORD (jsonrecord) select PARSE_JSON('{"customer":
"Walker"}');
INSERT INTO JSONRECORD (jsonrecord) select PARSE_JSON('{"customer":
"Stephen"}');
INSERT INTO JSONRECORD (jsonrecord) select PARSE_JSON('{"customer":
"Aphrodite", "age": 32}');
```

These records include a JSON array, **orders**.

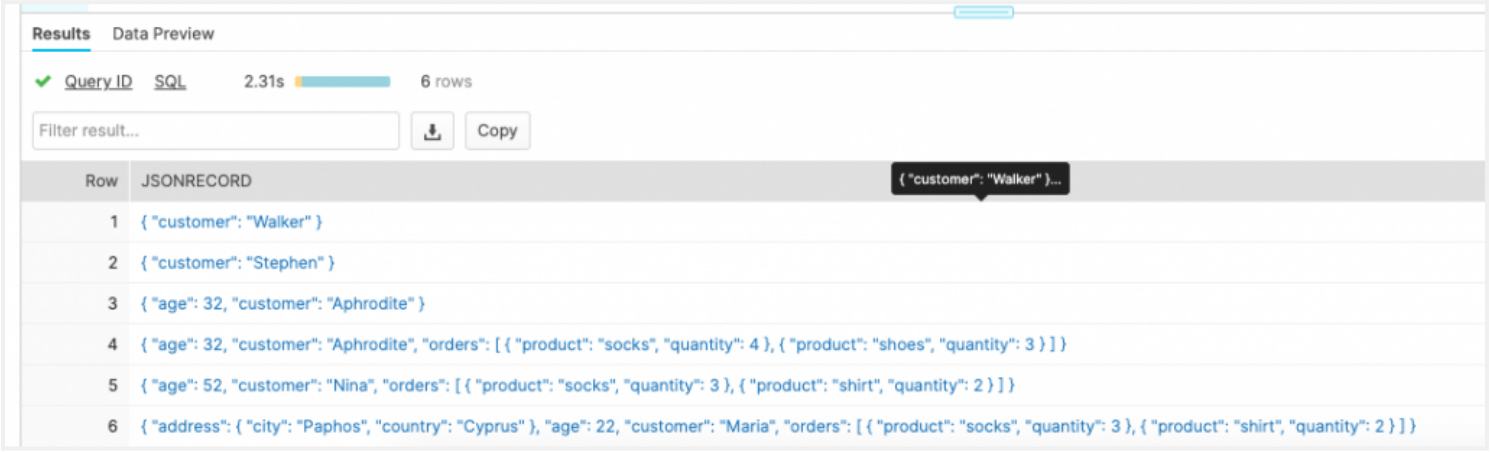
```
i
INSERT INTO JSONRECORD (jsonrecord) select PARSE_JSON(' {
            "customer": "Aphrodite",
            "age": 32,
            "orders":
 }');
INSERT INTO JSONRECORD (jsonrecord) select PARSE_JSON(' {
            "customer": "Nina",
            "age": 52,
            "orders":
 }');
```

This record includes nested JSON, meaning an attribute, **address**, whose value is another JSON object.

```
INSERT INTO JSONRECORD (jsonrecord) select PARSE_JSON(' {
            "customer": "Maria",
            "age": 22,
     "address" : { "city": "Paphos", "country":
"Cyprus"},
            "orders":
 }');
```

Now key **select * from JSONRECORD** to show all the records. Note that these are case-sensitive:

- Function
- Column
- Table names



How to select JSON data in Snowflake

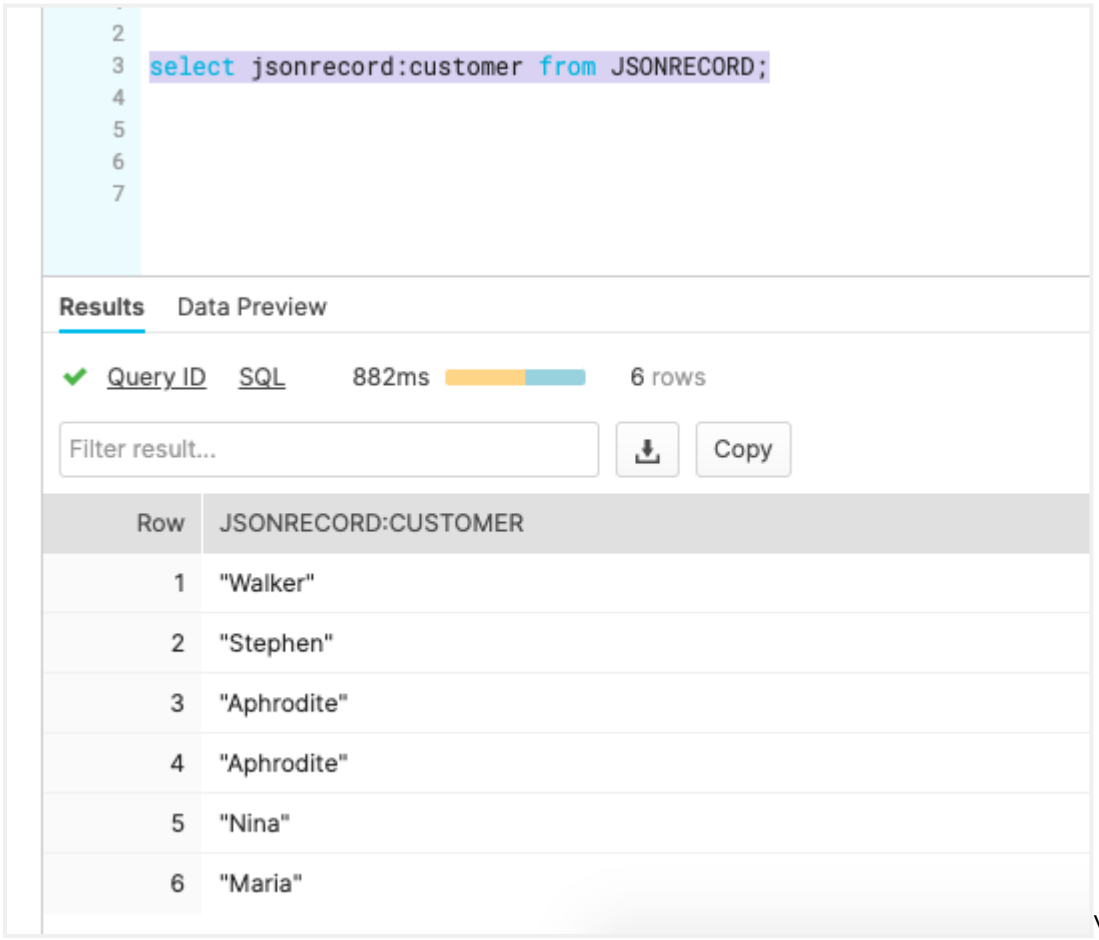
The format for selecting data includes all of the following:

- tableName:attribute
- tableName.attribute.JsonKey
- tableName.attribute.JsonKey
- tableName.attribute
- get_path(tableName, attribute)

Here we select the customer key from the JSON record. In JSON we call the items key value pairs, like: **{"key": "value"}**.

```
select jsonrecord:customer from JSONRECORD;
```

look like this:



We can also use the get_path() function:

```
select get_path(jsonrecord, 'address') from JSONRECORD;
```

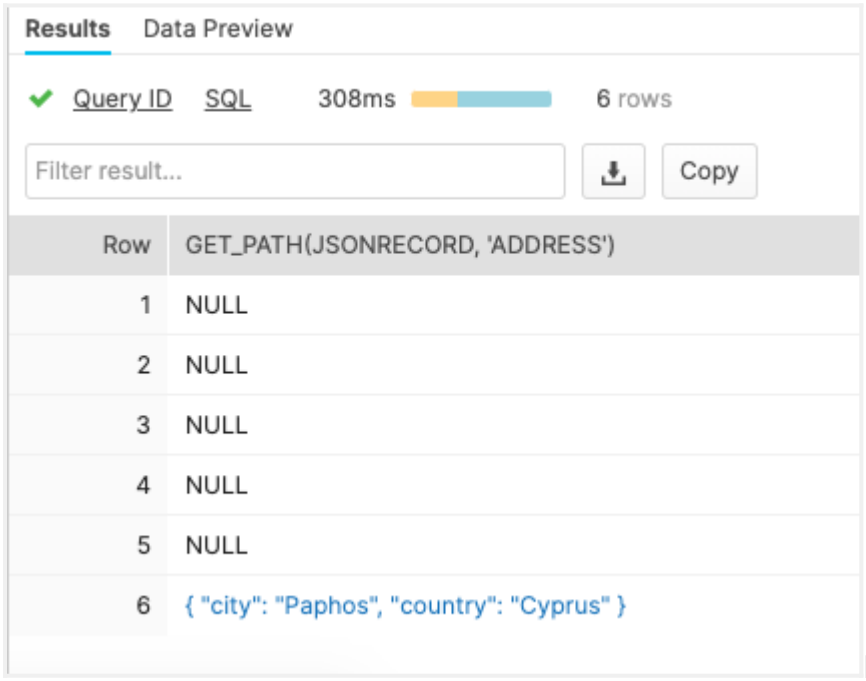
Here we add a **where clause**, using the same colon(:) and dot (.) notation as in the other side of the select statement.

```
select jsonrecord:address.city from JSONRECORD where jsonrecord:customer =
'Maria';
```

We use an alternate approach. We get nested JSON objects by putting the keys in brackets [].

```
select jsonrecord from JSONRECORD where jsonrecord:customer = 'Maria';
```

Values which do not exist are shown as NULL.



Here we pick the first element from an array since the array index (It starts at 0.).

```
select jsonrecord from JSONRECORD where jsonrecord:customer = 'Maria';
```

Here we use the colon (:) to get the same column.

```
select jsonrecord:orders from JSONRECORD where jsonrecord:customer = 'Maria';
```

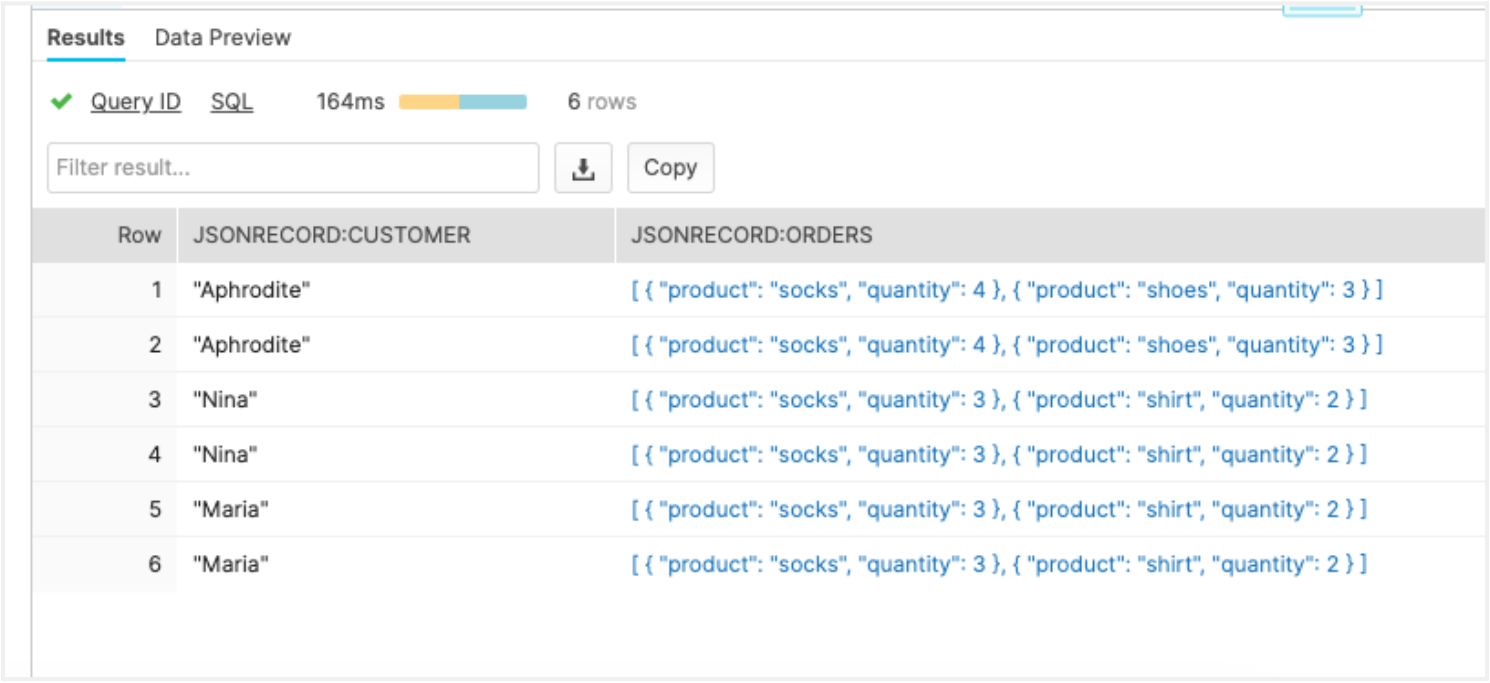
Results:

```
{ "product": "socks", "quantity": 3 }
```

Here, we flatten the array. This record has two order JSON records. So, it shows two rows in the results, with each record attached to the other attributes.

In other words, it explodes it out to **array_size** rows, filling out the other columns with the non-array columns in the select statement. Think of it as an easy was to show all the orders a customer made where the order data and the customer data are repeated to make it easy to see:

```
select jsonrecord:customer, jsonrecord:orders  from JSONRECORD ,
   lateral flatten(input => jsonrecord:orders) prod ;
```



Row	JSONRECORD:CUSTOMER	JSONRECORD:ORDERS
1	"Aphrodite"	[{ "product": "socks", "quantity": 4 }, { "product": "shoes", "quantity": 3 }]
2	"Aphrodite"	[{ "product": "socks", "quantity": 4 }, { "product": "shoes", "quantity": 3 }]
3	"Nina"	[{ "product": "socks", "quantity": 3 }, { "product": "shirt", "quantity": 2 }]
4	"Nina"	[{ "product": "socks", "quantity": 3 }, { "product": "shirt", "quantity": 2 }]
5	"Maria"	[{ "product": "socks", "quantity": 3 }, { "product": "shirt", "quantity": 2 }]
6	"Maria"	[{ "product": "socks", "quantity": 3 }, { "product": "shirt", "quantity": 2 }]

BMC, Control-M support Snowflake

BMC is a member of the Snowflake Technology Alliance Partner program. Snowflake's cloud data platform helps customers to accelerate the data-driven enterprise with Snowflake's market-leading, built-for-cloud data warehouse and Control-M, our market-leading enterprise application workflow orchestration platform.

(Learn how to integrate Snowflake with Control-M.)

Additional resources

For more tutorials like this, explore these resources:

- BMC Machine Learning & Big Data Blog
- AWS Guide, with 15 articles and tutorials
- Amazon Braket Quantum Computing: How To Get Started