JSON Format Handling

JSON dan XML

01 Fitur Pendukung

Array

Array is a Complex data types used to store the list of elements.

```
Complex Types | Description
--------------------------
Arrays        | ARRAY<data_type>
```

```
SELECT ARRAY(1,2,3) □ [1,2,3]
```

```
SELECT ARRAY("us","ca","fr") □ ["us","ca","fr"]
```

```
SELECT ARRAY(ARRAY("id","my","sg"),ARRAY("us","ca","fr"))
□ [["id","my","sg"],["us","ca","fr"]]
```

Split

Split the input string str by the regular pattern specified.

```
Return Type | Name (Signature)
------------|---------------------------

array       | split(string str, string pat)
```

```
SELECT SPLIT('1,2,3',’,’) □ ["1","2","3"]

SELECT SPLIT('us,ca,fr',',’) □ ["us","ca","fr"]

SELECT ARRAY(ARRAY("id","my","sg"),ARRAY("us","ca","fr"))
□ [["id","my","sg"],["us","ca","fr"]]
```

Explode

explode() takes in an array (or a map) as an input and outputs the elements of the array (map) as separate rows.

Output:
a UDTF (user-defined table generating functions) generates zero or more output rows for each input row.

```
SELECT
EXPLODE(ARRAY(ARRAY("id","my","sg"),ARRAY("us","ca","fr")))
```

```
["id","my","sg"]
["us","ca","fr"]
```

Posexplode

posexplode() is similar to explode but instead of just returning the elements of the array it returns the element as well as its position in the original array.

```
SELECT
POSEXPLODE(ARRAY(ARRAY("id","my","sg"),ARRAY("us","ca","fr")))
```

```
Pos | val
------------------------
0   | ["id","my","sg"]
1   | ["us","ca","fr"]
```

LATERAL VIEW

A lateral view first applies the UDTF(user-defined table generating functions) to each row of base table and then joins resulting output rows to the input rows to form a virtual table having the supplied table alias.

```
SELECT a.col
FROM (SELECT 0) t
LATERAL VIEW
EXPLODE(ARRAY(ARRAY("id","my","sg"),ARRAY("us","ca","fr"))) a AS col
```

```
["id","my","sg"]
["us","ca","fr"]
```

MULTIPLE LATERAL VIEW

A FROM clause can have multiple LATERAL VIEW clauses. Subsequent LATERAL VIEWS can reference columns from any of the tables appearing to the left of the LATERAL VIEW.

```
SELECT a.pos,a.col,b.name
FROM (SELECT 0) t LATERAL VIEW
POSEXPLODE(ARRAY(ARRAY("id","my","sg"),ARRAY("us","ca","fr"))) a AS pos,col
LATERAL VIEW EXPLODE(a.col) b AS name WHERE a.pos=1
```

```
Pos    col                        name
-----------------------------------
1 ["us","ca","fr"] us
1 ["us","ca","fr"] ca
1 ["us","ca","fr"] fr
```

LATERAL VIEW OUTER

The user can specify the optional OUTER keyword to generate rows even when a LATERAL VIEW usually would not generate a row.

```
SELECT t.col_dummy,a.arr_dummy
FROM (SELECT 'a' col_dummy UNION ALL
SELECT 'b' col_dummy UNION ALL
SELECT 'c' col_dummy) t
LATERAL VIEW OUTER EXPLODE(ARRAY()) a AS arr_dummy
```

```
col_dummy | arr_dummy
----------------------
a         | NULL
b         | NULL
c         | NULL
```

Struct

Struct is a Complex data types used for parent and child assosiations.

```
Complex Types | Description
-------------------------------------------------------------------------
Structs       | STRUCT<col_name : data_type [COMMENT col_comment],
...>
```

```
SELECT named_struct("name","Jhon","work_place",ARRAY('New
York','Washington'),"sex_age",named_struct("sex","Male","age",
25)) AS emp_profile;
```

{"name":"Jhon","work_place":["New York","Washington"],"sex_age":{"sex":"Male","age":25}}

Inline

The inline function will do 2 things here:

1.Explode the json into as many rows as there are array members.

2.Create a new column for each JSON key that exists on the top level of the array members.

```
SELECT
INLINE(ARRAY(named_struct("name","john","sex","Male","age",25),named_struct("name"
,"lidya","sex","Female","age",15)))
```

```
name    sex       age
---------------------
john Male 25
lidya Female 15
```

02 Fitur Json

Get_json_object

Extract json object from a json string based on json path specified, and return json string of the extracted json object. It will return null if the input json string is invalid. **get_json_object**(json_txt, **path**)

```
SELECT GET_JSON_OBJECT(src_json,'$.store.fruit[1].weight')
store_fruit_1_weight FROM (SELECT
'{"store":{"fruit":[{"weight":8,"type":"apple"},
                    {"weight":9,"type":"pear"}]}}' src_json) a
9
```


Json_tuple

json_tuple(jsonStr, p1, p2, ..., pn) - like get_json_object, but it takes multiple names and return a tuple.
All the input parameters and output column types are string.

```
SELECT JSON_TUPLE(src_json,'email','owner')
FROM (SELECT '{"store":
{"fruit":[{"weight":8,"type":"apple"},{"weight":9,"type":"pear"}
],"bicycle":{"price":19.95,"color":"red"}},
"email":"amy@only_for_json_udf_test.net",
"owner":"amy"}' src_json) a
```

amy@only_for_json_udf_test.net amy

Json Serde – What is SerDe?

Serialization and deserialization formats are popularly known as SerDes. **Hive allows the framework to read or write data in a particular format**. These formats parse the structured or unstructured data bytes stored in HDFS in accordance with the schema definition of Hive tables.

Json Serde – Hive Table Json Format SerDe

Sample Data:

```
{"store":{"fruit":[{"weight":8,"type":"apple"},{"weight":9,"type":"pear"}],
"bicycle":{"price":19.95,"color":"red"}},
"email":"amy@only_for_json_udf_test.net",
"owner":"amy"
}
```

```
CREATE TABLE IF NOT EXISTS it_json_serde
(store struct <fruit:ARRAY<struct<weight:INT,type:STRING>>,
bicycle:struct<price:DECIMAL(4,2),color:STRING>>, email STRING,owner STRING)
ROW FORMAT SERDE 'org.apache.hive.hcatalog.data.JsonSerDe'
STORED AS TEXTFILE
```

Json Serde – Insert Data Using named_struct #1

```
SELECT
named_struct("fruit",ARRAY(named_struct("weight",8,"type","apple"),named_struct("w
eight",9,"type","pear"))
,"bicycle",named_struct("price",19.95,"color","red")) store
,"amy@only_for_json_udf_test.net" email
,"amy" owner
```

```
Store                                          email                          owner
-----------------------------------------------------------------------------------
[{"fruit":[{"weight":8,"type":"apple"},    | amy@only_for_json_udf_test.net | amy
{"weight":9,"type":"pear"}],               |                                |
"bicycle":{"price":19.95,"color":"red"}}] |                                |
```


Json Serde – Insert Data Using named_struct #2

```
INSERT INTO it_json_serde (store,email,owner) SELECT
named_struct("fruit",ARRAY(named_struct("weight",8,"type","apple"),named_struct("
w eight",9,"type","pear"))
,"bicycle",named_struct("price",19.95,"color","red")) store
,"amy@only_for_json_udf_test.net" email
,"amy" owner
```

Json Serde – Query Hive Table Json Format SerDe #1

```
SELECT store
,email
,owner
,store.fruit store_fruit
,store.bicycle store_bicycle
,store.fruit[1] store_fruit_1
,store.bicycle.color store_bicycle_color
,store.fruit[1].weight store_fruit_1_weight
FROM it_json_serde
```


Json Serde – Query Hive Table Json Format SerDe #2

```
Store                email              name    store_fruit
-------------------------------------------------------------------
[{"fruit":       | amy@                   | amy     | [{"weight":8,"type":"apple"},
[{"weight":8,    | only_for_json_udf_test |         | {"weight":9,"type":"pear"}]
"type":"apple"}  | .net                   |         |
,{"weight":9,    |                        |         |
"type":"pear"}], |                        |         |
"bicycle":       |                        |         |
{"price":19.95,  |                        |         |
"color":"red"}}] |                        |         |


store_bicycle      store_fruit_1      store_bicycle_color    store_fruit_1_weight
-------------------------------------------------------------------
[{"price":19.95, | [{"weight":9,                | red                 |  9
"color":"red"}]  |  "type":"pear"}]             |                     |
```

Json Serde – Query Hive Table Json Format SerDe #3

```
SELECT a.store.fruit store_fruit
,b.fruits store_fruits
,store.fruit[b.pos_fruits].type store_fruits_type
,a.store.bicycle store_bicycle
,store.bicycle.price store_bicycle_price
,store.bicycle.color store_bicycle_color
,a.owner
FROM it_json_serde a
LATERAL VIEW POSEXPLODE (a.store.fruit) b AS pos_fruits, fruits
```


Json Serde – Query Hive Table Json Format SerDe #4

a.store_fruit	b.store_fruits	a.store_fruits_type [b.pos]	store_bicycle	store_bicycle_price	store_bicycle_color	owner
[{"weight":8, "type":"apple"}, {"weight":9, "type":"pear"}]	[{"weight":8, "type":"apple"}]	apple	[{"price":19.95, "color":"red"}]	19.95	red	amy
[{"weight":8, "type":"apple"}, {"weight":9, "type":"pear"}]	[{"weight":9, "type":"pear"}]	pear	[{"price":19.95, "color":"red"}]	19.95	red	amy

Json Serde – External Hive Table Json Format SerDe #1

File Json: json_sample.json (harus dalam format single line)


```
{"store":{"fruit":[{"weight":8,"type":"apple"},{"weight":9,"type":"pear"
}],"bicycle":{"price":19.95,"color":"red"}},"email":"amy@only_for_json_u
df_test.net","owner":"amy"}
```


Json Serde - External Hive Table Json Format SerDe #2

Upload file Json ke HDFS:

```
hdfs dfs -mkdir –p ./data/external/json_sample

hadoop fs -put /home/yava/dataset/json_sample.json
/user/yava/data/external/json_sample
```

Json Serde - External Hive Table Json Format SerDe #3

```
CREATE EXTERNAL TABLE IF NOT EXISTS it_json_serde_ext (store
struct <fruit:ARRAY<struct<weight:INT,type:STRING>>,
bicycle:struct<price:DECIMAL(4,2),color:STRING>>
,email STRING
,owner STRING)
ROW FORMAT SERDE 'org.apache.hive.hcatalog.data.JsonSerDe'
STORED AS TEXTFILE
LOCATION '/user/yava/data/external/json_sample'
```


Json Serde - External Hive Table Json Format SerDe #4

```
SELECT
a.store.fruit store_fruit ,b.fruits store_fruits
,store.fruit[b.pos_fruits].type store_fruits_type
,a.store.bicycle store_bicycle
,store.bicycle.price store_bicycle_price
,store.bicycle.color store_bicycle_color
,a.owner
FROM it_json_serde_ext a
LATERAL VIEW POSEXPLODE (a.store.fruit) b AS pos_fruits, fruits
```

Json Serde - External Hive Table Json Format SerDe #5

a.store_fruit	b.store_fruits	a.store_fruits_type [b.pos]	store_bicycle	store_bicycle_price	store_bicycle_color	owner
[{"weight":8, "type":"apple"}, {"weight":9, "type":"pear"}]	[{"weight":8, "type":"apple"}]	apple	[{"price":19.95, "color":"red"}]	19.95	red	amy
[{"weight":8, "type":"apple"}, {"weight":9, "type":"pear"}]	[{"weight":9, "type":"pear"}]	pear	[{"price":19.95, "color":"red"}]	19.95	red	amy

03 Json Transform

Json Transform - get_json_object & json_tuple #1

Sumber Data Json:


```
{"_id":"1","name":"abc","attribs":[{"minutes":0,"name":"sedentar
y"},{"minutes":0,"name":"lightly"},{"minutes":0,"name":"fairly"}
,{"minutes":28,"name":"very"}],"validated":true}
```


Hasil Transformasi (Baris & Kolom):

id	name	validated	attribs_minutes	attribs_name
1	abc	true	0	sedentary
1	abc	true	1	lightly
1	abc	true	2	fairly
1	abc	true	3	very

Json Transform - get_json_object & json_tuple #2

```
SELECT a.id, a.name, a.validated,
get_json_object(t.day_data,'$.attribs['||b.index||'].minutes')
attribs_minutes,b.attribs_name
FROM (SELECT
'{"_id":"1","name":"abc","attribs":[{"minutes":0,"name":"sedentary"},{"m
inutes":0,"name":"lightly"},{"minutes":0,"name":"fairly"},{"minutes":28,
"name":"very"}],"validated":true}' day_data) t
LATERAL VIEW json_tuple(t.day_data,'_id','name','validated') a as id,
name, validated
LATERAL VIEW posexplode(split(regexp_extract(get_json_object
(t.day_data,'$.attribs[*].name'),'^\\["(.*)\\"]$'),'","')) b as index,
attribs_name
```

Json Transform - get_json_object & json_tuple #3

```
 id | name | validated | attribs_minutes | attribs_name ----------------
-------------------------------------
1   | abc  | true      | 0               | sedentary
1   | abc  | true      | 1               | lightly
1   | abc  | true      | 2               | fairly
1   | abc  | true      | 3               | very
```

Json Transform - Hive Table Json Format SerDe #1

Sumber Data Json:


```
{"data":
{"receipt_time":"2018-09-28T10:00:00.000Z",
"site":"LosAngeles",
"measures":[{"test_id":"C23_PV",
"metrics":[{"val1":[0.76,0.75,0.71],
"temp":[0,2,5],
"TS":[1538128801336,1538128810408,1538128818420]}]},
{"test_id":"HBI2_XX",
"metrics":[{"val1":[0.65,0.71],
"temp":[1,-7],
"TS":[1538128828433,1538128834541]}]}]}
}
```

Json Transform - Hive Table Json Format SerDe #2

Target Hasil Transformasi (Baris & Kolom):

receipt_time	site	test_id	val1	temp	TS
2018-09-28T10:00:00.000Z	Los Angeles	C23_PV	0.76	0	1538128801336
2018-09-28T10:00:00.000Z	Los Angeles	C23_PV	0.75	2	1538128810408
2018-09-28T10:00:00.000Z	Los Angeles	C23_PV	0.71	5	1538128818420
2018-09-28T10:00:00.000Z	Los Angeles	HBI2_XX	0.65	1	1538128828433
2018-09-28T10:00:00.000Z	Los Angeles	HBI2_XX	0.71	-7	1538128834541

Json Transform – External Hive Table Json Format SerDe #1

Multi Line Json File: json_sample_serde.json


```
{"data":
{"receipt_time":"2018-09-28T10:00:00.000Z",
"site":"LosAngeles",
"measures":[{"test_id":"C23_PV",
"metrics":[{"val1":[0.76,0.75,0.71],
"temp":[0,2,5],
"TS":[1538128801336,1538128810408,1538128818420]}]},
{"test_id":"HBI2_XX",
"metrics":[{"val1":[0.65,0.71],
"temp":[1,-7],
"TS":[1538128828433,1538128834541]}]}]}
}
```

Json Transform – External Hive Table Json Format SerDe #2

Upload **Multi Line Json File** ke HDFS:

```
hdfs dfs -mkdir –p ./data/external/json_sample_serde
```

```
hadoop fs -put /home/yava/dataset/json_sample_serde.json
/user/yava/data/external/json_sample_serde
```

```
hadoop fs -text /user/yava/data/external/json_sample_serde/* | wc -l
13
```

```
hadoop fs -text /user/yava/data/external/json_sample/* | wc -l
1
```

Json Transform – External Hive Table Json Format SerDe #3

```
CREATE EXTERNAL TABLE IF NOT EXISTS it_json_serde_multi ( data
struct<receipt_time: STRING, site: STRING, measures: ARRAY<struct<test_id: STRING,
metrics: ARRAY<struct<val1: ARRAY<DOUBLE>, temp: ARRAY<SMALLINT>,
TS: ARRAY<BIGINT>>>>>>)
ROW FORMAT SERDE 'org.apache.hive.hcatalog.data.JsonSerDe'
LOCATION '/user/yava/data/external/json_sample_serde'
TBLPROPERTIES('transactional'='false','textinputformat.record.delimiter'='#');
```

*** Using Multi Line Json File:**

```
TBLPROPERTIES('textinputformat.record.delimiter'='#')
```

Json Transform – External Hive Table Json Format SerDe #4

```
SELECT e.receipt_time, e.site, e.test_id, e.val1[f.pos] val1,
e.temp[f.pos] temp, e.ts[f.pos] TS
FROM (SELECT c.receipt_time, c.site, c.test_id, d.val1, d.temp, d.ts
FROM (SELECT a.data.receipt_time receipt_time, a.data.site site,
b.test_id test_id, b.metrics metrics
FROM it_json_serde_multi a
LATERAL VIEW OUTER INLINE(a.data.measures) b AS test_id,metrics) c
LATERAL VIEW OUTER INLINE(c.metrics) d AS val1, temp, ts) e
LATERAL VIEW POSEXPLODE(e.val1) f;
```

Json Transform – External Hive Table Json Format SerDe #5

Target Hasil Transformasi (Baris & Kolom):

receipt_time	site	test_id	val1	temp	TS
2018-09-28T10:00:00.000Z	Los Angeles	C23_PV	0.76	0	1538128801336
2018-09-28T10:00:00.000Z	Los Angeles	C23_PV	0.75	2	1538128810408
2018-09-28T10:00:00.000Z	Los Angeles	C23_PV	0.71	5	1538128818420
2018-09-28T10:00:00.000Z	Los Angeles	HBI2_XX	0.65	1	1538128828433
2018-09-28T10:00:00.000Z	Los Angeles	HBI2_XX	0.71	-7	1538128834541

Terima Kasih