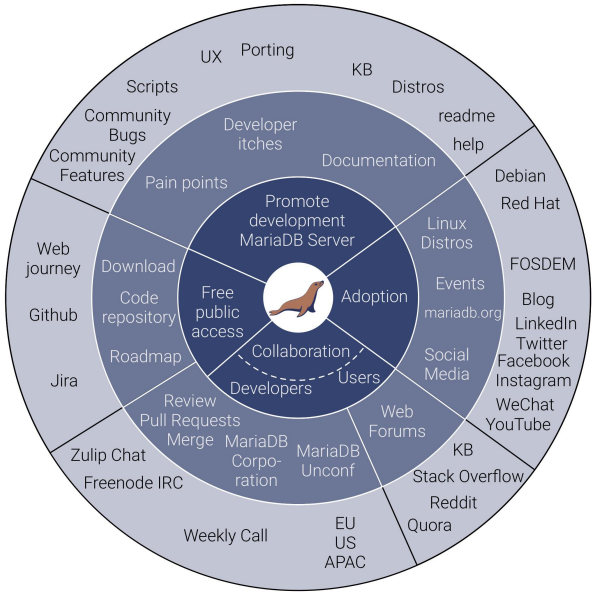


MariaDB Foundation

JSON additions in MariaDB featuring JSON_TABLE

Vicențiu Ciorbaru
Software Developer Team Lead
@ MariaDB Foundation

Percona Live
2021 May - MariaDB Community Room

whoami

- Vicențiu Ciorbaru
- MariaDB Foundation,
  Software Developer Team Lead
- MariaDB developer since 2013-...
- Implemented Roles, Window Functions and others

JSON Support in earlier versions

- MariaDB added support for JSON functions in 10.2
- Support has expanded with each release
- Supports JSON functionality according to SQL standard
- JSON is stored as text, easy to work with
- With the help of virtual columns, data can be normalized (and now JSON_TABLE too)

JSON Path

- Functionally defined in SQL Standard 2016
- MariaDB implements a subset

```
JSON Path :: [mode] $ [expression]

mode :: strict | lax

expression :: <member-selector> | <array-selector>
```

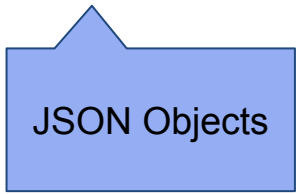
JSON Path

- Functionally defined in SQL Standard 2016
- MariaDB implements a subset

```
JSON Path :: [mode] $ [expression]

mode :: strict | lax

expression :: <member-selector> | <array-selector>
```



JSON Path

- Functionally defined in SQL Standard 2016
- MariaDB implements a subset

```
JSON Path :: [mode] $ [expression]

mode :: strict | lax

expression :: <member-selector> | <array-selector>
```

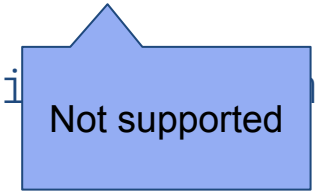
JSON Path

- Functionally defined in SQL Standard 2016
- MariaDB implements a subset

```
JSON Path :: [mode] $ [expression]

mode :: strict | lax

expressi                    nber-selector> | <array-selector>
```



JSON Path

- Multiple ways to access objects' members

```
{
  "employer" : "MariaDB",
  "title"    : "Software Developer"
}
```

- `$.<member-name>`
  - `$.employer -> "MariaDB"`
  - `$.title -> "Software Developer"`
- `$.*`
  - Array of all values
    `["MariaDB", "Software Developer"]`

There is a MariaDB Extension (also in MySQL)

- `$**.<member-name>`
  - Return all the values of <member-name>, wherever it is in the document.

JSON Path

- Array selection

  - Index based: `[N]`
    - `$[0] -> "small"`
    - `$[3] -> {"name":"John"}`

  - All elements in an array `[*]`
    - Array of all values
      `["small", 100, "big", {"name":"John"}]`

```
[
 "small", 100, "big", {"name":"John"}
]
```

- Other syntax not supported yet

  - Last element `[last]`
    - `$[last] -> {"name":"John"}`
  - Range `[N to M]`
    - `[0 to 2] -> ["small", 100, "big"]`
  - List `[N1,N2,N3]`

JSON_TABLE syntax

- `JSON_TABLE(<json_document>,`
  `<json-sequence-selection> columns(<data-extraction>))`

JSON_TABLE syntax

- JSON_TABLE(**<json_document>**,
  <json_sequence-selection> columns(<data-extraction>))

Must be valid JSON

JSON_TABLE syntax

- `JSON_TABLE(<json_document>,`
  `**<json-sequence-selection>** columns(<data-extraction>))`

Must be a valid JSON Path

JSON_TABLE syntax

- JSON_TABLE(<json_document>,
  <json-sequence-selection> columns(<data-extraction>))

- **data-extraction**
  column-name type **path** <json-path-selector>

JSON_TABLE syntax

- JSON_TABLE(<json_document>,
  <json-sequence-selection> columns(<data-extraction>))

- **data-extraction**
  column-name type **path** <json-path-selector>

- ```
  select *
  from json_table('["a", "b", "c"]',
                  '$[*]' columns (id varchar(20) path '$')) as jt;
  ```
  ```

JSON_TABLE syntax

- JSON_TABLE(**<json_document>**,
  <json-sequence-selection> columns(<data-extraction>))

- **data-extraction**
  column-name type **path** <json-path-selector>

- select *
  from json_table('["a", "b", "c"]',
  '$[*]' columns (id varchar(20) path '$')) as jt;

JSON_TABLE syntax

- JSON_TABLE(<json_document>,
  <json-sequence-selection> columns(<data-extraction>))

- **data-extraction**
  column-name type **path** <json-path-selector>

- ```
  select *
  from json_table('["a", "b", "c"]',
                  '$[*]' columns (id varchar(20) path '$')) as jt;
  ```
  ```

JSON_TABLE syntax

- JSON_TABLE(<json_document>,
  <json-sequence-selection> **columns(<data-extraction>)**)

- **data-extraction**
  column-name type **path** <json-path-selector>

- ```
  select *
  from json_table('["a", "b", "c"]',
                  '$[*]' columns (id varchar(20) path '$')) as jt;
  ```


  ```

JSON_TABLE syntax

- JSON_TABLE(<json_document>,
  <json-sequence-selection> columns(<data-extraction>))

- **data-extraction**
  column-name type **path** <json-path-selector>

- ```
  select *
  from json_table('["a", "b", "c"]',
                  '$[*]' columns (id varchar(20) path '$')) as jt;
  ```

```
+------+
| id   |
+------+
| a    |
| b    |
| c    |
+------+
```
```

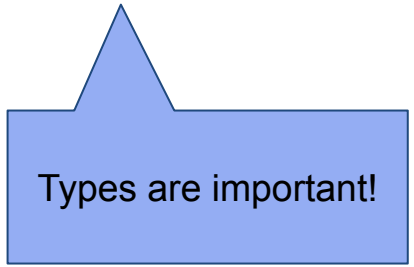
JSON_TABLE syntax

- JSON_TABLE(<json_document>,
  <json-sequence-selection> columns(<data-extraction>))

- **data-extraction**
  column-name type **path** <json-path-selector>

- ```
  select *
  from json_table('["a", "b", "c"]',
                  '$[*]' columns (id varchar(20) path '$')) as jt;
  ```

```
+------+
| id   |
+------+
| a    |
| b    |
| c    |
+------+
```


```

JSON_TABLE syntax

- JSON_TABLE(<json_document>,
  <json-sequence-selection> columns(<data-extraction>))

- **data-extraction**
  column-name type **path** <json-path-selector>

- ```
  select *
  from json_table('["aaa", "bbb", "ccc"]',
                  '$[*]' columns (id varchar(2) path '$')) as jt;
  ```

```
+------+
| id   |
+------+
| aa   |
| bb   |
| cc   |
+------+
```
```

Nested paths

- ```
  select * from json_table(
    '[
       { "brand": "Tesla", "models": ["S", "3", "X", "Y"] },
       { "brand": "Mercedes", "models": ["EQA", "EQC", "EQS"] }
     ]',
    '$[*]' columns (brand varchar(10) path '$.brand',
                    nested path '$.models[*]'
                      columns (model varchar(20) path '$'))) as jt;
  ```

```
+----------+-------+
| brand    | model |
+----------+-------+
| Tesla    | S     |
| Tesla    | 3     |
| Tesla    | X     |
| Tesla    | Y     |
| Mercedes | EQA   |
| Mercedes | EQC   |
| Mercedes | EQS   |
+----------+-------+
```
```

Nested paths

- ```
  select * from json_table(
    '[
       { "brand": "Tesla", "models": ["S", "3", "X", "Y"] },
       { "brand": "Mercedes", "models": ["EQA", "EQC", "EQS"] }
     ]',
    '$[*]' columns (brand varchar(10) path '$.brand',
                    nested path '$.models[*]'
                      columns (model varchar(20) path '$'))) as jt;
  ```

```
+----------+-------+
| brand    | model |
+----------+-------+
| Tesla    | S     |
| Tesla    | 3     |
| Tesla    | X     |
| Tesla    | Y     |
| Mercedes | EQA   |
| Mercedes | EQC   |
| Mercedes | EQS   |
+----------+-------+
```

Can have nested, nested paths.
```

Sibling Nested paths

- ```
  select * from json_table(
    '[
      { "brand": "Tesla", "models": ["S", "3", "X", "Y"], "colors":["red", "blue"] },
    ]',
    '$[*]' columns (brand varchar(10) path '$.brand',
                    nested path '$.models[*]'
                      columns (model varchar(20) path '$'),
                    nested path '$.colors[*]'
                      columns (color varchar(20) path '$'))) as jt;
  ```

```
+----------+-------+-------+
| brand    | model | color |
+----------+-------+-------+
| Tesla    | S     | NULL  |
| Tesla    | 3     | NULL  |
| Tesla    | X     | NULL  |
| Tesla    | Y     | NULL  |
| Tesla    | NULL  | red   |
| Tesla    | NULL  | blue  |
+----------+-------+-------+
```
```

Sibling Nested paths

```
select * from json_table(
  '[
    { "brand": "Tesla", "models": ["S", "3", "X", "Y"], "colors":["red", "blue"] },
  ]',
  '$[*]' columns (brand varchar(10) path '$.brand',
                  nested path '$.models[*]'
                    columns (model varchar(20) path '$'),
                  nested path '$.colors[*]'
                    columns (color varchar(20) path '$'))) as jt;
```

Not possible to get "cross-join" of nested paths, no **PLAN** clause

```
+----------+-------+-------+
| brand    | model | color |
+----------+-------+-------+
| Tesla    | S     | NULL  |
| Tesla    | 3     | NULL  |
| Tesla    | X     | NULL  |
| Tesla    | Y     | NULL  |
| Tesla    | NULL  | red   |
| Tesla    | NULL  | blue  |
+----------+-------+-------+
```

Handling errors

```
columns (column_name path '$path' [action on empty]
                                  [action on error])
```

Actions are:

- NULL
- EMPTY
- ERROR
- <default value>

Handling errors

```
columns (column_name path '$path' [action on empty]
                                  [action on error])
```

Actions are:

- NULL
- EMPTY
- ERROR
- <default value>

```
ERROR 4176 (HY000): Field 'color' can't be set for JSON_TABLE 'jt'.
```

Requires more specific error messages

JSON Table and LATERAL JOINS

ID	json_doc
1	[{"product": "Smartphone", "price": 300}, {"product": "Power Bank", "price": 50}]
2	[{"product": "Jacket", "price": 150}, {"product": "Tie", "price": 25}]

```
select
  orders.id,
  order_items.product,
  order_items.price
from
  orders,
  JSON_TABLE(orders.json_doc,
             '$[*]' columns (product varchar(32) path '$.product',
                             price   int         path '$.price')
            ) as order_items;
```

JSON Table and LATERAL JOINS

ID	json_doc
1	[{"product": "Smartphone", "price": 300}, {"product": "Power Bank", "price": 50}]
2	[{"product": "Jacket", "price": 150}, {"product": "Tie", "price": 25}]

```
select
  orders.id,
  order_items.product,
  order_items.price
from
  orders,
  JSON_TABLE(orders.json_doc,
             '$[*]' columns (product varchar(32) path '$.product',
                             price   int         path '$.price')
           ) as order_items;
```

JSON Table and LATERAL JOINS

ID	json_doc
1	[{"product": "Smartphone", "price": 300}, {"product": "Power Bank", "price": 50}]
2	[{"product": "Jacket", "price": 150}, {"product": "Tie", "price": 25}]

```
+------+------------+-------+
| id   | product    | price |
+------+------------+-------+
|    1 | Smartphone |   300 |
|    1 | Power Bank |    50 |
|    2 | Jacket     |   150 |
|    2 | Tie        |    25 |
+------+------------+-------+
```

```
select
  orders.id,
  order_items.product,
  order_items.price
from
  orders,
  JSON_TABLE(orders.json_doc,
             '$[*]' columns (product varchar(32) path '$.product',
                             price   int         path '$.price')
            ) as order_items;
```

JSON Table internal details

- Current implementation does a single pass through the JSON document
- Custom JSON parser
- The table generation is done using constant memory.
    - Some operations in the future (such as list of array indices) might require additional memory.
- Behind the scenes, it uses the HEAP storage engine, creating a temporary table for each JSON document.
- This allows regular block nested loop (BNL) join optimizations.

Overall

- MariaDB supports a subset of SQL Standard specification for JSON_TABLE
  Available in MariaDB 10.6.0

- Compared to other databases, a very competitive subset
  - MySQL has a few more additions (last, M to N, last - N)

- It is rather difficult to debug JSON Path errors
  - Error messages could be improved

- Provide useful LATERAL DERIVED join semantics
  - JSON_TABLE values can depend on previous tables in the same FROM clause.

Thank you!

Contact details:

vicentiu@mariadb.org

About:

https://mariadb.org/vicentiu