**HOW TO EASILY LOAD XML WITH SQL PT 1**

Apr 26, 2019•How To

Written by Seeling Cheung, Snowflake

With this [two-part blog](https://community.snowflake.com/s/article/How-to-Easily-Load-XML-with-SQL-Pt-2), we review how to load XML data into Snowflake and query with ANSI-standard SQL. Doing so will enable you to have full relational database capabilities for XML, despite it being a document store, non-relational, data format.

And if you’re asking yourself: “XML, is it still being used?” Rest assured, the answer is a resounding, “Yes!” Even though JSON has overtaken XML as the preferred light-weight, easy-to-understand data exchange file format of choice, XML remains very well entrenched in corporate IT. For example, XML has many use cases from desktop office applications to SOA-based enterprise applications to a variety of APIs and industry-level data-exchange protocol standards. In regard to industry standards specifically, there’s HL7 for the healthcare industry, there’s FpML for financial services, and OTA for travel. All indications are XML will continue to thrive and evolve.

Given the prevalence of XML, you’ll want a data warehouse that makes it easy to query XML to ask questions of the data and to produce insights.

Other approaches require you to first transform XML before loading into a SQL database. This can add complexity or delays since you must use a separate tool to transform the XML data. Otherwise, you’ll have to custom-write the transformation scripts yourself. With Snowflake, you can ingest XML direct into the solution. No pre-transformation required. Thanks to Snowflake’s patented VARIANT data type, XML data can be stored as its own columnar database table or inserted into an existing table in Snowflake.

The following steps describe how.

**GATHERING SAMPLE XML DATA**

First, let’s gather XML data. For our sample, we use publicly available US Treasury auction data. The data can be found [here](https://www.treasurydirect.gov/xml/). This website provides a record of all Treasury auction announcements for T-bills, notes, bonds, etc., over the past decade (from 2008 to 2017) formatted as XML data. In total, the site contains about 25,000 XML files. We’ve used five as examples for this post.

Figure 1 provides a read-out of one of the XML-based, Treasury auction data files:

 Figure 1. U.S. Treasury Auction Data in XML Format

**LOADING XML DATA INTO SNOWFLAKE**

Next, we are going to load the data into Snowflake. Snowflake provides an intuitive UI, which makes it easy to load and prep the data to run queries for analysis. The steps are summarized as follows:

* 1. Create a new table with an XML column using Snowflake’s patented VARIANT data type.
  2. Create a new column-oriented file format for XML (“columnarizing” the XML), to be used with the new table.
  3. Load the XML sample data into the XML column of the new table.

The actual sequence of steps is as follows.

**Create a New Table**

* 1. Create a table with XML column with VARIANT data type.
  2. From the Snowflake UI, select the database into which the XML data will be loaded (e.g., ‘TUTORIAL\_DB’ for our example here). [To learn how to create a new database, click [here](https://docs.snowflake.net/manuals/sql-reference/sql/create-database.html)].
  3. From the database tool bar, click on Tables -> Create
  4. Enter the table name as ‘treasury\_auction\_xml’. The default schema name is ‘Public’. Specify the schema (column and data type as shown):

If you prefer to work with SQL statements for a script or a workbook, you can click on the ‘Show SQL’ link in the **Create Table** dialog box to display the equivalent SQL statements that can be cut and pasted into your script. Snowflake provides this option for most selection boxes. Doing so for this example will reveal:

**Create a New File Format for XML**

As part of the data loading process, columnarize the XML via a new file format:

* 1. Select the database ‘TUTORIAL\_DB’ and click on the table ‘treasury\_auction\_xml’.
  2. Click on **Load Table**. A series of pop-up windows will be displayed to guide the user through the data loading process. Select the warehouse you designate for this exercise. Next, specify the location of the XML files to be loaded. The dialog box for our example is shown below:
  3. Next, in the file format step, click on **+** and a pop-up window is displayed. Specify the various options as shown below and click on **Finish**:

**Complete the Loading Process**

Select **Load Options**. This is the last step to complete the loading of XML data into the new table with an XML column. A few different error handling options are presented as shown. Select the choice that is appropriate for your requirement.

* 1. For this example, we choose to skip an XML file if an error is detected in parsing. Click on **Load** to finish.
  2. Load result is displayed in the example below, showing that a number of XML files were successfully loaded into the new table ‘treasury\_auction\_xml’.

**QUERYING THE XML DATA**

With the XML data loaded, you can easily query the data in a fully relational manner, expressing queries with robust ANSI SQL. For example, for a quick glance of all of the XML documents loaded into the table, execute this query:

SELECT src\_xml FROM treasure\_auction\_xml;

This query will return all of the XML documents in the XML column. The following figure shows a portion of the output:

This is just a quick example of how you can easily query XML with Snowflake with standard SQL. In addition, you have an assortment of XML functions to query XML elements in the XML files. This includes a powerful FLATTEN table-value function, which is an un-nesting function that will allow you to access the inherent hierarchical structures within an XML file. [In part 2](https://www.snowflake.com/how-to-easily-load-and-query-xml-wata-with-snowflake-part-2/) of this post, we examine a few sample queries to easily analyze the data using the various XML functions and the semi-structured data querying feature called LATERAL FLATTEN.

**Additional Links**

* 1. [**SQL Definition**](https://www.snowflake.com/data-warehousing-glossary/sql/)

*From <*[*https://support.snowflake.net/s/article/How-to-Easily-Load-XML-with-SQL-Pt-1*](https://support.snowflake.net/s/article/How-to-Easily-Load-XML-with-SQL-Pt-1)*>*

HOW TO EASILY LOAD AND QUERY XML DATA WITH SNOWFLAKE (PART 2)

JUL 16, 2018

Author: **[Seeling Cheung](https://www.snowflake.com/blog/authors/seeling-cheung/)** | Contributing Authors: Michael Nixon

Engineering, How to Use Snowflake

With this blog, we conclude our two-part series on how to easily query XML with Snowflake SQL.

Picking up where we left off with [**Part 1**](https://www.snowflake.com/blog/easily-load-xml-sql/), with the XML data loaded, you can query the data in a fully relational manner, expressing queries with robust ANSI SQL. We can then easily issue SQL queries to gain insight into the data without transforming or pre-processing the XML.

This blog outlines a few sample queries to easily analyze the data using the various XML functions and the semi-structured data querying feature called LATERAL FLATTEN. Powerful LATERAL FLATTEN capabilities enable you to access the inherent hierarchical structures within the XML data.

**USING THE XML ELEMENT AND ATTRIBUTE FUNCTIONS**

Snowflake provides a number of SQL functions to access XML constructs like element and attributes of XML documents. The following queries show some usage examples of these functions. For a quick glance of all of the XML documents loaded into our table, execute the following query:

SELECT src\_xml FROM treasury\_auction\_xml;

* 1. This query returns the full XML document in the XML column
  2. **Example output:**

To access the attribute of an element, leverage the “@” operator in the following query:

SELECT src\_xml:"@" FROM treasury\_auction\_xml;

* 1. This query returns the name of root element of the XML document in XML column.
  2. **Example output:**

**"td:AuctionData"**

The following query leverages the “$” operator to query the contents of the root element of an XML document:

SELECT src\_xml:"$" FROM treasury\_auction\_xml;

* 1. This query returns the contents of the root element of the XML document in the XML column.
  2. **Example output:**

**USING THE FLEXIBLE XMLGET FUNCTION**

The following query retrieves the first element contained within the root element of the XML document:

SELECT XMLGET(src\_xml, 'AuctionAnnouncement', 0) FROM treasury\_auction\_xml;

* 1. This query returns the contents of the root element of the XML document in the XML column.
  2. **Example output:**

The following query retrieves all the contents of the first “AuctionAnnouncement” element:

SELECT XMLGET(src\_xml, 'AuctionAnnouncement', 0):"$" FROM treasury\_auction\_xml;

* 1. This query returns all the contents of the first element under the root element.
  2. **Example output:**

**USING THE POWERFUL LATERAL FLATTEN CAPABILITIES**

The powerful LATERAL FLATTEN table-value function is one of the most fundamental mechanism offered by Snowflake for querying and exploring semi-structured data. If you are familiar with the unnest concept function in relational database management, then you can think of FLATTEN along the same lines. Essentially, it returns a “virtual table” of {zero, one or many} values that are subsequently joined against the current row to yield {zero, one or many} rows that contain all the attributes of the original row plus the value from the virtual table.

In the following queries, we offer some examples on how to leverage this powerful functionality.

Let’s consider a query that we can use to return a row for each element included in the announcement of a Treasury security. Examples of element may include security term, offering amount, issue date, amount awarded, discount rate or interest rate.

SELECT  
auction\_announcement.index as auction\_contents\_index,  
auction\_announcement.value as auction\_contents\_value  
FROM treasury\_auction\_xml,  
LATERAL FLATTEN(to\_array(treasury\_auction\_xml.src\_xml:"$" )) xml\_doc,  
LATERAL FLATTEN(to\_array(xml\_doc.VALUE:"$" )) auction\_announcement;

* 1. In this query, LATERAL FLATTEN takes an XML input and explode the list of sub-elements into its own (denormalized) row.
  2. **Example output (Snowflake web interface):**

This example combines the usage of XMLGet with the LATERAL function:

SELECT XMLGET(value, 'SecurityType' ):"$" as "Security Type",  
XMLGET( value, 'MaturityDate' ):"$" as "Maturity Date",  
XMLGET( value, 'OfferingAmount' ):"$" as "Offering Amount",  
XMLGET( value, 'MatureSecurityAmount' ):"$" as "Mature Security Amount"  
FROM treasury\_auction\_xml,  
LATERAL FLATTEN(to\_array(treasury\_auction\_xml.src\_xml:"$" )) auction\_announcement;

* 1. In this query, LATERAL FLATTEN iterates through the child-elements of the XML document and XMLGet maps the child element “SecurityType”, or “MaturityDate”, or “OfferingAmount”, or “MatureSecurityAmount” to its own column.
  2. **Example output (Snowflake web interface):**

**SUMMARY**

With these steps and query examples over both blogs in this series, we demonstrate how straightforward it is to ingest XML data. Snowflake natively ingests semi-structured data and enables you to immediately query the data with robust ANSI SQL, without the need to first transform or pre-process the data. This approach also dramatically simplifies the process to work with semi-structured data by eliminating data preparation steps. This approach also dramatically simplifies your data architecture by eliminating the need for a second data processing platform, such as Hadoop.

Consolidate your diverse data onto Snowflake and receive a cloud-built, easy to use, software-driven SQL data warehouse-as-a-service, to boot, for your analytics.

[*https://www.snowflake.com/blog/how-to-easily-load-and-query-xml-data-with-snowflake-part-2/*](https://www.snowflake.com/blog/how-to-easily-load-and-query-xml-data-with-snowflake-part-2/)

*From <*[*https://www.snowflake.com/blog/how-to-easily-load-and-query-xml-data-with-snowflake-part-2/*](https://www.snowflake.com/blog/how-to-easily-load-and-query-xml-data-with-snowflake-part-2/)*>*