**JsonSerde - a read/write SerDe for JSON Data**

Build Status:

* master :
* develop:

This library enables Apache Hive to read and write in JSON format. It includes support for serialization and deserialization (SerDe) as well as JSON conversion UDF.

**Features**

* Read data stored in JSON format
* Convert data to JSON format during INSERT INTO <table>
* Support for JSON arrays and maps
* Support for nested data structures
* Support for Cloudera's Distribution Including Apache Hadoop (CDH)
* Support for multiple versions of Hadoop

**Installation**

Download the latest binaries (json-serde-X.Y.Z-jar-with-dependencies.jar and json-udf-X.Y.Z-jar-with-dependencies.jar) from [congiu.net/hive-json-serde](http://www.congiu.net/hive-json-serde). Choose the correct verson for CDH 4, CDH 5 or Hadoop 2.3. Place the JARs into hive/lib or use ADD JAR in Hive.

**JSON Data Files**

Upload JSON files to HDFS with hadoop fs -put or LOAD DATA LOCAL. JSON records in data files must appear *one per line*, an empty line would produce a NULL record. This is because Hadoop partitions files as text using CR/LF as a separator to distribute work.

The following example will work.

{ "key" : 10 }

{ "key" : 20 }

The following example will not work.

{

"key" : 10

}

{

"key" : 20

}

**Loading a JSON File and Querying Data**

Uses [json-serde/src/test/scripts/test-without-cr-lf.json](https://github.com/rcongiu/Hive-JSON-Serde/blob/develop/json-serde/src/test/scripts/test-without-cr-lf.json).

~$ cat test.json

{"text":"foo","number":123}

{"text":"bar","number":345}

$ hadoop fs -put -f test.json /user/data/test.json

$ hive

hive> CREATE DATABASE test;

hive> CREATE EXTERNAL TABLE test ( text string )

ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'

LOCATION '/user/data';

hive> SELECT \* FROM test;

OK

foo 123

bar 345

**Querying Complex Fields**

Uses [json-serde/src/test/scripts/data.txt](https://github.com/rcongiu/Hive-JSON-Serde/blob/develop/json-serde/src/test/scripts/data.txt).

hive> CREATE DATABASE test;

hive> CREATE TABLE test (

one boolean,

three array<string>,

two double,

four string )

ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'

STORED AS TEXTFILE;

hive> LOAD DATA LOCAL INPATH 'data.txt' OVERWRITE INTO TABLE test;

hive> select three[1] from test;

gold

yellow

If you have complex json it can be tedious to create tables manually. Try [hive-json-schema](https://github.com/quux00/hive-json-schema) to build your schema from data.

See [json-serde/src/test/scripts](https://github.com/rcongiu/Hive-JSON-Serde/blob/develop/json-serde/src/test/scripts) for more examples.

**Defining Nested Structures**

ADD JAR json-serde-1.3.7-SNAPSHOT-jar-with-dependencies.jar;

CREATE TABLE json\_nested\_test (

country string,

languages array<string>,

religions map<string,array<int>>)

ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'

STORED AS TEXTFILE;

-- data : {"country":"Switzerland","languages":["German","French","Italian"],

-- "religions":{"catholic":[10,20],"protestant":[40,50]}}

LOAD DATA LOCAL INPATH 'nesteddata.txt' OVERWRITE INTO TABLE json\_nested\_test;

select \* from json\_nested\_test;

-- result: Switzerland ["German","French","Italian"] {"catholic":[10,20],"protestant":[40,50]}

select languages[0] from json\_nested\_test;

-- result: German

select religions['catholic'][0] from json\_nested\_test;

-- result: 10

**Using Arrays**

Data in JSON arrays should be ordered identically to Hive columns, similarly to text/csv.

For example, array data as follows.

["John", 26 ]

["Mary", 23 ]

Can be imported into the following table.

CREATE TABLE people (name string, age int)

Arrays can also be nested.

CREATE TABLE complex\_array (

name string, address struct<street:string,city:string>

)

-- data:

["John", { street:"10 green street", city:"Paris" } .. ]

**Importing Malformed Data**

The SerDe will raise exceptions with malformed data. For example, the following malformed JSON will raiseorg.apache.hadoop.hive.serde2.SerDeException.

{"country":"Italy","languages" "Italian","religions":{"catholic":"90"}}

Failed with exception java.io.IOException:org.apache.hadoop.hive.serde2.SerDeException:

Row is not a valid JSON Object - JSONException: Expected a ':' after a key at 32 [character 33 line 1]

This may not be desirable if you have a few bad lines you wish to ignore. Set ignore.malformed.json in that case.

ALTER TABLE json\_table SET SERDEPROPERTIES ( "ignore.malformed.json" = "true");

While this option will not make the query fail, a NULL record will be inserted instead.

NULL NULL NULL

**Promoting a Scalar to an Array**

It is a common issue to have a field that sometimes is a scalar and sometimes an array.

{ "field" : "hello", .. }

{ "field" : [ "hello", "world" ], ...

Declare your table as array<string>, the SerDe will return a one-element array of the right type, promoting the scalar.

**Support for UNIONTYPE**

A Uniontype is a field that can contain different types. Hive usually stores a 'tag' that is basically the index of the datatype. For example, if you create a uniontype<int,string,float>, a tag would be 0 for int, 1 for string, 2 for float as per the [UnionType documentation](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Types" \l "LanguageManualTypes-UnionTypes).

JSON data does not store anything describing the type, so the SerDe will try and infer it. The order matters. For example, if you define a field f as UNIONTYPE<int,string> you will get different results.

The following data will be parsed as int, since it precedes the String type in the defintion and 123 is successfully parsed as a number.

{ "f": "123" }

The following data will parsed as a String.

{ "f": "asv" }

It's worth noting that complex Union types may not be very efficient, since the SerDe may try to parse the same data in multiple ways.

**Mapping Hive Keywords**

Sometimes JSON data has attributes named like reserved words in hive. For instance, you may have a JSON attribute named 'timestamp', and hive will fail when issuing a CREATE TABLE. This SerDe can map hive columns over attributes with different names using properties.

In the following example mapping.ts translates the ts field into it the JSON attribute called timestamp.

CREATE TABLE mytable (

myfield string, ts string

) ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'

WITH SERDEPROPERTIES ( "mapping.ts" = "timestamp" )

STORED AS TEXTFILE;

**Mapping Names with Periods**

Hive doesn't support column names containing periods. In theory they should work when quoted in backtics, but doesn't, as noted in [SO#35344480](http://stackoverflow.com/questions/35344480/hive-select-column-with-non-alphanumeric-characters/35349822). To work around this issue set the property dots.in.keys to true in the SerDe Properties and access these fields by substituting the period with an underscore.

For example, create the following table.

CREATE TABLE mytable (

my\_field string,

other struct<with\_dots:string> )

ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'

WITH SERDEPROPERTIES ("dots.in.keys" = "true" )

Load the following JSON.

{ "my.field" : "value" , "other" : { "with.dots" : "blah" } }

Query data substituting periods with underscores.

SELECT my\_field, other.with\_dots from mytable

value, blah

**Case Sensitivity in mappings**

Since hive is case insensitive, all JSON keys are by default lowercased, to accomodate situations where the same JSON key is in a different case. However, this may not be what you want, you may need to treat the same key with different case as two different ones. You'll then have to use mappings, since hive does not support case sensitive columns, and you'll also have to tell the SerDe not to be case insensitive (the default).

CREATE TABLE mytable (

time1 string,

time2 string)

ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'

WITH SERDEPROPERTIES (

"case.insensitive" = "false", --tells hive to ignore key case

"mapping.time1"= "time", -- lowercase 'time' mapped into 'time1'

"mapping.time2"= "Time) -- uppercase to 'time2'

-- Data: { "time" : "2012-10-22:, "Time": "2012-11-22"}

SELECT time1,time2 from mytable

**Explicit Null Value In Serialized JSON String**

In order to be complaint with some object oriented systems an explicit 'null' json value is required in the serialized string. As default, Hive-JSON-Serde will not produce null values in the output serialized JSON string and just drop the key, if you do want to have explicit 'null' values in your output JSON string, use the following:

DROP TABLE tableWithNull;

CREATE EXTERNAL TABLE tableWithNull

(

`stringCol` STRING,

'stringNullCol' STRING,

'stringMissingCol' STRING,

'structCol' STRUCT<name : STRING>,

'structNullCol' STRUCT<name : STRING>,

'structMissingCol' STRUCT<name : STRING>

)

ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'

WITH SERDEPROPERTIES ("explicit.null" = "true");

-- JSON string: {\"stringCol\":"blabla",\"stringNullCol\":null,\"structCol\":{\"name\":\"myName\"},\"structNullCol\":{\"name\":null}}

LOAD DATA LOCAL INPATH 'pathToJsonFile.json' OVERWRITE INTO TABLE tableWithNull;

-- The output when ("explicit.null" = "true"):

-- {\"stringCol\":"blabla",\"stringNullCol\":null,\"stringMissingCol\":null,\"structCol\":{\"name\":\"myName\"},\"structNullCol\":{\"name\":null},\"structMissingCol\":null}

-- The default output or when ("explicit.null" = "false"):

-- {\"stringCol\":"blabla",\"structCol\":{\"name\":\"myName\"},\"structNullCol\":{}}

**User Defined Functions (UDF)**

**tjson**

The tjson UDF can turn array, structs or strings into JSON.

ADD JAR json-udf-X.Y.Z-jar-with-dependencies.jar;

create temporary function tjson as 'org.openx.data.udf.JsonUDF';

hive> select tjson(named\_struct("name",name)) from mytest1;

OK

{"name":"roberto"}

The SerDe must also be in the classpath for the UDF to work. If not installed as a hive extra library, you should also ADD JARtheSerDe Jar

**Timestamps**

Note that the system default timezone is used to convert timestamps.

**Contributing**

See [CONTRIBUTING](https://github.com/rcongiu/Hive-JSON-Serde/blob/develop/CONTRIBUTING.md) for how to build the project.

**History**

This library is written by [Roberto Congiu](http://www.congiu.com/) <[rcongiu@yahoo.com](mailto:rcongiu@yahoo.com)> during his time at [OpenX Technologies, Inc.](https://www.openx.com/).

See [CHANGELOG](https://github.com/rcongiu/Hive-JSON-Serde/blob/develop/CHANGELOG.md) for details.

**Thanks**

Thanks to Douglas Crockford for the liberal license for his JSON library, and thanks to my employer OpenX and my boss Michael Lum for letting me open source the code.