JSON in SQL

What is JSON?

The full name of JSON is JavaScript Object Notation, which is known for its simplicity, high flexibility, and high compatibility. To learn more about JSON, please refer to MDN documentation.

The JSON has three prominent characteristics: 1. Wrap the data in curly brackets {}. 2. The text in double quotation marks can be used as the column name like "{"place": "U.S"}", followed by the value of the column. 3. The square brackets [] can wrap multiple, similar data.

JSON Data Sample

```
{
    "orderId": 54321,
    "info": [
        {
            "productID": 34,
            "productName": "productOne",
            "quantity": 1
        },
        {
            "productID": 56,
            "productName": "productTwo",
            "quantity": 3
        },
    ],
    "orderCompleted": true
}
```

We can observe that the above JSON describes an order with an `order_id` of 54321; its `info` includes two products, one whose `id` is 34 and the other whose `id` is 56.

In the modern age, where JSON is becoming more and more common, databases provide SQL functionalities that specifically handle the JSON format. There are even databases specially designed to handle data structures like JSON.

However, the syntax for using JSON differs across database systems, and the difference is more significant than the difference for basic syntax such as `SELECT` and `UPDATE`.

The mainstream database systems, MySQL, Oracle SQL, and MsSQL, have similar usage methods, while PostgreSQL is more special. Here, we will use MySQL style for each example.

Sample data

First of all, the data type of the column can be set to "JSON", which is convenient to use JSON-related syntax to retrieve data. It is written like:

```
ALTER TABLE `new_schema`.`users`
ADD COLUMN `contact` JSON NULL AFTER `id`;
```

The search results for this chapter will refer to this sample dataset:

id	name	contact
1	John	{"phone": "123-456", "address": "New York"}
2	May	{"phone": "888-99", "address": "LA"}
3	Tim	{"phone": "1236"}
4	Jay	{"phone": "321-6", "address": "Boston"}

To help solidify the new concepts, you can follow along by editing the provided table below:

SCHEMA SQL

```
CREATE SCHEMA `new_schema` DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;


CREATE TABLE `new_schema`.`users` (
  `id` INT NOT NULL AUTO_INCREMENT COMMENT 'This is the primary index',
  `name` VARCHAR(45) NOT NULL DEFAULT 'N/A',
  PRIMARY KEY (`id`)
);


ALTER TABLE `new_schema`.`users` ADD COLUMN `contact` JSON NULL AFTER `id`;


INSERT INTO `new_schema`.`users` (`id`, `name`, `contact`) VALUES
  (1, 'John', JSON_OBJECT('phone', '123-456', 'address', 'New York')),
  (2, 'May', JSON_OBJECT('phone', '888-99', 'address', 'LA')),
  (3, 'Tim', JSON_OBJECT('phone', '1236')),
  (4, 'Jay', JSON_OBJECT('phone', '321-6', 'address', 'Boston'));
```

QUERY SQL

```
SELECT `id`, `name`, `contact` FROM `new_schema`.`users`;
```

RESULT

Query #1 **Execution time: 0ms**

id	name	contact
1	John	{"phone": "123-456", "address": "New York"}
2	May	{"phone": "888-99", "address": "LA"}
3	Tim	{"phone": "1236"}
4	Jay	{"phone": "321-6", "address": "Boston"}

Read Data

The first thing we need to learn is how to use `SELECT` to display the JSON columns we want to view. As can be seen from the sample data, if the structure of the JSON data is more complex, the readability may be significantly reduced. If we only want to see the `phone` column data, we can use the following query:

```
SELECT `id`, JSON_EXTRACT(contact, '$.phone') AS phone
FROM `new_schema`.`users`;
```

The result is:

id	phone
1	"123-456"
2	"888-99"
3	"1236"
4	"321-6"

Let's divide this into two parts for analysis: 1. `JSON_EXTRACT`: This keyword needs to be combined with a pair of parentheses, and the words written inside are separated by commas. The first word is the outer column name, the second is the JSON column name (which is prefixed with a `$.` symbol). 2. `AS`: This is actually an additional keyword because it is not exclusively used for JSON. `AS` can be used to rename the selected column as `phone`. If we did not include `AS`, then the column name in the result would be `JSON_EXTRACT(contact, '$.phone')`.

However, you will notice that there is a flaw in the data here, that is, each data has a pair of double quotes, so I recommend using `JSON_UNQUOTE` to remove the flaw:

```
SELECT `id`, JSON_UNQUOTE(JSON_EXTRACT(contact, '$.phone')) AS phone
FROM `new_schema`.`users`;
```

Result:

id	phone
1	123-456
2	888-99
3	1236
4	321-6

After solving the data format problem, we will have another question: how to find the qualified data by the JSON column. For example, to find phone numbers that contain 456, we could write the following query:

```
SELECT `id`, JSON_UNQUOTE(JSON_EXTRACT(contact, '$.phone')) AS phone
FROM `new_schema`.`users`
WHERE JSON_EXTRACT(contact, '$.phone') like '%456%';
```

Result:

id	phone
1	123-456

We can see the `JSON_EXTRACT` can not only be used on `SELECT` but also can be used on `WHERE` for filtering data.

Add Data

If you want to add data in the column with JSON format, you do not need to write the JSON data structure in a string format such as `{"phone": "123123"}`. Instead, you can use `JSON_OBJECT` to make it more accessible and readable by wrapping the JSON data that needs to be stored. For example:

```
INSERT INTO `new_schema`.`users` (`id`, `name`, `contact`) VALUES (5,
'Harry', JSON_OBJECT('phone', '1231123', 'address', 'Miami'));
```

Result:

id	name	contact
1	John	{"phone": "123-456", "address": "New York"}
2	May	{"phone": "888-99", "address": "LA"}
3	Tim	{"phone": "1236"}
4	Jay	{"phone": "321-6", "address": "Boston"}
5	Harry	{"phone": "1231123", "address": "Miami"}

Pass the column names and values in the string format and separate them with commas. The first two are a pair: first is the JSON column name, and the second is the corresponding value.

Updates

To update a value, if MySQL did not provide JSON-specific operation syntax, it would be particularly troublesome to write, but through the `JSON_SET` keyword, it looks very pleasing to our brains:

```
UPDATE `new_schema`.`users` SET `contact` = JSON_SET(contact, '$.phone', '6666', '$.phone_2', '888') WHERE `id` = 2;
```

Result:

id	name	contact
1	John	{"phone": "123-456", "address": "New York"}
2	May	{"phone": "6666", "address": "LA", "phone_2": "888"}
3	Tim	{"phone": "1236"}
4	Jay	{"phone": "321-6", "address": "Boston"}

5	Harry	{"phone": "1231123", "address": "Miami"}

Notice that if the column set to `JSON_SET` is not in the original JSON column name, a new pair will be generated directly, which is a very convenient setting. Likewise, since we did not include "address" while updating the JSON, the previous address was retained.