HIVE SERDE :

Serde will have a diiferent jar as compared to the format .

Please chech correct formatting.

{“field1”:1212,”field2”:”string”,”field3”:”int,”field4”:”string”}

{“field1”:1212,”field2”:”string”,”field3”:”int,”field4”:”string”}

{“field1”:1212,”field2”:”string”,”field3”:”int,”field4”:”string”}

{“field1”:1212,”field2”:”string”,”field3”:”int,”field4”:”string”}

{"code":"adad","name":"122323","details":{"empid":1234,"design":"mentor"}}

{"code":"adad","name":"122323","details":{"empid":1234,"design":"mentor"}}

If the format is key and value file or nested key and value file then use this jar

Json key and value tables example.

Follow this command

Add jar json-serde-1.3.1-SNAPSHOT-jar-with-dependencies.jar

This is for the first table which is not nested .

Create table keyvalue(field1 int,field2 string,field3 int,field4 string)

* Row Format SERDE ‘org.openx.data.jsonserde.JsonSerDe’;

Load data inpath ‘/user/hue/keyandvalue.json’ into table keyvalue;

For the nested key and value which is the second table format goes like

Create table nested\_key\_value

* (
* code string,
* name int,
* details struct<empid:int,design:string>
* )
* ROW FORMAT SERDE ‘org.openx.data.jsonserde.JsonSerDe’;

Load data inpath ‘/user/hue/keyandvalue.json’ into table nested\_key\_value;

Now serde for xml files

There is a separate jar for XML files in serde.

Add jar hivexmlserde-1.0.5.3.jar;

<record customer\_id="0000-JTALA">

<income>200000</income>

<demographics>

<gender>F</gender>

<agecat>1</agecat>

<edcat>1</edcat>

<jobcat>2</jobcat>

<empcat>2</empcat>

<retire>0</retire>

<jobsat>1</jobsat>

<marital>1</marital>

<spousedcat>1</spousedcat>

<residecat>4</residecat>

<homeown>0</homeown>

<hometype>2</hometype>

<addresscat>2</addresscat>

</demographics>

<financial>

<income>18</income>

<creddebt>1.003392</creddebt>

<othdebt>2.740608</othdebt>

<default>0</default>

</financial>

</record>

create table xmltable(customer\_id STRING, income BIGINT

, demographics map<string,string>, financial map<string,string>)

* ROW FORMAT SERDE ‘com.ibm.spss.hive.serde2.xml.XmlSerDe’
* WITH SERDEPROPERTIES (
* “column.xpath.customer\_id”=”/record/@customer\_id”, here we used @bcz its

beausce is a single value

right after records.

* “column.xpath.income”=”/record/income/text()”, takes text of single file
* “column.xpath.demographics”=”/record/demographics/\*”, star means select all.
* “column.xpath.financial”=”/record/financial/\*”
* )
* STORED AS
* INPUTFORMAT ‘com.ibm.spss.hive.serde2.xml.XmlInputFormat’
* OUTPUTFORMAT'org.apache.hadoop.hive.ql.io.IgnoreKeyTextOutputFormat'
* TBLPROPERTIES(
* “xmlinput.start”=”<record customer\_id”, reads from the starting from colum name.
* “xmlinput.end”=” </record>”
* );

load data inpath ‘/user/hue/xmlfile’ into table xmltable;

select\*from xmltable ;

if you close the section after starting hive you again have to add jar for the xml serde and other serde jars and then use the select command for their respective table.

This statement will select the tag sub values under the tag values.

select demographics['empcat'],demographics['jobcat'],demographics['gender'] from xml\_serde;

Select financial[‘income’] , demographics[‘gender’] from xml\_serde;

Passing the parameter.

Set MYINCOME = 1000;

Select \* from xml\_serde where income > ${hiveconf:MYINCOME};

This will display income higher than the values passed in the parameter.

create table if not exists xml(result map<string,string>)

> ROW FORMAT SERDE 'com.ibm.spss.hive.serde2.xml.XmlSerDe'

> WITH SERDEPROPERTIES(

> "column.xpath.result"="/record/result/\*",

> "xml.map.specification.entry"="@name->#content"

> )

> STORED AS

> INPUTFORMAT 'com.ibm.spss.hive.serde2.xml.XmlInputFormat'

> OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.IgnoreKeyTextOutputFormat'

> TBLPROPERTIES(

> "xmlinput.start"="<record",

> "xmlinput.end"="</record>"

> );

<record>

<result>

<entry name = “key1”>value1</entry>

<entry name =”key2”>value2</entry>

<entry name = “key3”>value3</entry>

</result>

</record>

answer : {"key3":"value3","key2":"value2","key1":"value1"}

<record myname = “Adylad”>

<result>

<entry name = “key1”>value1</entry>

<entry name =”key2”>value2</entry>

<entry name = “key3”>value3</entry>

</result>

</record>

create table if not exists xml1(myname string, result map<string,string>)

> ROW FORMAT SERDE 'com.ibm.spss.hive.serde2.xml.XmlSerDe'

> WITH SERDEPROPERTIES(

> "column.xpath.myname"="/record/@myname",

> "column.xpath.result"="/record/result/\*",

> "xml.map.specification.entry"="@name->#content"

> )

> STORED AS

> INPUTFORMAT 'com.ibm.spss.hive.serde2.xml.XmlInputFormat'

> OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.IgnoreKeyTextOutputFormat'

> TBLPROPERTIES(

> "xmlinput.start"="<record myname",

> "xmlinput.end"="</record>"

> );

answer: Adylad {"key3":"value3","key2":"value2","key1":"value1"}

<record myname = “Adylad”>

<result>

<entry name = “key1” value=”value1”/>

<entry name =”key2” value=”value2”/>

<entry name = “key3” value=”value3”/>

</result>

</record

create table if not exists xml1(myname string, result map<string,string>)

> ROW FORMAT SERDE 'com.ibm.spss.hive.serde2.xml.XmlSerDe'

> WITH SERDEPROPERTIES(

> "column.xpath.myname"="/record/@myname",

> "column.xpath.result"="/record/result/\*",

> "xml.map.specification.entry"="@name->@value"

> )

> STORED AS

> INPUTFORMAT 'com.ibm.spss.hive.serde2.xml.XmlInputFormat'

> OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.IgnoreKeyTextOutputFormat'

> TBLPROPERTIES(

> "xmlinput.start"="<record myname",

> "xmlinput.end"="</record>"

> );

result MAP<STRING,STRING>,

....

"column.xpath.result"="/record/result",

"xml.map.specification.entry1"="@name->entry1"

"xml.map.specification.entry2"="@name->entry2"

"xml.map.specification.entry3"="@name->entry3"

* **XML data**

<record>

<result>

<entry1 name=”key1”/>

<entry2 name=”key2”/>

<entry3 name=”key3”/>

</result>

</record>

create table nestedjson > ( > DocId string, > User struct<Id:int,Username:string,Name:string, > ShippingAddress:struct<Address1:string,Address2:string,City:string,State:string>,

> Orders:array<struct<ItemId:int,OrderDate:string>>> > )

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

OK

Time taken: 0.296 seconds

hive> load data inpath '/user/hue/jsontest.txt' into table nestedjson;

Loading data to table default.nestedjson

Table default.nestedjson stats: [numFiles=1, totalSize=263]

OK

Time taken: 0.888 seconds

hive> select \* from nestedjson;

OK

ABC {"id":1234,"username":"sam1234","name":"Sam","shippingaddress":{"address1":"123 Main St.","address2":null,"city":null,"state":"NC"},"orders":[{"itemid":6789,"orderdate":"11/11/2012"},{"itemid":4352,"orderdate":"12/12/2012"}]}

Time taken: 0.169 seconds, Fetched: 1 row(s)

hive> select DocId from nestedjson;

OK

ABC

Time taken: 0.19 seconds, Fetched: 1 row(s)

hive> select DocId ,User.Id from nestedjson;

OK

ABC 1234

Time taken: 0.204 seconds, Fetched: 1 row(s)

hive> select DocId ,User.id from nestedjson;

OK

ABC 1234

Time taken: 0.169 seconds, Fetched: 1 row(s)

hive> select DocId ,User.id,user.shippingaddress.city as city from nestedjson;

OK

ABC 1234 NULL

Time taken: 0.154 seconds, Fetched: 1 row(s)

hive> select DocId ,User.id,user.shippingaddress.city as city from nestedjson;

OK

ABC 1234 NULL

Time taken: 0.15 seconds, Fetched: 1 row(s)

hive>

> select DocId ,User.id,user.shippingaddress.city as city,user.orders[0].itemid from nestedjson;

OK

ABC 1234 NULL 6789

Time taken: 0.128 seconds, Fetched: 1 row(s)

hive>

hive> create table jsons

> (

> messageId string,

> messageSize int,

> sender string,

> recipients array<string>,

> messageParts array<struct<extension:string,size:int>>,

> headers map<string,string>

> )

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

OK

Time taken: 0.291 seconds

hive> load data inpath '/user/hue/testjsonarray.txt' into table jsons;

Loading data to table default.jsons

Table default.jsons stats: [numFiles=1, totalSize=292]

OK

Time taken: 1.317 seconds

hive> select \* from jsons;

OK

34dd0d3c-f53b-11e0-ac12-d3e782dff199 12345 alice@example.com ["joe@example.com","bob@example.com"] [{"extension":"pdf","size":4567},{"extension":"jpg","size":9451}] {"x-broadcast-id":"9876","received-spf":"pass"}

{"messageId": "34dd0d3c-f53b-11e0-ac12-d3e782dff199",

"messageSize": 12345,

"sender": "alice@example.com",

"recipients": ["joe@example.com", "bob@example.com"],

"messageParts": [

{

"extension": "pdf",

"size": 4567

},

{

"extension": "jpg",

"size": 9451

}

],

"headers": {

"Received-SPF": "pass",

"X-Broadcast-Id": "9876"

}

}

sort the file in this format

{"messageId":"34dd0d3c-f53b-11e0-ac12-d3e782dff199","messageSize":12345,"sender":"alice@example.com","recipients":["joe@example.com","bob@example.com"],"messageParts":[{"extension":"pdf","size": 4567},{"extension":"jpg","size":9451}],"headers":{"Received-SPF":"pass","X-Broadcast-Id":"9876"}}

create table if not exists complex1

(

key string,

columns array <array<string>>

)

ROW FORMAT SERDE ‘org.openx.data.jsonserde.JsonSerDe’;

{"key": "somehashvalue","columns": [["Event:2014-03-26 00\\:29\\:13+0200:json","{\"user\":{\"credType\":\"ADDRESS\",\"credValue\":\"01:AA:A4:G1:HH:UU\",\"cAgent\":null,\"cType\":\"ACE\"},\"timestamp\":1395786553,\"sessionId\":1395785353,\"className\":\"Event\",\"subtype\":\"CURRENTLYACTIVE\",\"vType\":\"TEST\",\"vId\":1235080,\"eType\":\"CURRENTLYACTIVE\",\"eData\":\"1\"}",1395786553381001],["Event:2014-03-26 00\\:29\\:13+0200:","",1395786553381001]]}