Go to \etc\hive\conf\hive-site.xml and add property **hive.cli.print.header=true**

|  |
| --- |
| <property>  <name>hive.cli.print.header</name>  <value>true</value>  </property> |

This property prints the column name when we execute select query on hive prompt

# Built in function #1: get\_json\_object

Create a simple.json file like below

|  |
| --- |
| {"Foo":"ABC","Bar":"20090101100000","Quux":{"QuuxId":1234,"QuuxName":"Sam"}} |

Create a hive table

|  |
| --- |
| CREATE TABLE json\_table ( json string ); |

Load simple.json into table

|  |
| --- |
| LOAD DATA LOCAL INPATH '/home/cloudera/user/training/simple.json' INTO TABLE json\_table; |

Query the table using get\_json\_object UDF

|  |
| --- |
| hive (experiments)> select get\_json\_object(json\_table.json, '$') from json\_table;  OK  \_c0  {"Foo":"ABC","Bar":"20090101100000","Quux":{"QuuxId":1234,"QuuxName":"Sam"}}  Time taken: 0.151 seconds, Fetched: 1 row(s) |

Here $ means root object, so basically we are fetching root object same like **select \* from json\_table**

if we want to fetch the data individually we can write query something like below

|  |
| --- |
| hive (experiments)> select get\_json\_object(json\_table.json, '$.Foo') as foo,  > get\_json\_object(json\_table.json, '$.Bar') as bar,  > get\_json\_object(json\_table.json, '$.Quux.QuuxId') as qid,  > get\_json\_object(json\_table.json, '$.Quux.QuuxName') as qname  > from json\_table;  OK  foo bar qid qname  ABC 20090101100000 1234 Sam  Time taken: 0.117 seconds, Fetched: 1 row(s) |

This works and has a nice JavaScript like "dotted" notation, but notice that you have to parse the same document once for every field you want to pull out of your JSON document, so it is rather inefficient

# Built in function #2: json\_tuple

So let's see what json\_tuple looks like. It has the benefit of being able to pass in multiple fields, but it only works to a single level deep. You also need to use Hive's slightly odd LATERAL VIEW notation

|  |
| --- |
| select v.foo, v.bar, v.quux, v.qid  from json\_table jt  LATERAL VIEW json\_tuple(jt.json, 'Foo', 'Bar', 'Quux', 'Quux.QuuxId') v  as foo, bar, quux, qid; |

This returns:

foo bar quux qid

ABC 20090101100000 {"QuuxId":1234,"QuuxName":"Sam"} NULL

**It doesn't know how to look inside the Quux subdocument** and that is why qid is null. And this is where json\_tuple gets clunky fast - you have to create another lateral view for each subdocument you want to descend into:

|  |
| --- |
| select v1.foo, v1.bar, v2.qid, v2.qname  from json\_table jt  LATERAL VIEW json\_tuple(jt.json, 'Foo', 'Bar', 'Quux') v1  as foo, bar, quux  LATERAL VIEW json\_tuple(v1.quux, 'QuuxId', 'QuuxName') v2  as qid, qname; |

This gives us the output we want:

foo bar qid qname

ABC 20090101100000 1234 Sam

With a complicated highly nested JSON doc, json\_tuple is also quite inefficient and clunky as hell. So let's turn to a custom SerDe to solve this problem.

First you need to download the jar and add it to hive

<http://www.congiu.net/hive-json-serde/1.3.6-SNAPSHOT/cdh4/>



Hive > add jar /home/cloudera/user/training/json-serde-1.3.6-jar-with-dependencies.jar;

|  |
| --- |
| CREATE TABLE json\_serde (  Foo string,  Bar string,  Quux struct<QuuxId:int, QuuxName:string>  )  ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe' |

|  |
| --- |
| LOAD DATA LOCAL INPATH '/home/cloudera/user/training/simple.json' INTO TABLE json\_serde; |

The query to match the above examples is beautifully simple:

|  |
| --- |
| SELECT Foo, Bar, Quux.QuuxId, Quux.QuuxName  FROM json\_serde; |

Result:

foo bar quuxid quuxname

ABC 20090101100000 1234 Sam

And now let's do a more complex JSON document:

|  |
| --- |
| {    "DocId": "ABC",    "User": {      "Id": 1234,      "Username": "sam1234",      "Name": "Sam",      "ShippingAddress": {        "Address1": "123 Main St.",        "Address2": null,        "City": "Durham",        "State": "NC"      },      "Orders": [        {          "ItemId": 6789,          "OrderDate": "11/11/2012"        },        {          "ItemId": 4352,          "OrderDate": "12/12/2012"        }      ]    }  } |

Collapsed version: put it in one line and save as a file namely complex.json

|  |
| --- |
| {"DocId":"ABC","User":{"Id":1234,"Username":"sam1234","Name":"Sam","ShippingAddress":{"Address1":"123 Main St.","Address2":"","City":"Durham","State":"NC"},"Orders":[{"ItemId":6789,"OrderDate":"11/11/2012"},{"ItemId":4352,"OrderDate":"12/12/2012"}]}} |

Now create a hive table for this json

|  |
| --- |
| CREATE TABLE complex\_json (    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'; |

|  |
| --- |
| LOAD DATA LOCAL INPATH '/home/cloudera/user/training/complex.json' INTO TABLE complex\_json; |

|  |
| --- |
| SELECT DocId, User.Id, User.ShippingAddress.City as city,  User.Orders[0].ItemId as order0id,  User.Orders[1].ItemId as order1id  FROM complex\_json; |

Result:

docid id city order0id order1id

ABC 1234 Durham 6789 4352

But what if we don't know how many orders there are and we want a list of all a user's order Ids? This will work

|  |
| --- |
| SELECT DocId, User.Id, User.Orders.ItemId  FROM complex\_json; |

Result:

docid id itemid

ABC 1234 [6789,4352]

Finally, does the openx JsonSerDe require me to define the whole schema? Or what if I have two JSON docs (say version 1 and version 2) where they differ in some fields? How constraining is this Hive schema definition?

Let's add two more JSON entries to our JSON document - the first has no orders; the second has a new "PostalCode" field in Shipping Address.

|  |
| --- |
| {      "DocId": "ABC",      "User": {          "Id": 1235,          "Username": "fred1235",          "Name": "Fred",          "ShippingAddress": {              "Address1": "456 Main St.",              "Address2": "",              "City": "Durham",              "State": "NC"          }      }  }    {      "DocId": "ABC",      "User": {          "Id": 1236,          "Username": "larry1234",          "Name": "Larry",          "ShippingAddress": {              "Address1": "789 Main St.",              "Address2": "",              "City": "Durham",              "State": "NC",              "PostalCode": "27713"          },          "Orders": [              {                  "ItemId": 1111,                  "OrderDate": "11/11/2012"              },              {                  "ItemId": 2222,                  "OrderDate": "12/12/2012"              }          ]      }  } |

Collapse Version: save it in complex1.json

|  |
| --- |
| {"DocId":"ABC","User":{"Id":1235,"Username":"fred1235","Name":"Fred","ShippingAddress":{"Address1":"456 Main St.","Address2":"","City":"Durham","State":"NC"}}}  {"DocId":"ABC","User":{"Id":1236,"Username":"larry1234","Name":"Larry","ShippingAddress":{"Address1":"789 Main St.","Address2":"","City":"Durham","State":"NC","PostalCode":"27713"},"Orders":[{"ItemId":1111,"OrderDate":"11/11/2012"},{"ItemId":2222,"OrderDate":"12/12/2012"}]}} |

Load the file

|  |
| --- |
| LOAD DATA LOCAL INPATH '/home/cloudera/user/training/complex1.json' INTO TABLE complex\_json; |

|  |
| --- |
| hive (experiments)> **SELECT DocId, User.Id, User.Orders.ItemId**  **> FROM complex\_json;**  OK  docid id itemid  ABC 1234 [6789,4352]  ABC 1235 NULL  ABC 1236 [1111,2222] |

Any field not present will just return null, as Hive normally does even for non-JSON formats.

Note that we cannot query for User.ShippingAddress.PostalCode because we haven't put it on our Hive schema. You would have to revise the schema and then reissue the query.

# Load json data in ORC format

|  |
| --- |
| create table complex\_json\_orc  stored as orc  as select \* from complex\_json; |

Check file in warehouse

|  |
| --- |
| hive (experiments)> dfs -ls /user/hive/warehouse/experiments.db/complex\_json\_orc;  Found 1 items  -rwxrwxrwx 1 cloudera supergroup 1355 2016-10-11 22:34 /user/hive/warehouse/experiments.db/complex\_json\_orc/000000\_0  hive (experiments)> |

|  |
| --- |
| hive (experiments)> SELECT DocId, User.Id, User.Orders.ItemId from complex\_json\_orc;  OK  docid id itemid  ABC 1234 [6789,4352]  ABC 1235 NULL  ABC 1236 [1111,2222]  Time taken: 0.111 seconds, Fetched: 3 row(s) |