MySQL JSON Data Type

Summary: in this tutorial, you will learn how to use MySQL JSON data type to store JSON documents in the database.

What is JSON

JSON stands for "JavaScript Object Notation". JSON is a lightweight data-interchange format that is easy for humans to read and write and simple for computers to parse and generate.

JSON is built on two data structures: **objects** and **arrays**.

Objects

An object is an unordered collection of key-value pairs enclosed in curly braces `{}`. Each key is a string, followed by a colon (`:`), and then the associated value. For example:

```
{
    "name": "John Doe",
    "age": 22
}
```

> Note that you must enclose the key in a JSON document with double quotes (`"`).

Arrays

An array is an ordered list of values closed in square brackets `[]`. An array may contain values of any valid JSON data type, including objects and other arrays. For example:

```
["John Doe", 22]
```

In this example, we have a JSON array of two values that represent the name and age of a person.

JSON supports several data types, including:

- String: “JSON”
- Number: 10, 12.25
- Boolean: true and false.
- Null: null

In practice, you use JSON in web development for:

- Configuration files
- Data exchange between a client and a server

Introduction to MySQL JSON data type

MySQL supports the native JSON data type defined by RFC 7159 starting in version 5.7.8. The native JSON data type offers the following advantages:

- MySQL validates the JSON documents stored in the JSON column and issues an error if they are invalid.
- MySQL stores the JSON documents in a binary format optimized for quick searches.

MySQL uses roughly the same storage for JSON documents as for LONGBLOG or LONGTEXT data.

The following shows how to define a table column with the JSON data type:

```
column_name JSON
```

The JSON column can store NULL values. Also, starting from MySQL 8.0.13, a JSON column can accept a default value.

It is not possible to directly index a JSON column. Instead, you can use a functional index to create an index on values extracted from the JSON column.

When querying data from a JSON column, the MySQL optimizer will search for functional indexes to match the JSON expressions.

MySQL JSON data type example

We'll take an example of using the JSON data type.

1) Creating a table with a JSON column

The following statement creates a new table called `products` with a JSON column:

```
CREATE TABLE products(
   id INT AUTO_INCREMENT PRIMARY KEY,
   name VARCHAR(255) NOT NULL,
   price DECIMAL(10,2) NOT NULL,
   properties JSON
);
```

The `products` table has four columns:

- `id` : This is an auto-increment primary key column that uniquely identifies a product.
- `name` : This column stores the name of the product.
- `price` : This column stores the price of the product.
- `properties` : This column stores the properties of the product such as colors and sizes.

2) Inserting JSON data into the table

The following statement inserts a new row with JSON data into the `products` table:

```
INSERT INTO products(name, price, properties)
```

```
VALUES('T-Shirt', 25.99, '{"sizes":["S","M","L","XL"], "colors": ["white","black"]}');
```

In this example, we inserted a new row into the `products` table. The value of the `properties` column is a JSON document with the following format:

```
{"sizes":["S","M","L","XL"],"colors":["white","black"]}
```

And we place the JSON document inside a single quote (').

When the statement executes, MySQL performs the following steps:

- First, validate the JSON document for validity.
- Second, convert the JSON string into binary format and store it in the JSON column.

3) Querying JSON data from the table

The following query retrieves data from the `products` table:

```
SELECT name, properties
FROM products;
```

Output:

```
+---------+-------------------------------------------------------------------+
| name    | properties                                                        |
+---------+-------------------------------------------------------------------+
| T-Shirt | {"sizes": ["S", "M", "L", "XL"], "colors": ["white", "black"]} |
+---------+-------------------------------------------------------------------+
1 row in set (0.00 sec)
```

If the output is difficult to read, you can use the `JSON_PRETTY()` function to format it.

```
SELECT JSON_PRETTY(properties)
FROM products;
```

Output:

```
+-----------------------------------------------------------------------------------------
| JSON_PRETTY(properties)
+-----------------------------------------------------------------------------------------
| {
  "sizes": [
    "S",
    "M",
    "L",
    "XL"
  ],
  "colors": [
    "white",
    "black"
  ]
} |
+-----------------------------------------------------------------------------------------
1 row in set (0.00 sec)
```

4) Getting the keys of a JSON document

The following uses the `JSON_KEYS()` function to list all the keys of the JSON document:

```
SELECT JSON_KEYS(properties)
FROM products;
```

Output:

```
+-----------------------+
| JSON_KEYS(properties) |
+-----------------------+
| ["sizes", "colors"]   |
+-----------------------+
1 row in set (0.01 sec)
```

In this example, we use the `JSON_KEYS()` function to get all the keys of the JSON document stored in the `properties` column.

The function returns a JSON array containing all the keys of the JSON document.

5) Extracting data from a JSON document

To specify a location within a JSON document, you use a JSON path expression. A path expression allows you to navigate through the structure of a JSON document to access specific data.

- Use the dollar sign (`$`) to represent the current document.
- Use a key or array index to specify the exact location.

For example, to get the first colors of the product, you use the following path:

```
$.colors[0]
```

To extract the first color from the JSON document with the above path, you use the `JSON_EXTRACT()` function. For example:

```
SELECT JSON_EXTRACT(properties, "$.colors[0]")
FROM products;
```

Output:

```
+----------------------------------------+
| JSON_EXTRACT(properties, "$.colors[0]") |
+----------------------------------------+
| "white"                                |
+----------------------------------------+
1 row in set (0.00 sec)
```

Summary

- MySQL supports native JSON data types starting in version 5.7.8.
- MySQL stores JSON in a binary format optimized for quick searches.
- Use the `JSON_PRETTY()` function to format the JSON documents.
- Use the `JSON_KEY()` function to get all keys of a JSON document.
- Use the `JSON_EXTRACT()` function to extract data from a JSON document.

Was this tutorial helpful?

PREVIOUSLY
MySQL JSON

UP NEXT
MySQL JSON_ARRAY() Function

Search ...

GETTING STARTED

What Is MySQL?

Install MySQL Database Server

Connect to MySQL Server

Download MySQL Sample Database

Load Sample Database

QUERYING DATA

SELECT FROM

SELECT

ORDER BY

WHERE

SELECT DISTINCT

AND

OR

IN

NOT IN

BETWEEN

LIKE

LIMIT

IS NULL

Table & Column Aliases

Joins

INNER JOIN

LEFT JOIN

RIGHT JOIN

Self Join

CROSS JOIN

GROUP BY

HAVING

HAVING COUNT

ROLLUP

Subquery

Derived Tables

EXISTS

EXCEPT

INTERSECT

MANAGING DATABASES

Select a Database

Create Databases

Drop Databases

MANAGING TABLES

Create Tables

AUTO_INCREMENT

Rename Tables

Add Columns

Drop Columns

Drop Tables

Temporary Tables

Generated Columns

MYSQL CONSTRAINTS

Primary Key

Foreign Key

Disable Foreign Key Checks

UNIQUE Constraint

NOT NULL Constraint

DEFAULT Constraint

CHECK Constraint

INSERT DATA

Insert Into

Insert Multiple Rows

INSERT INTO SELECT

Insert On Duplicate Key Update

INSERT IGNORE

Insert DateTimes

Insert Dates

UPDATE DATA

UPDATE

UPDATE JOIN

DELETE DATA

DELETE JOIN

ON DELETE CASCADE

TRUNCATE TABLE

MYSQL TRANSACTIONS

Table Locking

MYSQL DATA TYPES

BIT

INT

BOOLEAN

DECIMAL

DATETIME

TIMESTAMP

DATE

TIME

CHAR

VARCHAR

TEXT

BINARY

VARBINARY

ENUM

BLOB

MYSQL GLOBALIZATION

MySQL Character Sets

MySQL Collation

MYSQL IMPORT & EXPORT

Import a CSV File Into a Table

Export a Table to a CSV File