Trafodion Code Examples

Contents

Use this page to share code that you have developed for Trafodion and that could be useful for others. This could include, but is not restricted to:

- UDFs
- Stored Procedures
- Scripts to manage a Trafodion instance
- SQL queries to query Trafodion metadata

If the code is just a few lines, you could list it here. If it's multiple files, you could publish it in github and put a short description and a link here.

Also, please consider adding a license to your code in github, for example you can add an Apache license, so it is clear how your code can be used.

Stored Procedures

Scalar UDFs

Using Regular Expression in SQL SQL

This demo UDF aims to provide an example about how to write a scalar UDF, instead of providing a production-level UDF. But one can start with it to enhance it to be more useful.

Standard SQL cannot do some complex query against a string. Like to filter out a valid phone number, a valid IP address or an email address. Regular Expression is a powerful tool to do this. This demo shows you how to write a UDF to enhance/extend Trafodion to support RegExp to do some more complex queries.

The source code can be downloaded from : https://github.com/traflm/trafodion-repos/tree/master/scalar-udf

Simulate to_char Oracle function

This demo will simulate the to_char function from Oracle.

The source code can be downloaded from: https://github.com/traflm/trafodion-repos/blob/master/scalar-udf/to_date.c

Table-Mapping UDFs

Execution model of a Table-Mapping UDF (TMUDF). Syntax for invoking a TMUDF looks like this...

```
udf( <udf name> ( TABLE ( <select stmt> ) [, <parameters>] ) )
```

Result rows from <select stmt> will be presented to UDF and user UDF logic decides what to do with it.

Prior to beginning execution of <select stmt> query, user UDF logic

- examines the columns named in the select-list of <select stmt> and any <parameters> passed in the UDF call
- produces the list of output columns - this may/may not include items from the select-list of <select stmt>; the UDF can add new columns
- examines predicates and partitioning clauses in <select stmt> (as seen in the Sessionize TMUDF tutorial)
- influences statistics (as seen in Sessionize)

When <select stmt> executes. result rows are returned to the UDF for action. The UDF consumes each row and, according to its logic, outputs a row corresponding to the output columns that were defined in the previous step.

Reading Kafka Events

This UDF can read a topic from Apache Kafka. It is a basic serial Kafka consumer.

The source code can be downloaded from https://github.com/esgyn/code-examples/blob/master/src/main/java/org/trafodion/examples/udrs/udfs/table_valued/KafkaConsumerSerial

Group Concatenation

This UDF groups rows by one column and concatenates the values of one column occurring in the group, similar to the GROUP_CONCAT function in MySQL. It behaves like an aggregate or sequence function, but rather than computing min or max, etc. it computes a concatenation of the values in the group.

The source code can be downloaded from https://github.com/esgyn/code-examples/tree/master/src/main/java/org/trafodion/examples/udrs/udfs/table_valued/group_concat

JSON Column-izer

When stored in a database, JSON data is typically stored in a character column that holds the entire document, unless the JSON has been previously parsed into name-value pairs, which then presents different questions of how to store it and access it.

A Hive table DDL can depict a JSON layout including its embedded structs and arrays. For example, this JSON

{"Employee" : {"Person" : {"First name" : "John", "Middle Name" : "Quincy", "Last Name" : "Smith" }, "Phones" : {"Home" : "555-555-1212", "Mobile" : "888-555-1212"} } }

...would be declared in Hive similar to the following (but column names need adjustment - Hive has problems with backticked column names within a struct):

```
CREATE TABLE  t1 (employee STRUCT <
person STRUCT <
          `first name` : string,
          `middle name` : string,
          `last name` : string >,
phones STRUCT <
          home : string,
          mobile : string >
>)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe';
```

The serde interprets the JSON and provides access to individual JSON elements. Whether you load data into this (INTERNAL) table or define it as an EXTERNAL table and link it to file(s) in HDFS, physically, the JSON is still a single string of characters - it is not stored as multiple columns. A HiveQL SELECT statement would be like the following with JsonSerDe resolving the "column" reference to extract the correct value. Note that use of hierarchical naming is essential.

```
                SELECT employee.person.`first name` FROM t1;
```

However, when a variety of JSON formats occur within the same data stream, it becomes very cumbersome to define a single table with all the tags that accommodates all the possible forms. Challenges such as these in making JSON query-able led to this UDF.

This is a Table Mapping UDF that expects a single column containing a JSON document. Parameters passed in the call identify JSON tags whose values are to be output. For this, we show creating a new table in Hive (or a similar one can be created in Trafodion).

> hive

```
CREATE TABLE t2 (jsondata STRING);
LOAD DATA LOCAL PATH '<json text input file>'  OVERWRITE INTO TABLE t2;
```

or

> hive

> *CREATE EXTERNAL TABLE t2 (jsondata STRING) ROW FORMAT DELIMITED STORED AS TEXTFILE LOCATION '<hdfs dir containing data>';*
>
> *(files in the hdfs dir are implicitly linked to the table)*

Applying the JSON layout above, one sample invocation of the UDF is

> *SELECT * FROM udf(unjson(TABLE(select * from hive.hive.t2), 'employee.person.last name', 'employee.person.first name', 'employee.phones.mobile'));*

The SELECT-list within the TABLE() specification should resolve to a single column containing the JSON. The table referenced can be any Trafodion-supported type. There is no defined limit on the number of tag parameters included in the call. Note that the tag names match how they are spelled in the JSON document (case insensitive). The output will be three columns for each row, either with the extracted values or nulls.

The java source can be copied from: https://github.com/esgyn/code-examples/tree/master/src/main/java/org/trafodion/examples/udrs/udfs/table_valued/json_columnizer.

Trafodion steps after java compiling, jar'ing, and uploading to every node of the Trafodion cluser:

> sqlci

```
        create library <your library name> file '<the full path name of your jar>';

        create table_mapping function <what you want> ()
        external name 'org.trafodion.examples.udrs.udfs.table_valued.json_columnizer.json_columnizer'
        -- name of your class
        language java
        library <your library name>;
```

FilterProg

This is a UDF to start an arbitrary executable, send it the table-valued input of the UDF (if any) as input, and convert the standard output of the program into the result table of the UDF.

Since we don't know what data the program generates, number and types of the result columns needs to be specified as arguments.

NOTE: This UDF could be a potential security vulnerability, if not set up carefully. It restricts the executables to a "sandbox" directory, which should be carefully managed by the person who creates the UDF.

See more comments in the source file.

The source code can be downloaded from https://github.com/esgyn/code-examples/tree/master/src/main/cpp/FilterProg.

Text Search using Lucene

Lucene is an Apache project written in Java that provides text indexing and searching capabilities. In terms of function, the analogy for this particular use case is SQL's "LIKE" predicate (but Lucene offers more capabilities). This TMUDF shows the basics of how to use those APIs. A typical text index/search scenario involves many "documents" with a search result consisting of documents that match the search query. For this UDF, indexing and matching is made row-by-row, that is, the index represents just the current row. Also, an in-memory directory class is used rather than one on disk.

This example was developed with Lucene 5.5.2, a version using Java 7. Later Lucene releases (> 6.x) use Java 8 and beyond.

The java source can be copied from: https://github.com/esgyn/code-examples/tree/master/src/main/java/org/trafodion/examples/udrs/udfs/table_valued/lucene.

Manageability

Tools

SQL Scripts and Queries

Miscellaneous