**INTRODUCTION TO LOADING AND PARSING XML DATA USING SQL**

Jul 13, 2017•Feature Documentation

**Overview**

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!

**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.

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:

| **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:

| **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" |

**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);

1. Load the XML raw data from the file into the table you just created:
2. COPY INTO demo\_db.public.sample\_xml
3. FROM @~/xml
4. 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 | DESC |

|--------+------------+-----------------+-------------------+------------------+--------------+--------+----------------------------|

| spring | 2015-04-15 | Some Great Book | Great Books | Jon Smith | 2001-12-28 | 23.39 | This is a great book! |

| spring | 2015-04-15 | Sad Music | Sad | Emo Jones | 2010-11-23 | 15.25 | This music is so sad! |

| spring | 2015-04-15 | Good CD | North America | Joey Bagadonuts | 2013-02-02 | 102.95 | Trail map of North America |

+--------+------------+-----------------+-------------------+------------------+--------------+--------+----------------------------+

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

|  |  |  |
| --- | --- | --- |
| 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:

