MariaDB®
FOUNDATION

JSON Support in MariaDB

Vicențiu Ciorbaru
Software Engineer @ MariaDB Foundation

JSON & GeoJSON

- JSON (Java Script Object Notation), a text based, platform independent data exchange format
- MariaDB Supports:
  - JSON Functions (24 currently)
  - Converting Geometrical data types to JSON
  - Storing JSON in String based data types like VARCHAR

Why JSON?

- JSON is a very-simple key-value format.
- For a particular row we can store an arbitrary number of keys.
- Combine MariaDB's relational capabilities with JSON and you get the best of both traditional SQL and NoSQL worlds.

How is JSON Useful?

- We are designing an online store.
- We have a simple products table.

Products			
ID	Name	Price	Stock
1	Jeans	100	2
2	TShirt	30	10

How is JSON Useful?

- We are designing an online store.
- We have a simple products table.

Products			
ID	Name	Price	Stock
1	Jeans	100	2
2	TShirt	30	10

What if we want to store more details about each product?

How is JSON Useful?

- Solution #1: Create a "Details" table and have FKs pointing to Products.

How is JSON Useful?

- Solution #1: Create a "Details" table and have FKs pointing to Products.

Products			
ID	Name	Price	Stock
1	Jeans	100	2
2	TShirt	30	10

Details			
ID	Name	Value	ProdRef
1	Size	M	1
2	Color	White	2
3	Logo	Seal	2

How is JSON Useful?

- Solution #1: Create a "Details" table and have FKs pointing to Products.

Products			
ID	Name	Price	Stock
1	Jeans	100	2
2	TShirt	30	10

Details			
ID	Name	Value	ProdRef
1	Size	M	1
2	Color	White	2
3	Logo	Seal	2

What if "Logo" has multiple details? Ex: With/Without text, Color Green etc.

How is JSON Useful?

- Expressing collections using just SQL & Tables is difficult.

Products			
ID	Name	Price	Stock
1	Jeans	100	2
2	TShirt	30	10

Details			
ID	Name	Value	ProdRef
1	Size	M	1
2	Color	White	2
3	Logo	Seal	2

What if "Logo" has multiple details? Ex: With/Without text, Color Green etc.

How is JSON Useful?

- Expressing collections using just SQL & Tables is difficult.

Products

ID			
1			
2	TShirt	30	10

What if TShirt has a different color than the Logo?

What if "Logo" has multiple details? Ex: With/Without text, Color Green etc.

Details

ID	Name	Value	ProdRef
1	Size	M	1
2	Color	White	2
3	Logo	Seal	2

How is JSON Useful?

- JSON allows a much simpler approach!

Products

ID			
1			
2	TShirt	30	10

What if TShirt has a different color than the Logo?

What if "Logo" has multiple details? Ex: With/Without text, Color Green etc.

Details

ID	Name	Value	ProdRef
1	Size	M	1
2	Color	White	2
3	Logo	Seal	2

How is JSON Useful?

- We extend the Products table with an "Attribute" column.

Products				
ID	Name	Price	Stock	Attribute
1	Jeans	100	2	`{ "size" : "M", "color" : "red" }`
2	TShirt	30	10	`{ "logo" : { "type" : "seal", "color" : "white" }, "color" : "blue" }`

How is JSON Useful?

- All that JSON is stored as a regular string. How does MariaDB help?
- MariaDB (starting with 10.2) actually enforces CHECK constraints
  - Ensure JSON is valid on INSERT / UPDATE.
- Ideally you want some sort of index on the JSON for fast access.
- Regular Full-Text Index does not (usually) help.
  - Virtual Columns however can be!

Enforcing Check Constraints

```
CREATE TABLE products(id INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT,
                      name VARCHAR(255) NOT NULL,
                      price DECIMAL(9,2) NOT NULL,
                      stock INTEGER NOT NULL,
                      attribute VARCHAR(1024),
             CHECK (JSON_VALID(attribute)));
```

- All values on attribute will be correctly formatted JSON.

```
INSERT INTO products VALUES(NULL, 'Jeans', 10.5, 165, NULL);
ERROR 4025 (23000): CONSTRAINT `CONSTRAINT_1` failed for
`inventory`.`products`

INSERT INTO products VALUES(NULL, 'Blouse', 17, 15, '{"colour": "white}');
ERROR 4025 (23000): CONSTRAINT `CONSTRAINT_1` failed for
`inventory`.`products`
```

Enforcing Check Constraints

```
CREATE TABLE products(id INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT,
                      name VARCHAR(255) NOT NULL,
                      price DECIMAL(9,2) NOT NULL,
                      stock INTEGER NOT NULL,
                      attribute VARCHAR(1024),
             CHECK (JSON_VALID(attribute)));
```

- All values on attribute will be correctly formatted JSON.

```
INSERT INTO products VALUES(NULL, 'Blouse', 17, 15, '{"color": "white"}');
Query OK, 1 row affected (0.01 sec)
```

JSON & Performance

- We can index JSON columns using a regular text index.
- But we usually want indices on specific JSON Keys.

```
ALTER TABLE products ADD attr_color VARCHAR(32) AS (
                                              JSON_VALUE(attr, '$.color'));
CREATE INDEX products_attr_colour_ix ON products(attr_colour);

SELECT * FROM products WHERE attr_colour = 'red' or attr_color = 'white';
+----+--------+-------+-------+---------------------------------+-------------+
| id | name   | price | stock | attribute                       | attr_color  |
+----+--------+-------+-------+---------------------------------+-------------+
|  1 | Jeans  |   100 |     2 | { "size" : "M","color":"red"  } | red         |
|  2 | TShirt |    30 |    10 | { "color":"white", "logo":{..}} | white       |
+----+--------+-------+-------+---------------------------------+-------------+
```

Updating JSON

- We can update data using virtual columns.

```
UPDATE products SET attr = JSON_REPLACE(attr, '$.colour', 'red') WHERE name =
'TShirt';

SELECT * FROM products WHERE attr_colour = 'red';
+----+--------+-------+-------+----------------------------------+-------------+
| id | name   | price | stock | attribute                        | attr_color  |
+----+--------+-------+-------+----------------------------------+-------------+
|  1 | Jeans  |   100 |     2 | { "size" : "M","color":"red"  } | red         |
|  2 | TShirt |    30 |    10 | { "color":"white", "logo":{..}} | white       |
+----+--------+-------+-------+----------------------------------+-------------+
```

Things to do / improve

- Better indexes, without resorting to virtual columns.
- JSON datatype?
- Code can become a bit verbose. Simplify?

```
db.restaurants.find( { "address.zipcode": "10075" } )
```

Thank You!

Contact me at:
vicentiu@mariadb.org
vicentiu@ciorbaru.io

Blogs:
mariadb.org/blog
vicentiu.ciorbaru.io