INTRODUCTION TO LOADING AND PARSING XML DATA USING SQL

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:

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:

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;

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

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

3. 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 | 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:

Note that this example introduces two important additional concepts utilized in querying XML:
- LATERAL FLATTEN (table function)

- THIS (output from LATERAL FLATTEN)

LATERAL FLATTEN

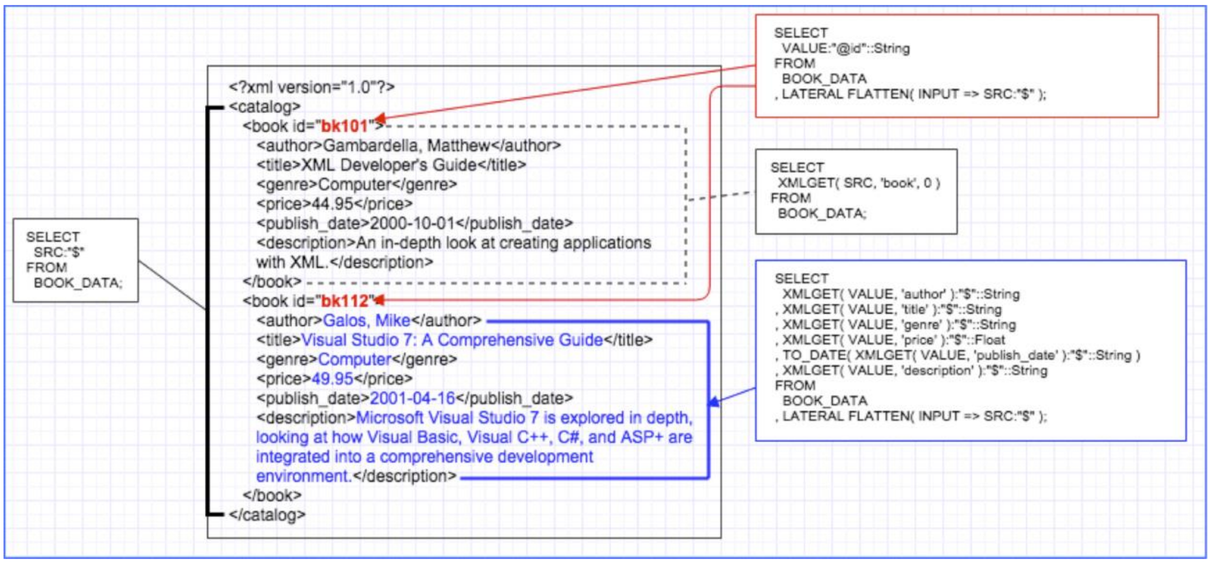
XML commonly has repeating, nested child elements. Child elements can very in their types, meaning you can have a number of different child elements under a single parent element. In the example above, the "catalog" element can also contain "cd"s, "map"s and other items sold or displayed in a catalog. 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:"$" );
```