HOW-TO-QUERY-NESTED-XML-DATA-IN-SNOWFLAKE

Snowflake has native support for semi-structured data types such as JSON, XML, AVRO, Parquet, and ORC through the data type VARIANT. With this feature, it is possible to incorporate the semi-structured data formats natively without the need for any complex transformation process with an ETL tool.

There are a few subtle differences that exist in the support for the various semi-structured formats inside a VARIANT column. This article will focus specifically on XML data type and a few of the techniques for querying nested XML data stored inside a Snowflake VARIANT column.

Support for XML in Snowflake is currently in preview, but the feature is sufficiently stable for loading data in this file format into tables in your account and querying the data once it is loaded. We are unlikely to be presented with a full complex representation of data or the desire to keep it in XML format for querying. The way that XML needs to be queried may be the biggest deterrent, which will be a benefit as querying XML is not very performant. This article seeks to demystify querying XML!

1. Examine the XML file

Let us copy the below XML contents into a file named **dept.xml** and place it in a folder, let's say in, **/tmp/data**.

```
<dept>
    <dept_id>1</dept_id>
    <dept_name>Marketing</dept_name>
    <employee>
        <emp_id>1</emp_id>
        <emp_fname>Chandler</emp_fname>
        <emp_lname>Bing</emp_lname>
        <emp_title>Director</emp_title>
        <emp_ssn>123-45-6789</emp_ssn>
        <address>
            <street_1>12345 Central Park Ave.</street_1>
            <street_2></street_2>
            <city>Hazel Crest</city>
            <state>IL</state>
            <zipcode>60429</zipcode>
            <start_date>1968-06-01</start_date>
            <end_date>1976-05-01</end_date>
        </address>
        <address>
            <street_1>1011 Spencer Ave.</street_1>
            <street_2></street_2>
            <city>Indianapolis</city>
            <state>IN</state>
            <zipcode>46218</zipcode>
            <start_date>1976-05-02</start_date>
            <end_date>1980-08-31</end_date>
        </address>
        <address>
            <street_1>4300 36th Street S.</street_1>
            <street_2></street_2>
            <city>Arlington</city>
            <state>VA</state>
            <zipcode>22206</zipcode>
            <start_date>1980-09-01</start_date>
            <end_date></end_date>
        </address>
    </employee>
    <employee>
        <emp_id>2</emp_id>
        <emp_fname>Joey</emp_fname>
        <emp_lname>Tribbiani</emp_lname>
        <emp_title>Manager</emp_title>
        <emp_ssn>234-56-7890</emp_ssn>
        <address>
            <street_1>2312 28th Rd. S.</street_1>
            <street_2></street_2>
            <city>Arlington</city>
            <state>VA</state>
            <zipcode>22206</zipcode>
            <start_date>2000-03-01</start_date>
            <end_date>2012-12-31</end_date>
        </address>
        <address>
            <street_1>4102 29th Street S.</street_1>
            <street_2></street_2>
            <city>Arlington</city>
            <state>VA</state>
            <zipcode>22206</zipcode>
            <start_date>2013-01-01</start_date>
            <end_date>2017-08-31</end_date>
        </address>
        <address>
            <street_1>1501 5th Avenue</street_1>
            <street_2>#305</street_2>
            <city>New York</city>
            <state>NY</state>
            <zipcode>10012</zipcode>
            <start_date>2017-09-01</start_date>
            <end_date></end_date>
        </address>
    </employee>
    <employee>
        <emp_id>3</emp_id>
        <emp_fname>Ross</emp_fname>
        <emp_lname>Geller</emp_lname>
        <emp_title>Associate</emp_title>
        <emp_ssn>345-67-8901</emp_ssn>
        <address>
            <street_1>123 Burbank Dr</street_1>
            <street_2></street_2>
            <city>Orchard Park</city>
            <state>NY</state>
            <zipcode>14127</zipcode>
            <start_date>2013-05-07</start_date>
            <end_date>2017-08-31</end_date>
        </address>
        <address>
            <street_1>27 Brantford Place</street_1>
            <street_2></street_2>
            <city>Buffalo</city>
            <state>NY</state>
            <zipcode>14222</zipcode>
            <start_date>2017-09-01</start_date>
            <end_date></end_date>
        </address>
    </employee>
</dept>
<dept>
    <dept_id>2</dept_id>
    <dept_name>Sales</dept_name>
    <employee>
        <emp_id>4</emp_id>
        <emp_fname>Monica</emp_fname>
        <emp_lname>Geller</emp_lname>
        <emp_title>Global VP</emp_title>
        <emp_ssn>456-78-9012</emp_ssn>
        <address>
```

```
            <street_1>123 Oak St.</street_1>
            <street_2></street_2>
            <city>Brooklyn</city>
            <state>NY</state>
            <zipcode>11223</zipcode>
            <start_date>1997-01-01</start_date>
            <end_date>2011-08-30</end_date>
        </address>
        <address>
            <street_1>54321 Willow Ln.</street_1>
            <street_2></street_2>
            <city>Queens</city>
            <state>NY</state>
            <zipcode>11265</zipcode>
            <start_date>2011-09-01</start_date>
            <end_date>2015-07-18</end_date>
        </address>
        <address>
            <street_1>52 Longmeadow Dr.</street_1>
            <street_2></street_2>
            <city>Southampton</city>
            <state>NY</state>
            <zipcode>11713</zipcode>
            <start_date>2015-07-19</start_date>
            <end_date></end_date>
        </address>
    </employee>
    <employee>
        <emp_id>5</emp_id>
        <emp_fname>Phoebe</emp_fname>
        <emp_lname>Buffay</emp_lname>
        <emp_title>Manager - US East</emp_title>
        <emp_ssn>567-89-0123</emp_ssn>
        <address>
            <street_1>123 Howard Street</street_1>
            <street_2>Apt #712</street_2>
            <city>New York</city>
            <state>NY</state>
            <zipcode>10013</zipcode>
            <start_date>1997-01-01</start_date>
            <end_date>2011-08-30</end_date>
        </address>
        <address>
            <street_1>30 Rockefeller Plaza</street_1>
            <street_2>9th Floor</street_2>
            <city>New York</city>
            <state>NY</state>
            <zipcode>10036</zipcode>
            <start_date>2011-09-01</start_date>
            <end_date>2015-07-18</end_date>
        </address>
        <address>
            <street_1>1616 Flatbush Ave.</street_1>
            <street_2></street_2>
            <city>Queens</city>
            <state>NY</state>
            <zipcode>11265</zipcode>
            <start_date>2015-07-19</start_date>
            <end_date></end_date>
        </address>
    </employee>
    <employee>
        <emp_id>6</emp_id>
        <emp_fname>Rachel</emp_fname>
        <emp_lname>Green</emp_lname>
        <emp_title>Sales Director</emp_title>
        <emp_ssn>678-90-1234</emp_ssn>
        <address>
            <street_1>7200 Central Park West</street_1>
            <street_2></street_2>
            <city>New York</city>
            <state>NY</state>
            <zipcode>10023</zipcode>
            <start_date>1997-01-01</start_date>
            <end_date>2011-08-30</end_date>
        </address>
        <address>
            <street_1>450 Concar Ave</street_1>
            <street_2></street_2>
            <city>San Mateo</city>
            <state>CA</state>
            <zipcode>94402</zipcode>
            <start_date>2011-09-01</start_date>
            <end_date>2015-07-18</end_date>
        </address>
        <address>
            <street_1>12345 Meadowbrook Dr.</street_1>
            <street_2></street_2>
            <city>Westfield</city>
            <state>IN</state>
            <zipcode>46038</zipcode>
            <start_date>2015-07-19</start_date>
            <end_date>2019-06-30</end_date>
        </address>
        <address>
            <street_1>15 Campus Dr.</street_1>
            <street_2>Apt 102</street_2>
            <city>Bloomington</city>
            <state>IN</state>
            <zipcode>44444</zipcode>
            <start_date>2019-07-01</start_date>
            <end_date></end_date>
        </address>
    </employee>
</dept>
<dept>
    <dept_id>3</dept_id>
    <dept_name>Engineering</dept_name>
    <employee>
        <emp_id>7</emp_id>
        <emp_fname>Quentin</emp_fname>
        <emp_lname>Tarantino</emp_lname>
        <emp_title>Director</emp_title>
        <emp_ssn>789-10-2345</emp_ssn>
        <address>
            <street_1>77 Sunset Strip</street_1>
            <street_2></street_2>
            <city>Hollywood</city>
            <state>CA</state>
            <zipcode>90038</zipcode>
            <start_date>1997-01-01</start_date>
            <end_date>2011-08-30</end_date>
        </address>
        <address>
            <street_1>10202 Washington Blvd</street_1>
            <street_2></street_2>
            <city>Culver City</city>
            <state>CA</state>
            <zipcode>90232</zipcode>
            <start_date>2011-09-01</start_date>
            <end_date>2015-07-18</end_date>
        </address>
        <address>
            <street_1>515 Melrose Avenue</street_1>
            <street_2></street_2>
            <city>Los Angeles</city>
            <state>CA</state>
            <zipcode>90038</zipcode>
            <start_date>2015-07-19</start_date>
            <end_date></end_date>
        </address>
    </employee>
    <employee>
        <emp_id>8</emp_id>
        <emp_fname>Bradley</emp_fname>
        <emp_lname>Pitt</emp_lname>
        <emp_title>Manager</emp_title>
        <emp_ssn>890-12-3456</emp_ssn>
        <address>
            <street_1>15 Wilshire Blvd</street_1>
            <street_2></street_2>
            <city>Los Angeles</city>
            <state>CA</state>
            <zipcode>90038</zipcode>
            <start_date>1997-01-01</start_date>
            <end_date>2011-08-30</end_date>
        </address>
        <address>
            <street_1>8591 Mulholland Drive</street_1>
            <street_2></street_2>
            <city>Los Angeles</city>
            <state>CA</state>
            <zipcode>90046</zipcode>
            <start_date>2011-09-01</start_date>
            <end_date>2015-07-18</end_date>
        </address>
        <address>
            <street_1>2859 Coldwater Canyon</street_1>
            <street_2></street_2>
            <city>Beverly Hills</city>
            <state>CA</state>
            <zipcode>90210</zipcode>
            <start_date>2015-07-19</start_date>
            <end_date></end_date>
        </address>
    </employee>
    <employee>
        <emp_id>9</emp_id>
        <emp_fname>Leonardo</emp_fname>
        <emp_lname>DiCaprio</emp_lname>
        <emp_title>Engineer</emp_title>
        <emp_ssn>901-23-4567</emp_ssn>
        <address>
            <street_1>9255 Sunset Blvd</street_1>
            <street_2>Suite 625</street_2>
            <city>West Hollywood</city>
            <state>CA</state>
            <zipcode>90069</zipcode>
            <start_date>1997-01-01</start_date>
            <end_date>2011-08-30</end_date>
        </address>
        <address>
            <street_1>1515 La Cienega Blvd</street_1>
            <street_2>#21</street_2>
            <city>Los Angeles</city>
            <state>CA</state>
            <zipcode>90038</zipcode>
            <start_date>2011-09-01</start_date>
            <end_date>2015-07-18</end_date>
        </address>
        <address>
            <street_1>23800 Malibu Crest Dr.</street_1>
            <street_2></street_2>
            <city>Malibu</city>
            <state>CA</state>
            <zipcode>90265</zipcode>
            <start_date>2015-07-19</start_date>
            <end_date></end_date>
        </address>
    </employee>
</dept>
<dept>
    <dept_id>4</dept_id>
    <dept_name>Human Resources</dept_name>
    <employee>
        <emp_id>10</emp_id>
        <emp_fname>Margot</emp_fname>
        <emp_lname>Robbie</emp_lname>
        <emp_title>EVP</emp_title>
        <emp_ssn>012-34-5678</emp_ssn>
        <address>
            <street_1>2000 Main Street</street_1>
            <street_2></street_2>
            <city>Napa</city>
            <state>CA</state>
            <zipcode>94574</zipcode>
            <start_date>1997-01-01</start_date>
            <end_date>2011-08-30</end_date>
        </address>
        <address>
            <street_1>8300 St. Helena Hwy</street_1>
            <street_2></street_2>
            <city>Napa</city>
            <state>CA</state>
            <zipcode>94573</zipcode>
            <start_date>2011-09-01</start_date>
            <end_date>2015-07-18</end_date>
        </address>
        <address>
            <street_1>1429 Tubbs Lane</street_1>
            <street_2></street_2>
            <city>Calistoga</city>
            <state>CA</state>
            <zipcode>94515</zipcode>
            <start_date>2015-07-19</start_date>
            <end_date></end_date>
        </address>
    </employee>
    <employee>
        <emp_id>11</emp_id>
        <emp_fname>Robert</emp_fname>
        <emp_lname>Redford</emp_lname>
        <emp_title>Recruiting</emp_title>
        <emp_ssn>987-65-4321</emp_ssn>
        <address>
            <street_1>1 E. 161st Street</street_1>
            <street_2></street_2>
            <city>The Bronx</city>
            <state>NY</state>
            <zipcode>10451</zipcode>
            <start_date>1997-01-01</start_date>
            <end_date>2011-08-30</end_date>
        </address>
        <address>
            <street_1>1060 W Addison St</street_1>
            <street_2></street_2>
            <city>Chicago</city>
            <state>IL</state>
            <zipcode>60613</zipcode>
            <start_date>2011-09-01</start_date>
            <end_date>2015-07-18</end_date>
        </address>
        <address>
            <street_1>24 Willie Mays Plaza</street_1>
            <street_2></street_2>
            <city>San Francisco</city>
            <state>CA</state>
            <zipcode>94107</zipcode>
            <start_date>2015-07-19</start_date>
            <end_date></end_date>
        </address>
    </employee>
</dept>
```

The data contains a three-level nested structure: DEPT -> EMPLOYEE -> ADDRESS.

Each Department has multiple Employees, and within every Employee, there is a record of their Addresses over time.

2. Load the XML file into Snowflake

All this requires is a single table with a single variant column.

```
// Create a Snowflake table with a single variant column
  create or replace table DEPT_EMP_ADDR (
     xmldata variant not null
);
```

Since this is a small file, you can use the **Load Table** dialog in the WebUI (from the Databases tab, select the Tables tab, select the DEPT_EMP_ADDR table and then select the Load Data....option).

We can also use the **PUT** and the **COPY** commands through the SnowSQL utility.

Here is a code block that uses the User internal stage (assuming you've placed the **dept.xml** file in your **~/tmp/data** folder...):

```
PUT file://~/tmp/data/dept.xml @~/xml ;

COPY INTO DEPT_EMP_ADDR
FROM @~/xml
FILE_FORMAT=(TYPE=XML) ON_ERROR='CONTINUE';
```

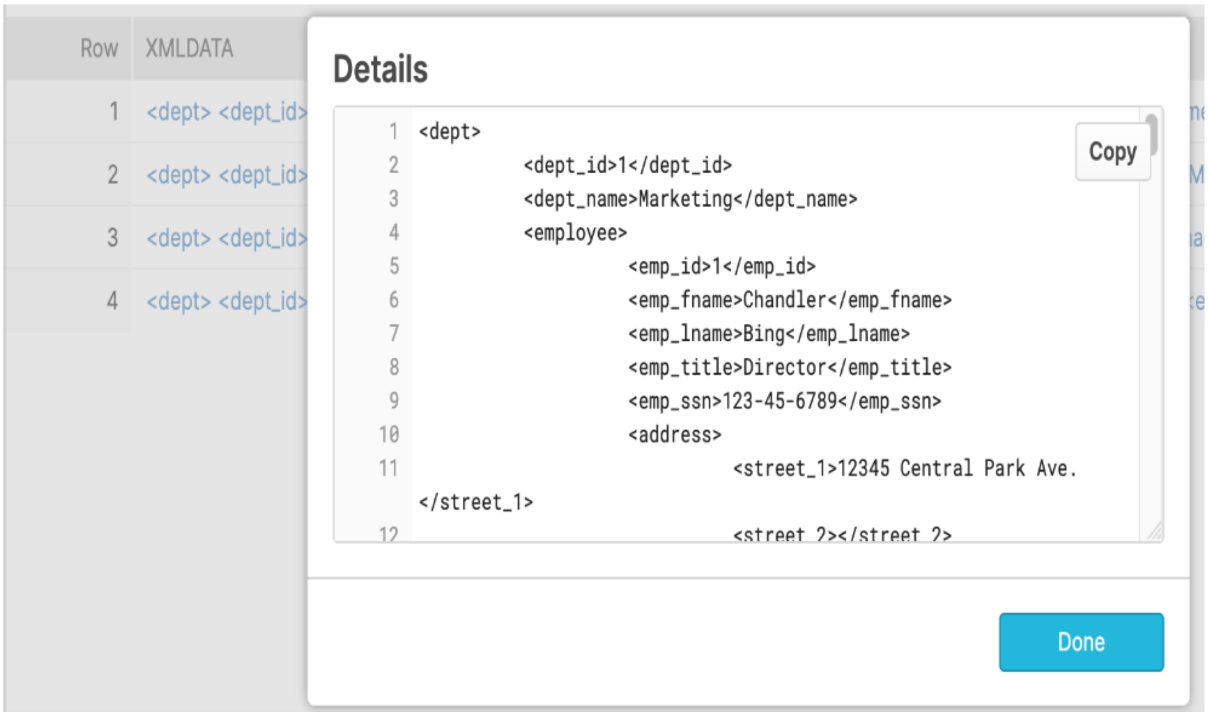
This should load 4 rows into the **DEPT_EMP_ADDR** table, as there are four outermost **<dept>** elements in the XML document.

3. Run some queries against the VARIANT column

Let us now start with a simple **SELECT** * to see what Snowflake is storing in the table.

```
SELECT * FROM DEPT_EMP_ADDR;
```

As you can see, each row contains one entire root node from the XML document. Since there are 4 root nodes in the document, there are 4 rows in the Snowflake table. If you click on any of the XML values, you'll pop open a dialog window that shows the entire node.



Row	XMLDATA
1	<dept> <dept_id>
2	<dept> <dept_id>
3	<dept> <dept_id>
4	<dept> <dept_id>

```
<dept>
        <dept_id>1</dept_id>
        <dept_name>Marketing</dept_name>
        <employee>
                <emp_id>1</emp_id>
                <emp_fname>Chandler</emp_fname>
                <emp_lname>Bing</emp_lname>
                <emp_title>Director</emp_title>
                <emp_ssn>123-45-6789</emp_ssn>
                <address>
                        <street_1>12345 Central Park Ave.
</street_1>
                        <street_2></street_2>
```

Even though this data is in a VARIANT column, the usual "dot notation" syntax that can be used to access JSON data does not work against XML. Fortunately, there's a set of functions that can be used to explicitly retrieve data from the XML document.

XML Operators and Functions

$ (dollar sign)

The dollar sign operator returns the contents, as a VARIANT, of the value it operates on. For an element, the contents of that element are returned:

- If the contents of that element is text, text is returned as a VARIANT.
- If the contents are an element, the element is returned as a VARIANT in XML format.
- If the contents are a series of elements, an array of the elements are returned as a VARIANT in JSON format.

HOW-TO-QUERY-NESTED-XML-DATA-IN-SNOWFLAKE

The latter form is important as it exposes how the VARIANT type is accessed:

- "$" for the value.
- "@" for the name.
- "@attr" for the value of the named attribute (e.g. 'attr' in this example).

@ (ampersand)

The ampersand operator returns the name of the current element. This is useful when you are iterating through differently-named elements.

@attr (ampersand followed by the attribute name)

This operator returns the value of the attribute with the name that directly follows the ampersand; here we are displaying the value of an attribute named "attr". If no attribute is found, NULL is returned.

XMLGET(XML, NAME [, INDEX])

The **XMLGET** function returns the element of NAME at index INDEX, if provided directly under the XML.

If the optional INDEX is not provided, the default value is 0, and the first element of name **NAME** is returned as **XML** with all children nodes. If no value is found, **NULL** is returned.

If an element of name **NAME** is found nested in one of the direct children of **XML**, **NULL** is returned since the **XMLGET** function is not recursive.

Examples Using Sample Data

Simple XML Example (Single Element)

Consider the following simple XML object stored in a table in a VARIANT column named `src`:

```
<catalog issue="spring">
    <book id="bk101">The Good Book</book>
</catalog>
```

Querying on the `src` column using the various operators directly and using the XMLGET function in conjunction with the operators produces the following results:

HOW-TO-QUERY-NESTED-XML-DATA-IN-SNOWFLAKE

Selecting	Returns	Example Output
`src`	The XML contained in the VARIANT field, i.e. the full XML document.	`<catalog="spring"> <book id="bk101">The Good Book</book> </catalog>`
`src:"$"`	The contents of the VARIANT field.	`<book id="bk101">The Good Book</book>`
`src:"@"`	The name of the "root" element of the XML contained in the VARIANT field.	`"catalog"`
`src:"@issue"`	The "issue" attribute of the root "catalog" element.	`"spring"`
`src:"$"."@id"`	The "id" attribute of the first and only node that is contained within the root element of the XML.	`"bk101"`
`XMLGET(src, 'book')`	The first element contained within the root element of the XML.	`<book id="bk101">The Good Book</book>`
`XMLGET(src, 'book'):"$"`	The contents of the first "book" element.	`"The Good Book"`
`XMLGET(src, 'book'):"@"`	The name of the first "book" element in the root element.	`"book"`
`XMLGET(src, 'book'):"@id"`	The "id" attribute of the first element in the root element of the XML.	`"bk101"`

Simple XML Example (Multiple Elements)

Expanding on the previous example, consider the following simple XML object containing multiple child elements:

```
<catalog issue="spring">
    <book id="bk101">The Good Book</book>
    <book id="bk102">The OK Book</book>
</catalog>
```

Performing the same set of queries produces the following results:

HOW-TO-QUERY-NESTED-XML-DATA-IN-SNOWFLAKE

Selecting	Returns	Example Output

`src`	The XML contained in the VARIANT field, i.e. the full XML document.	`<catalog="spring">`
		`  <book id="bk101">The Good Book</book>`
		`  <book id="bk102">The OK Book</book>`
		`</catalog>`

`src:"$"`	The meta-description of the XML array in the root element of the XML.	`[`
		`  {`
		`   "$": "The Good Book",`
		`   "@": "book",`
		`   "@id": "bk101"`
		`  },`
		`  {`
		`   "$": "The OK Book",`
		`   "@": "book",`
		`   "@id": "bk102"`
		`  }`
		`]`

`src:"@"`	The name of the "root" element of the XML contained in the VARIANT field.	`"catalog"`
`src:"@issue"`	The "issue" attribute of the root "catalog" element.	`"spring"`
`src:"$"."@id"`	Unlike elements with a single child, 2 or more child elements become an array, so this notation becomes invalid as we need to refer to the index of the child element before querying for the attribute.	`NULL`

`src:"$"[0]."@id"`
 	
`src:"$"[1]."@id"`
	The "id" attribute of the element at the first (0) and second (1) index in the array of child elements in the root element.	`"bk101"`
		
		`"bk102"`

`XMLGET(src, 'book')`
 	
`XMLGET(src, 'book', 1)`
	The first and second elements contained within the root element of the XML. INDEX is optional and defaults to 0.	`<book id="bk101">The Good Book</book>`
		
		`<book id="bk102">The OK Book</book>`

`XMLGET(src, 'book'):"$"`
 	
`XMLGET(src, 'book', 1):"$"`
	The first (index 0) and second (index 1) child element's contents.	`"The Good Book"`
		
		`"The OK Book"`

`XMLGET(src, 'book'):"@"`
 	
`XMLGET(src, 'book', 1):"@"`
	The first (index 0) and second (index 1) child element's name.	`"book"`
		
		`"book"`

`XMLGET(src, 'book'):"@id"`
 	
`XMLGET(src, 'book', 1):"@id"`
	The first (index 0) and second (index 1) child element's "id" attribute.	`"bk101"`
		
		`"bk102"`

HOW-TO-QUERY-NESTED-XML-DATA-IN-SNOWFLAKE

4. Using the XMLGET() function

The **XMLGET()** function takes three arguments:

- the name of a variant (either the variant column itself or something that evaluates to a variant);
- the name of the tag you're looking for (the node or attribute name);
- an optional instance number that is used as an "index" into the XML structure.

Let us see this in action:

```
//1. Query to parse out named element values at the root level
(DEPT).
  SELECT
     GET( XMLGET( xmldata, 'dept_id'), '$')::INTEGER as deptID,
     XMLGET( xmldata, 'dept_id' ):"$"::INTEGER AS DEPT_ID,
     XMLGET( xmldata, 'dept_name' ):"$"::STRING AS DEPT_NAME
  FROM DEPT_EMP_ADDR;
```

Notice that there are two different syntax styles used in the first two expressions in this query. They happen to return exactly the same value. The **XMLGET()** function, by itself, just returns an XML element from a variant. To extract an element or attribute value from that, we must use either an explicit call to the **GET()** function or a convenient shorthand notation.

The first expression uses the **GET()** function and passes in the XML element returned from **XMLGET()**, plus a single-quoted **$** symbol.

There are three options for that second parameter:

- The argument '$' returns the element's value.
- The argument '@' returns the element's name.
- Passing '@attrName' returns the value of an attribute of the element. (I'll discuss attributes vs. elements in the next section.)

The expressions used for columns 2 and 3 demonstrate the simplified syntax of the shorthand notation. The same **XMLGET()** function is used but is followed by a single colon character, then one of the same three options shown above. (NOTE: these must now be double-quoted!)

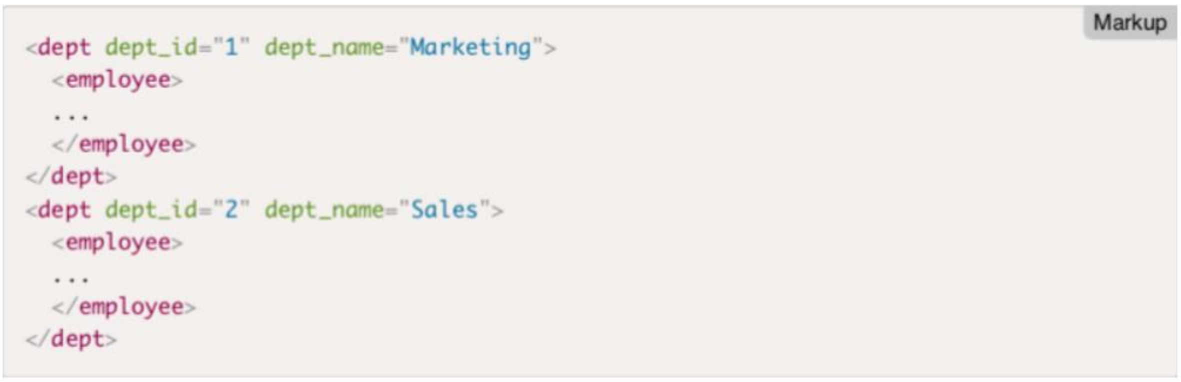
You can then explicitly cast these to any data type you need with the double-colon operator. In the example above, **DEPT_ID** is being cast as an INTEGER, and **DEPT_NAME** is being cast as a STRING.

Understanding Element values vs. Attributes

There are no hard and fast rules concerning the use of element values vs. attributes in XML. The example above assumes the use of element values, but you may often see both styles used in the same document.

Snowflake has different methods for working with element values and attribute values.

Here's a snippet from the sample XML doc, but with the **dept_id** and **dept_name** called out as attributes of **<dept>** instead of as dependent elements.



Markup

```
<dept dept_id="1" dept_name="Marketing">
  <employee>
  ...
  </employee>
</dept>
<dept dept_id="2" dept_name="Sales">
  <employee>
  ...
  </employee>
</dept>
```

If that is the style that you are dealing with in your XML, then the query to retrieve the **dept_id** and **dept_name** values changes slightly.

Notice that we don't need **XMLGET()**, as we're retrieving attributes from the top-level node.

To retrieve attributes from lower-level nodes, we'd have to **XMLGET()** those levels first.

```
SELECT
     GET(xmldata, '@dept_id')::integer as dept_id,
     GET(xmldata, '@dept_name')::string as dept_name
FROM DEPT_EMP_ADDR;
```

HOW-TO-QUERY-NESTED-XML-DATA-IN-SNOWFLAKE

5. Retrieving nested arrays from XML

One of the main features of XML is its ability to “nest” related datasets as sub-nodes in the same document. The downside of that approach comes at data extraction time when trying to maintain the relationships between the entities.

In the sample XML document, we’ve built a three-tier hierarchy, from DEPT to EMPLOYEE to ADDRESS, and each array is nested inside its parent node.

Using FLATTEN() and Lateral Joins

In this next example, we’ll use the **FLATTEN()** function to transform the set of EMPLOYEE nodes under each DEPT node into an array, and then use a LATERAL join to that array.

```
//2. Query to parse out all the elements at the next level down
(EMPLOYEE).
select
       XMLGET( xmldata, 'dept_id' ):"$"::string AS dept_id
     , XMLGET( xmldata, 'dept_name' ):"$"::string AS dept_name
     , XMLGET( emp.value, 'emp_id' ):"$"::integer as emp_id
     , XMLGET( emp.value, 'emp_fname' ):"$"::string as emp_fname
     , XMLGET( emp.value, 'emp_lname' ):"$"::string as emp_lname
     , XMLGET( emp.value, 'emp_title' ):"$"::string as emp_title
     , XMLGET( emp.value, 'emp_ssn' ):"$"::string as emp_ssn
from
    dept_emp_addr
    ,  lateral FLATTEN(dept_emp_addr.xmldata:"$") emp
  where GET( emp.value, '@') = 'employee'
  order by dept_id, emp_id;
```

Some items of note here:

- The **FLATTEN()** function creates an internal array structure that contains 6 elements: **SEQ**, **KEY**, **PATH**, **INDEX**, **THIS**, and **VALUE**. You can see this structure by adding the expression, emp.* into the SELECT list above. While all 6 fields are useful, we’re only interested in the **value** field here.
- The array structure created by the **FLATTEN()** function is aliased with the name **emp** so that we can refer to it in the SELECT list expressions.
- The emp structure array contains a row for every subnode of **<dept>**. This includes not just the **<employee>** nodes, but also the **<dept_id>** and **<dept_name>** nodes as well. If we don’t filter these out, there will be too many rows in the resultset.

We need to add a WHERE clause that only includes the **<employee>** nodes. To see this, comment out the WHERE clause and re-execute the query.

- We are using the shorthand:"$" syntax to access the element's value, not the more complex **GET()** method described above.

Going one level deeper:

Now let's write a query that dives into the **<address>** nodes, which are subnodes of **<employee>**. This will require another call to **FLATTEN()** and another lateral join in the FROM clause.

```
//3. Query to parse out all the elements at the next level down
(ADDRESS).
select
     XMLGET( xmldata, 'dept_id' ):"$"::INTEGER AS ID
   , XMLGET( xmldata, 'dept_name' ):"$"::STRING AS NAME
   , XMLGET( emp.value, 'emp_id' ):"$"::INTEGER as emp_id
   , XMLGET( emp.value, 'emp_fname' ):"$"::STRING || ' ' ||
     XMLGET( emp.value, 'emp_lname' ):"$"::STRING as emp_name
   , XMLGET( addr.value, 'street_1' ):"$"::STRING as street_1
   , XMLGET( addr.value, 'city' ):"$"::STRING as city
   , XMLGET( addr.value, 'state' ):"$"::STRING as state
   , XMLGET( addr.value, 'zipcode' ):"$"::STRING as zipcode
   , XMLGET( addr.value, 'start_date' ):"$"::DATE as start_date
   , XMLGET( addr.value, 'end_date' ):"$"::DATE as end_date
from
  dept_emp_addr
  ,  lateral FLATTEN( xmldata:"$" ) emp
  ,  lateral FLATTEN( emp.value:"$" ) addr
where emp.value like '<employee>%'
  and addr.value like '<address>%'
order by ID, emp_id, start_date desc;
```

Notes on this version of the query:

- The second **FLATTEN()** call passes **emp.value** and is aliased as **addr**.
- The WHERE clause includes only **<employee>** and **<address>** nodes.
- The **emp_fname** and **emp_lname** fields have been concatenated into a single value.

HOW-TO-QUERY-NESTED-XML-DATA-IN-SNOWFLAKE

More Complex XML Example

This example uses the following data (expanded from the previous example) and stored in an XML file:

```
<catalog issue="spring" date="2015-04-15">
    <book id="bk101">
        <title>Some Great Book</title>
        <genre>Great Books</genre>
        <author>Jon Smith</author>
        <publish_date>2001-12-28</publish_date>
        <price>23.39</price>
        <description>This is a great book!</description>
    </book>
    <cd id="cd101">
        <title>Sad Music</title>
        <genre>Sad</genre>
        <artist>Emo Jones</artist>
        <publish_date>2010-11-23</publish_date>
        <price>15.25</price>
        <description>This music is so sad!</description>
    </cd>
    <map id="map101">
        <title>Good CD</title>
        <location>North America</location>
        <author>Joey Bagadonuts</author>
        <publish_date>2013-02-02</publish_date>
        <price>102.95</price>
        <description>Trail map of North America</description>
    </map>
</catalog>
```

To use this XML data file:

1. Stage the file in the internal staging location for your Snowflake user:

```
PUT file:///examples/xml/* @~/xml;
```

1. Create the table structure for holding the XML data in Snowflake:

```
CREATE OR REPLACE TABLE demo_db.public.sample_xml(src VARIANT);
```

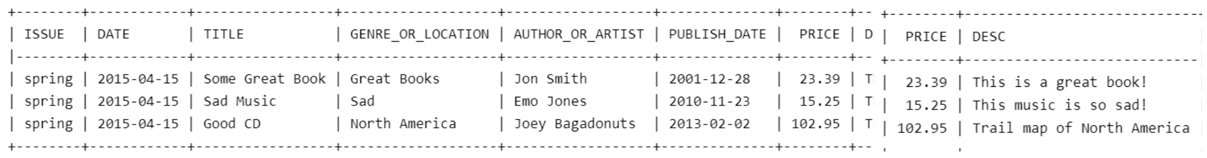
HOW-TO-QUERY-NESTED-XML-DATA-IN-SNOWFLAKE

1. Load the XML raw data from the file into the table you just created:

```
COPY INTO demo_db.public.sample_xml
FROM @~/xml
FILE_FORMAT=(TYPE=XML) ON_ERROR='CONTINUE';
```

To display all of the data loaded into the VARIANT column from the XML file:

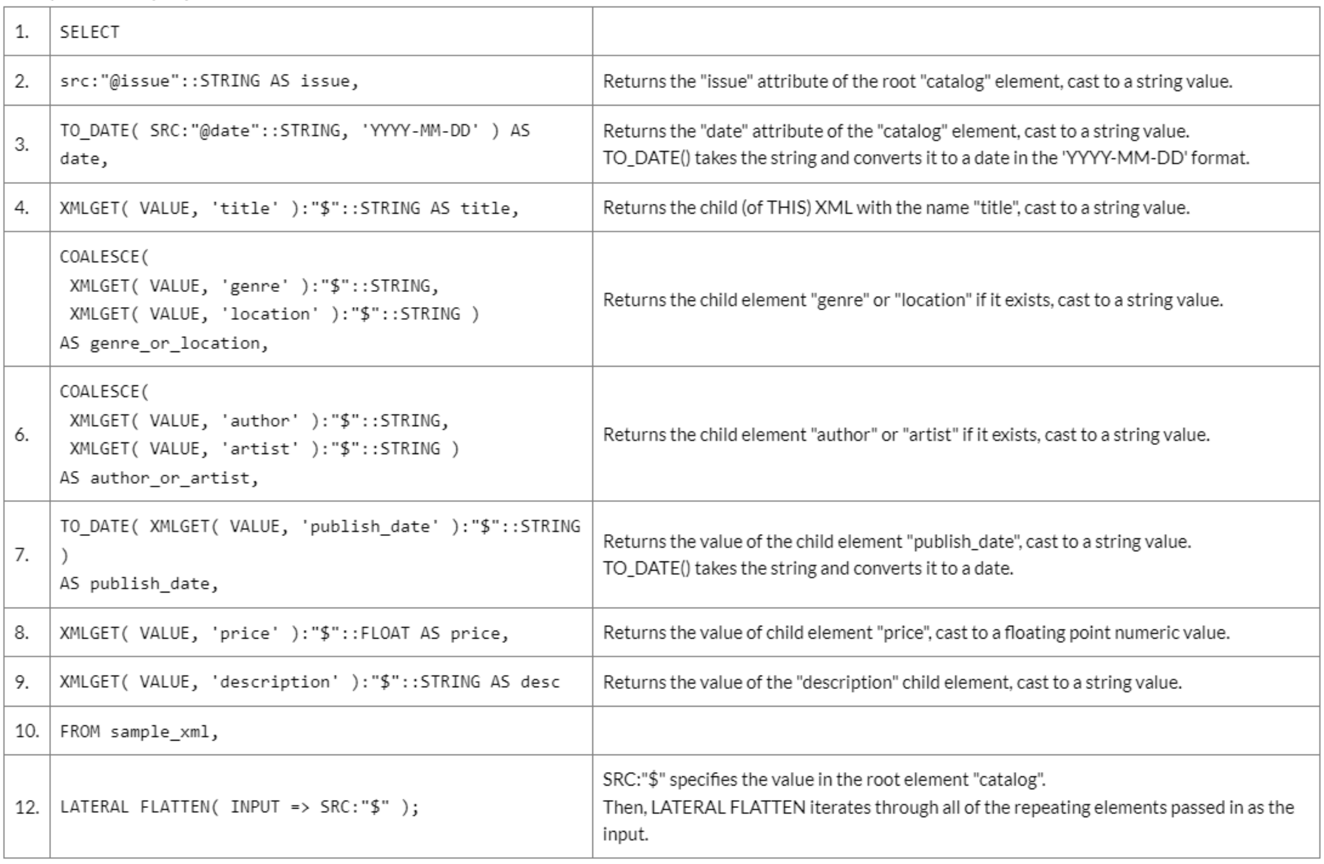
```
SELECT
src:"@issue"::STRING AS issue,
TO_DATE( src:"@date"::STRING, 'YYYY-MM-DD' ) AS date,
XMLGET( VALUE, 'title' ):"$"::STRING AS title,
COALESCE( XMLGET( VALUE, 'genre' ):"$"::STRING,
          XMLGET( VALUE, 'location' ):"$"::STRING ) AS
genre_or_location,
COALESCE( XMLGET( VALUE, 'author' ):"$"::STRING,
          XMLGET( VALUE, 'artist' ):"$"::STRING ) AS
author_or_artist,
TO_DATE( XMLGET( VALUE, 'publish_date' ):"$"::String ) AS
publish_date,
XMLGET( VALUE, 'price' ):"$"::FLOAT AS price,
XMLGET( VALUE, 'description' ):"$"::STRING AS desc
FROM sample_xml,
LATERAL FLATTEN( INPUT => SRC:"$" );
```



```
+--------+------------+-----------------+-------------------+-----------------+--------------+--------+-- +--------+---------------------------
| ISSUE  | DATE       | TITLE           | GENRE_OR_LOCATION | AUTHOR_OR_ARTIST | PUBLISH_DATE |  PRICE | D |  PRICE | DESC
|--------+------------+-----------------+-------------------+-----------------+--------------+--------+-- +--------+---------------------------
| spring | 2015-04-15 | Some Great Book | Great Books       | Jon Smith        | 2001-12-28   |  23.39 | T |  23.39 | This is a great book!
| spring | 2015-04-15 | Sad Music       | Sad               | Emo Jones        | 2010-11-23   |  15.25 | T |  15.25 | This music is so sad!
| spring | 2015-04-15 | Good CD         | North America     | Joey Bagadonuts  | 2013-02-02   | 102.95 | T | 102.95 | Trail map of North America
+--------+------------+-----------------+-------------------+-----------------+--------------+--------+--
```

To help break this query down, we'll describe each line:

HOW-TO-QUERY-NESTED-XML-DATA-IN-SNOWFLAKE



1.	`SELECT`	
2.	`src:"@issue"::STRING AS issue,`	Returns the "issue" attribute of the root "catalog" element, cast to a string value.
3.	`TO_DATE( SRC:"@date"::STRING, 'YYYY-MM-DD' ) AS date,`	Returns the "date" attribute of the "catalog" element, cast to a string value. TO_DATE() takes the string and converts it to a date in the 'YYYY-MM-DD' format.
4.	`XMLGET( VALUE, 'title' ):"$"::STRING AS title,`	Returns the child (of THIS) XML with the name "title", cast to a string value.
	`COALESCE( XMLGET( VALUE, 'genre' ):"$"::STRING, XMLGET( VALUE, 'location' ):"$"::STRING ) AS genre_or_location,`	Returns the child element "genre" or "location" if it exists, cast to a string value.
6.	`COALESCE( XMLGET( VALUE, 'author' ):"$"::STRING, XMLGET( VALUE, 'artist' ):"$"::STRING ) AS author_or_artist,`	Returns the child element "author" or "artist" if it exists, cast to a string value.
7.	`TO_DATE( XMLGET( VALUE, 'publish_date' ):"$"::STRING ) AS publish_date,`	Returns the value of the child element "publish_date", cast to a string value. TO_DATE() takes the string and converts it to a date.
8.	`XMLGET( VALUE, 'price' ):"$"::FLOAT AS price,`	Returns the value of child element "price", cast to a floating point numeric value.
9.	`XMLGET( VALUE, 'description' ):"$"::STRING AS desc`	Returns the value of the "description" child element, cast to a string value.
10.	`FROM sample_xml,`	
12.	`LATERAL FLATTEN( INPUT => SRC:"$" );`	SRC:"$" specifies the value in the root element "catalog". Then, LATERAL FLATTEN iterates through all of the repeating elements passed in as the input.

Note that this example introduces two important additional concepts utilized in querying XML:

- **LATERAL FLATTEN (table function)**
- **THIS (output from LATERAL FLATTEN)**

LATERAL FLATTEN

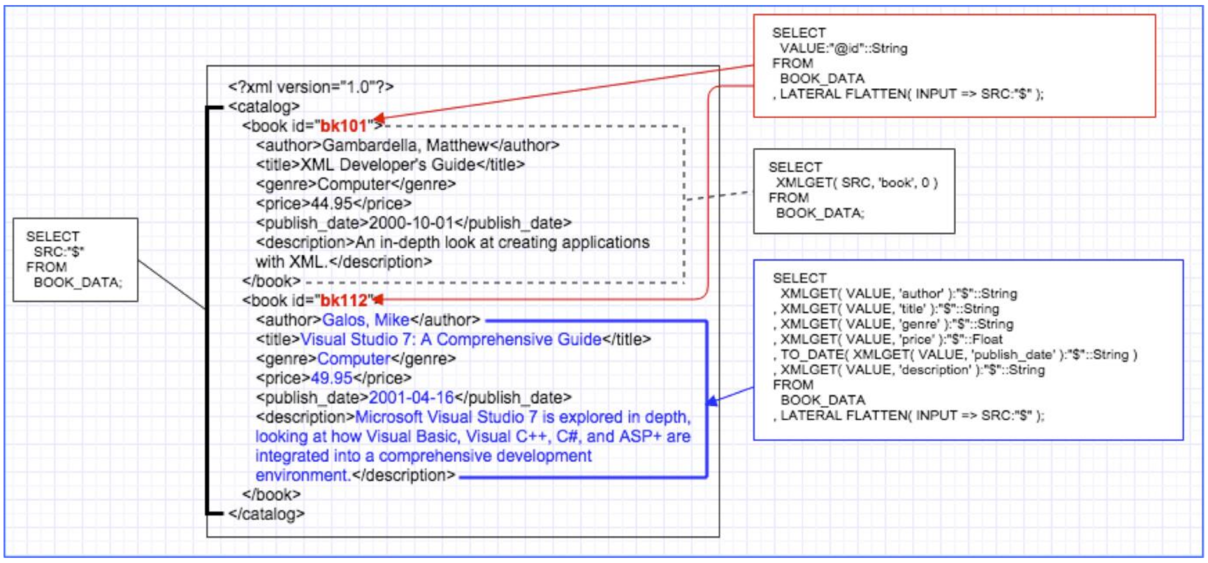
XML commonly has repeating, nested child elements. Child elements can very in their types, meaning you can have a number of different child elements under a single parent element. In the example above, the "catalog" element can also contain "cd"s, "map"s and other items sold or displayed in a catalog. This is enough to illustrate dealing with LATERAL FLATTEN with filtering.

THIS

The concept of THIS in LATERAL FLATTEN is similar to what is described above. The current element, or THIS, is directly queried for attributes using "$", "@", and "@attr". The direct children of THIS are queried using the XMLGET function. Note that querying deeper elements gets tricky and will be discussed in a future article.

Breaking XML Queries Down Graphically

Lastly, now that we've explained the basic concepts, we can show a more realistic use case, iterating through the various child elements. The following diagram illustrates a series of queries and points to the value, values, or XML that each query returns:



```
SELECT
 SRC:"$"
FROM
 BOOK_DATA;
```

```
<?xml version="1.0"?>
<catalog>
  <book id="bk101">
    <author>Gambardella, Matthew</author>
    <title>XML Developer's Guide</title>
    <genre>Computer</genre>
    <price>44.95</price>
    <publish_date>2000-10-01</publish_date>
    <description>An in-depth look at creating applications
    with XML.</description>
  </book>
  <book id="bk112">
    <author>Galos, Mike</author>
    <title>Visual Studio 7: A Comprehensive Guide</title>
    <genre>Computer</genre>
    <price>49.95</price>
    <publish_date>2001-04-16</publish_date>
    <description>Microsoft Visual Studio 7 is explored in depth,
    looking at how Visual Basic, Visual C++, C#, and ASP+ are
    integrated into a comprehensive development
    environment.</description>
  </book>
</catalog>
```

```
SELECT
 VALUE:"@id"::String
FROM
 BOOK_DATA
, LATERAL FLATTEN( INPUT => SRC:"$" );
```

```
SELECT
 XMLGET( SRC, 'book', 0 )
FROM
 BOOK_DATA;
```

```
SELECT
 XMLGET( VALUE, 'author' ):"$"::String
, XMLGET( VALUE, 'title' ):"$"::String
, XMLGET( VALUE, 'genre' ):"$"::String
, XMLGET( VALUE, 'price' ):"$"::Float
, TO_DATE( XMLGET( VALUE, 'publish_date' ):"$"::String )
, XMLGET( VALUE, 'description' ):"$"::String
FROM
 BOOK_DATA
, LATERAL FLATTEN( INPUT => SRC:"$" );
```

6. Conclusion

While XML is not as well supported in Snowflake as are JSON and other semi-structured formats, XML can still be an integral component of an overall data architecture that includes Snowflake. This blog has shown a few of the key techniques for working with nested XML inside Snowflake's VARIANT column type.

7. References

https://community.snowflake.com/s/article/HOW-TO-QUERY-NESTED-XML-DATA-IN-SNOWFLAKE

https://community.snowflake.com/s/article/introduction-to-loading-and-parsing-xml-data-using-sql