Working with Nested Data

Type your text

```
SELECT * FROM events_strings
```

Spark Jobs

e +

key	value
UA000000107384208	{"device":"macOS","ecommerce":{},"event_name":"checkout","event_previous_timestamp":1593880801027797,"event_timestamp":1593880822506642,"geo":{"city":"Traverse City","state":"MI"},"items":[{"item_id":"M_STAN_T","item_name":"Standard Twin Mattress","item_revenue_in_usd":595.0,"price_in_usd":595.0,"quantity":1}],"traffic_source":"google","user_first_touch_timestamp":1593879413256859,"user_id":"UA000000107384208"}
UA000000107388621	{"device":"Windows","ecommerce":{},"event_name":"email_coupon","event_previous_timestamp":1593880770092554,"event_timestamp":1593880829320848,"geo":{"city":"Hickory","state":"NC"},"items":[{"coupon":"NEWBED10","item_id":"M_STAN_F","item_name":"Standard Full Mattress","item_revenue_in_usd":850.5,"price_in_usd":945.0,"quantity":1}],"traffic_source":"direct","user_first_touch_timestamp":1593879889503719,"user_id":"UA000000107388621"}

NOTE: Spark SQL has built-in functionality to directly interact with nested data stored as JSON strings or struct types.

- Use `:` syntax in queries to access subfields in JSON strings
- Use `.` syntax in queries to access subfields in struct types

value is a JSON string

Cmd 9

```
1  SELECT * FROM events_strings WHERE value:event_name = "finalize" ORDER BY key LIMIT 1
```

nested key in value JSON

Cmd 10

```
1  %python
2  display(events_stringsDF
3      .where("value:event_name = 'finalize'")
4      .orderBy("key")
5      .limit(1)
6  )
```

Above code will fetch only 1 record, in which the value of event_name key will be 'finalize'.

The obtained result set will have the following JSON object as string in the value column:-


```
{
  "device": "Linux",
  "ecommerce": {
    "purchase_revenue_in_usd": 1075.5,
    "total_item_quantity": 1,
    "unique_items": 1
  },
  "event_name": "finalize",
  "event_previous_timestamp": 1593879231210816,
  "event_timestamp": 1593879335779563,
  "geo": {
    "city": "Houston",
    "state": "TX"
  },
  "items": [
    {
      "coupon": "NEWBED10",
      "item_id": "M_STAN_K",
      "item_name": "Standard King Mattress",
      "item_revenue_in_usd": 1075.5,
      "price_in_usd": 1195,
      "quantity": 1
    }
  ],
  "traffic_source": "email",
  "user_first_touch_timestamp": 1593454417513109,
  "user_id": "UA000000106116176"
}
```

Let's use the JSON string example above to derive the schema, then parse the entire JSON column into struct types.

- `schema_of_json()` returns the schema derived from an example JSON string.
- `from_json()` parses a column containing a JSON string into a struct type using the specified schema.

After we unpack the JSON string to a struct type, let's unpack and flatten all struct fields into columns.

- `*` unpacking can be used to flattens structs; `col_name.*` pulls out the subfields of `col_name` into their own columns.

star unpacking

md 13

SQL

```
SELECT schema_of_json('{"device":"Linux","ecommerce":{"purchase_revenue_in_usd":1075.5,"total_item_quantity":1,"unique_items":1},
"event_name":"finalize","event_previous_timestamp":1593879231210816,"event_timestamp":1593879335779563,"geo":{"city":"Houston","state":"TX"},"items":
[{"coupon":"NEWBED10","item_id":"M_STAN_K","item_name":"Standard King Mattress","item_revenue_in_usd":1075.5,"price_in_usd":1195.0,"quantity":1}],
"traffic_source":"email","user_first_touch_timestamp":1593454417513109,"user_id":"UA000000106116176"}') AS schema
```

(1) Spark Jobs

single column named schema

Table +

	schema
1	STRUCT<device: STRING, ecommerce: STRUCT<purchase_revenue_in_usd: DOUBLE, total_item_quantity: BIGINT, unique_items: BIGINT>, event_name: STRING, event_previous_timestamp: BIGINT, event_timestamp: BIGINT, geo: STRUCT<city: STRING, state: STRING>, items: ARRAY<STRUCT<coupon: STRING, item_id: STRING, item_name: STRING, item_revenue_in_usd: DOUBLE, price_in_usd: DOUBLE, quantity: BIGINT>>, traffic_source: STRING, user_first_touch_timestamp: BIGINT, user_id: STRING>

Schema

This column is further operated by from_json()

This column actually contains the schema to which the JSON string fits in

json value containing column

```
-- My seggregation for understanding
SELECT from_json(value, 'STRUCT<device: STRING, ecommerce: STRUCT<purchase_revenue_in_usd: DOUBLE, total_item_quantity: BIGINT, unique_items:
BIGINT>, event_name: STRING, event_previous_timestamp: BIGINT, event_timestamp: BIGINT, geo: STRUCT<city: STRING, state: STRING>, items:
ARRAY<STRUCT<coupon: STRING, item_id: STRING, item_name: STRING, item_revenue_in_usd: DOUBLE, price_in_usd: DOUBLE, quantity: BIGINT>>,
traffic_source: STRING, user_first_touch_timestamp: BIGINT, user_id: STRING>') AS json
FROM events_strings
```

this is the schema, that we obtained in the previous step
It is the 2nd parameter of the from_json() function

ark Jobs

+

json
{"device": "macOS", "ecommerce": {"purchase_revenue_in_usd": null, "total_item_quantity": null, "unique_items": null}, "event_name": "checkout", "event_previous_timestamp": 1593880801027797, "event_timestamp": 1593880822506642, "geo": {"city": "Traverse City", "state": "MI"}, "items": [{"coupon": null, "item_id": "M_STAN_T", "item_name": "Standard Twin Mattress", "item_revenue_in_usd": 595, "price_in_usd": 595, "quantity": 1}], "traffic_source": "google", "user_first_touch_timestamp": 1593879413256859, "user_id": "UA000000107384208"}
{"device": "Windows", "ecommerce": {"purchase_revenue_in_usd": null, "total_item_quantity": null, "unique_items": null}, "event_name": "email_coupon", "event_previous_timestamp": 1593880770092554, "event_timestamp": 1593880829320848, "geo": {"city": "Hickory", "state": "NC"}, "items": [{"coupon": "NEWBED10", "item_id": "M_STAN_F", "item_name": "Standard Full Mattress", "item_revenue_in_usd": 850.5, "price_in_usd": 945, "quantity": 1}], "traffic_source": "direct", "user_first_touch_timestamp": 1593879889503719, "user_id": "UA000000107388621"}

⇒ schema_of_json()

Creates schema string from JSON string column.

⇒ from_json(<json string>, <schema>)

column having json document as a string

schema string obtained from schema_of_json()

Returns a struct value (parsed json), with the <json_string> passed to it, applying the <schema> in the 2nd parameter.

⇒ struct type value obtained from from_json() function, can be converted to a tabular format, using star unpacking :-

⇒ Unpacking Struct type object fields into columns :-

```
CREATE OR REPLACE TEMP VIEW parsed_events AS
SELECT json.* FROM (
  SELECT from_json(value, 'STRUCT<device: STRING, ecommerce: STRUCT<purchase_revenue_in_usd: DOUBLE, total_item_quantity: BIGINT, unique_items:
  BIGINT>, event_name: STRING, event_previous_timestamp: BIGINT, event_timestamp: BIGINT, geo: STRUCT<city: STRING, state: STRING>, items:
  ARRAY<STRUCT<coupon: STRING, item_id: STRING, item_name: STRING, item_revenue_in_usd: DOUBLE, price_in_usd: DOUBLE, quantity: BIGINT>>,
  traffic_source: STRING, user_first_touch_timestamp: BIGINT, user_id: STRING>'
  ) AS json
  FROM events_strings
);

SELECT * FROM parsed_events
```

obtained from from_json()

ark Jobs

device	ecommerce	event_name	event_previous_timestamp	event_timestamp	geo
macOS	{"purchase_revenue_in_usd": null, "total_item_quantity": null, "unique_items": null}	checkout	1593880801027797	1593880822506642	{"city": "Traverse
Windows	{"purchase_revenue_in_usd": null, "total_item_quantity": null, "unique_items": null}	email_coupon	1593880770092554	1593880829320848	{"city": "Hickory"
Windows	{"purchase_revenue_in_usd": null, "total_item_quantity": null, "unique_items": null}	main	null	1593880824305898	{"city": "Fargo", "
Android	{"purchase_revenue_in_usd": null, "total_item_quantity": null, "unique_items": null}	add_item	1593880753875794	1593880826675403	{"city": "Chicago"
macOS	{"purchase_revenue_in_usd": null, "total_item_quantity": null, "unique_items": null}	checkout	1593880741311315	1593880830140019	{"city": "Fargo", "
Android	{"purchase_revenue_in_usd": 1195, "total_item_quantity": 1, "unique_items": 1}	finalize	1593876666372094	1593880830038130	{"city": "Tuttle", "

- * unpacking unpacks structs [Ex:- json.*]
- col_name.* pulls out subfields of col_name into their own columns.

Manipulating Arrays :-

- explode() seperates elements of an array (in a column) into multiple rows, and this creates a new row for each element.
- size() provides count for the number of elements in an array for each row (that can be exploded).

Ex:-

this column values are arrays

```
SELECT *, explode(items) FROM itemsTbl;
```

```
%python
itemsTblDF.withColumn("item", explode("items"))
```

- collect_set():

Collects unique values for a field, including fields within arrays.

2) flatten():

Combines multiple arrays into a single array.

3) array_distinct():

Remove duplicate elements from an array.