链滴

MySQL 中 JSON 类型应用

作者：mnizht

原文链接：https://ld246.com/article/1563329010115

来源网站：链滴

MySQL 8.0

The JSON Data Type

MySQL 从 5.7.8 开始，支持原生的JSON数据类型。可以高效的访问JSON文档中的数据。与在字符列中存储JSON格式字符串相比，JSON数据类型有以下优势：

- 可自动验证存储的JSON数据格式是否正确。
- 优化的存储格式。存储在JSON列中的JSON文档将转化为内部格式，以允许对文档元素进行快速访。当服务器读取以二进制存储的JSON值时，不需要再对文本解析后取值。二进制格式的结构使服务能够直接通过键或数组索引查找子对象或嵌套值，而无需在文档之前或之后读取所有值。

JSON列占用的空间与LONGBLOB或LONGTEXT基本相同。

在MySQL 8.0.13 之前，JSON列不能定义非null的默认值。

以下测试mysql版本：8.0.16, 个别测试会与较老的版本结果不一致。

创建JSON值

JSON数组包含由逗号分隔并由中括号包裹的值列表

```
["abc", 10, null, true, false]
```

JSON对象包含一组由逗号分隔并且由大括号包裹的键值对

```
{"k1": "value", "k2": 10}
```

JSON对象中的键必须是字符串。JSON数组中的元素和JSON对象键值允许嵌套。

```
[99, {"id": "HK500", "cost": 75.99}, ["hot", "cold"]]
{"k1": "value", "k2": [10, 20]}
```

JSON值在插入前会校验正确性。

```
create table t1 (jdoc JSON);

insert into t1 values
('{"key1":"value1","key2":"value2"}');

insert into t1 values
('"stringvalue"');

insert into t1 values
('[1, 2,');
/**插入的value值中下标6的位置有错
> 3140 - Invalid JSON text: "Invalid value." at position 6 in value for column 't1.jdoc'.*/
```

JSON_TYPE()函数需要JSON参数，并尝试将其解析为JSON值。如果值有效，则返回值的JSON类型否则报错。

```
mysql> SELECT JSON_TYPE('["a", "b", 1]');
```

```
> ARRAY

mysql> SELECT JSON_TYPE('"hello"');
> STRING
mysql> select json_type('{"key":"value"}') type;
> OBJECT

mysql> SELECT JSON_TYPE('hello');
> ERROR 3146 (22032): Invalid data type for JSON data in argument 1
to function json_type; a JSON string or JSON type is required.
```

JSON_ARRAY()函数可以将传入的参数（允许null）转化为一个JSON数组

```
mysql> SELECT JSON_ARRAY('a', 1, NOW());
> ["a", 1, "2019-07-13 15:56:47.000000"]
```

JSON_OBJECT()函数可将传入参数（允许null）转化为一个JSON对象

```
mysql> SELECT JSON_OBJECT('key1', 1, 'key2', 'abc');
> {"key1": 1, "key2": "abc"}

/**
* JSON对象中key值不能为null，value可以为null*/
mysql> SELECT JSON_OBJECT('A','a',null,null,'C','c');
> JSON documents may not contain NULL member names.
```

JSON_MERGE_PRESERVE()接受两个或多个JSON文档并返回组合结果(关于JSON值的合并，下面会更加详细的测试)

```
mysql> SELECT JSON_MERGE_PRESERVE('["a", 1]', '{"key": "value"}');
>    ["a", 1, {"key": "value"}]
```

JSON值可以赋给用户自定义变量

```
mysql> set @j = JSON_OBJECT('key','value');
mysql> select @j;
> {"key": "value"}
```

然而，用户定义的变量不能是JSON数据类型，因此尽管前面示例中的@j看起来像JSON值并且具有与JSON值相同的字符集和排序规则，但它没有JSON数据类型。相反，JSON_OBJECT(的结果在分配给变量时会转换成字符串。

ps：官方文档上的这段话可能只是提示一下用户变量的数据类型不能是JSON，但是使用起来好像也什么问题。

下面的例子可以看到，变量是正确的json格式数据时，使用json函数对其操作都没问题。

```
mysql> SET @x = '{ "a": 1, "b": 2 }',
    >     @y = '{ "a": 3, "c": 4 }',
    >     @z = '{ "a": 5, "d": 6 }';

mysql> SELECT  JSON_MERGE_PATCH(@x, @y, @z)    AS Patch,
    ->         JSON_MERGE_PRESERVE(@x, @y, @z) AS Preserve\G
```

```
>   Patch: {"a": 5, "b": 2, "c": 4, "d": 6}
>Preserve: {"a": [1, 3, 5], "b": 2, "c": 4, "d": 6}


mysql> SET @j = '["a", ["b", "c"], "d"]';
mysql> SELECT JSON_REMOVE(@j, '$[1]');
> ["a", "d"]

mysql> set @a = Json_object('key','value'), @b = json_object('key','value2');
mysql> select json_merge_preserve(@a,@b);
> {"key": ["value", "value2"]}
```

通过转化json值生成的字符串，字符集是utf8mb4，排序规则是utf8mb4_bin

```
mysql> SELECT CHARSET(@j), COLLATION(@j);
>  utf8mb4  |  utf8mb4_bin
```

由于utf8mb4_bin是二进制排序规则，因此json值的比较区分大小写

```
mysql> SELECT JSON_ARRAY('x') = JSON_ARRAY('x'),JSON_ARRAY('x') = JSON_ARRAY('X');
> 1  |  0
```

区分大小写也适用于JSON null，true和false 文字，他们必须始终以小写形式写入。只有小写字母时json才是json有效值。

```
mysql>  SELECT JSON_VALID('null'), JSON_VALID('Null'), JSON_VALID('NULL');
>  1    |    0    |    0

mysql>  SELECT CAST('null' AS JSON);
> null

mysql>  SELECT CAST('Null' AS JSON);
> Invalid JSON text in argument 1 to function cast_as_json: "Invalid value." at position 0.
```

JSON是区分大小写的，而SQL是不区分的。下列值都可以成功被识别为null。

```
mysql> SELECT ISNULL(null), ISNULL(Null), ISNULL(NULL);
> 1    |    1    |    1
```

当你想要讲引号字符(" 或 ")插入到JSON文档中，你需要使用 \ 转义字符。

```
mysql> CREATE TABLE facts (sentence JSON);
mysql> INSERT INTO facts VALUES
    >   (JSON_OBJECT("mascot", "Our mascot is a dolphin named \"Sakila\"."));
```

如果将值作为JSON对象文字传入，则此方法不起作用。如下示例，第一个 \
 的位置会被当做Our ma...这个属性的结束位置，所以报了缺少 逗号 或 大括号 的错误。

```
mysql> INSERT INTO facts VALUES
    >   ('{"mascot": "Our mascot is a dolphin named \"Sakila\"."}');
>    > 3140 - Invalid JSON text: "Missing a comma or '}' after an object member." at position
3 in value for column 'facts.sentence'.
```

这里需要使用双反斜杠来使之生效

```
mysql> INSERT INTO facts VALUES
    >   ('{"mascot": "Our mascot is a dolphin named \\"Sakila\\"."}');
```

查询查看效果,可以看到两种方式的结果是一样的。

```
mysql>    SELECT * FROM facts;
>    {"mascot": "Our mascot is a dolphin named \"Sakila\"."}
   {"mascot": "Our mascot is a dolphin named \"Sakila\"."}
```

要使用键查找特定的值，可以使用column-path操作符 ->,可以看到查询结果是完整的value值，包引号和转义符

```
mysql> select sentence->"$.mascot" from facts;
> "Our mascot is a dolphin named \"Sakila\"."
```

有时候我们只需要里面的字符串，可以使用 ->> 运算符来查询;这样查询到的结果就只是需要显示的了。

```
mysql> select sentence->>"$.mascot" from facts;
>    Our mascot is a dolphin named "Sakila".
```

如果启动了NO_BACKSLASH_ESCAPES服务器SQL模式，则前面的插入对象的实将无法正常工作。如果设置了此模式，则可以使用单个反斜杠而不是双反斜杠来插入JSON对象文字并保留反斜杠。如果在执行插入时使用JSON_OBJECT() 函数并设置了此模式，则必须替换单引号和引号

```
mysql> INSERT INTO facts VALUES (JSON_OBJECT('mascot', 'Our mascot is a dolphin named Sakila.'));

mysql>    SELECT * FROM facts;
>    {"mascot": "Our mascot is a dolphin named \"Sakila\"."}
```

JSON值的规范化，合并和自动包装

解析字符串并发现它是有效的JSON文档时，它也会进行规范化。比如JSON对象中不能有重复的key那么有重复key时，后面的value会覆盖前面的

```
mysql> SELECT JSON_OBJECT('key1', 1, 'key2', 'abc', 'key1', 'def');
>    {"key1": "def", "key2": "abc"}
```

将值插入json列时也会执行规范化

```
mysql>    CREATE TABLE t1 (c1 JSON);
mysql>    INSERT INTO t1 VALUES
('{"x": 17, "x": "red"}'),
('{"x": 17, "x": "red", "x": [3, 5, 7]}');

mysql>    SELECT c1 FROM t1;
>    {"x": "red"}
   {"x": [3, 5, 7]}
```

这种 “last duplicate key wins（最后重复秘钥获胜）” 的规则由 RFC 7159建议并且大多数JavaScript解析器都支持这个规则。（Bug #86866,Bug #26369555）

在8.0.3之前的MySQL版本中，重复的key value会被丢弃。如在5.7.26版本的MySL做上面同样的测试，结果就不一样

```
mysql> SELECT JSON_OBJECT('key1', 1, 'key2', 'abc', 'key1', 'def');
>    {"key1": 1, "key2": "abc"}

mysql>    CREATE TABLE t1 (c1 JSON);
mysql>    INSERT INTO t1 VALUES
('{"x": 17, "x": "red"}'),
('{"x": 17, "x": "red", "x": [3, 5, 7]}');

mysql>    SELECT c1 FROM t1;
>    {"x": 17}
   {"x": 17}
```

MySQL还会丢弃原始JSON文档中键、值或元素之间的额外空格。为了使查找更有效，它还对JSON象的键进行排序。目前在这两个版本中做的简单测试，排序结果都是一样的。当然官方文档上也有提：此排序结果可能会发生变化，并且不保证在各个版本中保持一致。

```
mysql>    select JSON_OBJECT('key1','value1','key3','value3','key2','value2');
>    {"key1": "value1", "key2": "value2", "key3": "value3"}
```

合并JSON值

MySQL中有三个合并方法：

- JSON_MERGE：在MySQL 8.0.3中已弃用，并且在将来的版本中将被删除。
- JSON_MERGE_PRESERVE：保留重复键的值。就是重命名后的JSON_MERGE()
- JSON_MERGE_PATCH：丢弃除最后一个值之外的所有键

合并数组：将多个数组合并成单个数组。

JSON_MERGE_PRESERVE() 通过将后一数组追加到前一数组末尾实现。

JSON_MERGE_PATCH() 将每个参数视为由单个元素组成的数组（因此它们的下标都是0），然后应用“last duplicate key wins” 规则取最后一个参数。

```
mysql> SELECT
    ->    JSON_MERGE_PRESERVE('[1, 2]', '["a", "b", "c"]', '[true, false]') AS Preserve,
    ->    JSON_MERGE_PATCH('[1, 2]', '["a", "b", "c"]', '[true, false]') AS Patch\G
*************************** 1. row ***************************
Preserve: [1, 2, "a", "b", "c", true, false]
   Patch: [true, false]
```

注意：方法中都允许有null，但是结果需要注意。JSON_MERGE_PRESERVE() 方法中，若作为参数数组中有个null值，那最终结果中也会有个null，方法不会去除null值和重复值。

```
mysql> SELECT JSON_MERGE_PRESERVE('[1, 2]', '["a", "b", "c"]','[true, false]','["a", null, "c"]') AS
Preserve;
>    [1, 2, "a", "b", "c", true, false, "a", null, "c"]
```

但如果作为参数的数组本身就是null，那合并结果也是null

```
mysql>    SELECT JSON_MERGE_PRESERVE('[1, 2]', '["a", "b", "c"]',null,'[true, false]') AS Preserve

>    Null
```

JSON_MERGE_PATCH() 方法由于只保留最后一个数组，所以null对它并没有什么影响，只有当最后个参数为null时结果才为null

```
mysql>    SELECT JSON_MERGE_PATCH('[1, 2]', '["a", "b", "c"]', '[true, false]',null,'["a", "b", "c"]',
ull) AS Patch;
>    Null
```

合并对象：将多个对象合并成单个对象

JSON_MERGE_PRESERVE():将多个对象的键值对组合成一个新对象，会将重复键的值组成一个数组不会去除重复值和null）赋给这个键。

JSON_MERGE_PATCH():将多个对象的键值对组合成一个新对象，重复键的值会用 last duplicate key wins（最后重复秘钥获胜）规则取最后一个。

```
mysql> SELECT JSON_MERGE_PRESERVE('{"a": 1, "b": 2}', '{"c": 3, "a": 1}', '{"c": 5, "d": 3}','{"d":nu
l}') AS Preserve,
JSON_MERGE_PATCH('{"a": 3, "b": 2}', '{"c": 3, "a": 4}', '{"c": 5, "d": 3}') AS Patch\G
*************************** 1. row ***************************
Preserve: {"a": [1, 1], "b": 2, "c": [3, 5], "d": [3, null]}
   Patch: {"a": 4, "b": 2, "c": 5, "d": 3}
```

合并非阵列值：

在需要数组值的上下文中使用的非阵列值被自动包装：该值被 [和] 字符包围以将其转换为数组。在列语句中，每个参数都自动包装为数组（[1],[2]）。然后按照 合并数组 的规则进行合并。

```
mysql> SELECT
    ->   JSON_MERGE_PRESERVE('1', '2') AS Preserve,
    ->   JSON_MERGE_PATCH('1', '2') AS Patch\G
*************************** 1. row ***************************
Preserve: [1, 2]
   Patch: 2
```

需要注意的是，参数必须是正确的json类型值。可以使用 JSON_TYPE()方法检查值是否正确。

```
mysql>    SELECT JSON_MERGE_PRESERVE('"a"', '"b"') AS Preserve, JSON_MERGE_PATCH('"a"',
"b"') AS Patch;
*************************** 1. row ***************************
Preserve: ["a", "b"]
   Patch: "b"


mysql>    SELECT JSON_MERGE_PRESERVE('a', 'b') AS Preserve, JSON_MERGE_PATCH('a', 'b')
S Patch;
>    Invalid JSON text in argument 1 to function json_merge_preserve: "Invalid value." at posit
on 0.
```

合并数组和对象的组合：合并的参数中既有数组，又有对象

合并时会将对象自动包装为数组，然后按照 合并数组 的规则合并。

```
mysql> SELECT
    ->   JSON_MERGE_PRESERVE('[10, 20]', '{"a": "x", "b": "y"}') AS Preserve,
    ->   JSON_MERGE_PATCH('[10, 20]', '{"a": "x", "b": "y"}') AS Patch\G
*************************** 1. row ***************************
Preserve: [10, 20, {"a": "x", "b": "y"}]
   Patch: {"a": "x", "b": "y"}
```

查询和修改JSON值

JSON路径表达式选择JSON文档中的值。

路径表达式对于提取 JSON 文档的一部分或修改JSON文档的函数很有用，以指定该文档中的操作位。

以下查询从JSON文档中提取名称为key的成员的值：

```
mysql> SELECT JSON_EXTRACT('{"id": 14, "name": "Aztalan"}', '$.name');
>     "Aztalan"
```

路径语法使用前导$字符来表示正在计算的JSON文档，后面可以跟选择器，来连续指示文档更具体的分：

- 后跟秘钥名称的句点用具有给定键的对象命名成员。如果没有引号的名称在路径表达式中不合法（如，如果它包含空格），则必须在双引号内指定键名。
- 附加到选择数组的路径的 [N] 命名数组中位置N处的值。数组位置是从零开始的整数。如果path没选择数组值，则path[0]计算为与path相同的值

```
mysql>    SELECT JSON_SET('"x"', '$[0]', 'a');
>    "a"
```

- [M] 到 [N] 指定以位置M处的值开始并以位置N处的值结束的数组值的子集或范围。

last支持作为最右边数组元素的索引的同义词。还支持数组元素的相对寻址。如果path未选择数组值则path[last] 将计算为与path相同的值。

- 路径可以包含 * 或 ** 通配符
  - .[*] 计算JSON对象中所有成员的值

```
mysql>    SELECT JSON_EXTRACT('{"id": 14, "name": "Aztalan"}', '$.*');
>    [14, "Aztalan"]
```

  - [*] 计算JSON数组中所有元素的值

```
mysql>    SELECT JSON_EXTRACT('[1,2,"a","b",3,"c",null]', '$[*]');
>    [1, 2, "a", "b", 3, "c", null]
```

  - prefix**suffix 计算所有以命名前缀开头并以命名后缀结尾的路径

```
mysql>    select json_extract('{ "a": [ [ 3, 2 ], [ { "c" : "d" }, 1 ] ], "b": { "b.c" : 6 }, "one potato": 7, c" : 8 }','$**.c');
>    [8, "d"]
```

这里之所以是[8, "d"],我们先换一种方法查询,可以看到，这两个查询的路径都是以 .c 结尾的。

```
mysql>    select json_extract('{ "a": [ [ 3, 2 ], [ { "c" : "d" }, 1 ] ], "b": { "b.c" : 6 }, "one potato": 7, c" : 8 }','$.a[1][0].c');
```

```
>    "d"

mysql>    select json_extract('{ "a": [ [ 3, 2 ], [ { "c" : "d" }, 1 ] ], "b": { "b.c" : 6 }, "one potato": 7,
c" : 8 }','$.c');
>    8
```

● 文档中不存在的路径（评估为不存在的数据）的计算结果为Null

让 $ 引用这个带有三个元素的JSON数组：

```
[3, {"a" : [5, 6], "b" : 10}, [99, 100]]
```

然后：

● $[0] 的值为 3
● $[1] 的值为 {"a" : [5, 6], "b" : 10}
● $[2] 的值为 [99, 100]
● $[3] 的值为 Null

因为 [1] 和 [2] 的值为非标量值，所以它们可以用作更具体的路径表达式的基础来选择嵌套的值

● $[1].a 的值为 [5, 6]
● $[1].a[1] 的值为 6
● $[1].b 的值为 10
● $[2][0] 的值为 99

如前所述，如果路径表达式中的未加引号的键名称不合法，则必须引用命名键的路径组件。让$引用个值

```
{"a fish": "shark", "a bird": "sparrow"}
```

key都包含空格，必须引用：

● $."a fish" 的值为 shark
● $."a bird" 的值为 sparrow

使用通配符的路径计算的源数据可以是一个包含多个值的数组

```
mysql> SELECT JSON_EXTRACT('{"a": 1, "b": 2, "c": [3, 4, 5]}', '$.*');
>     [1, 2, [3, 4, 5]]

mysql> SELECT JSON_EXTRACT('{"a": 1, "b": 2, "c": [3, 4, 5]}', '$.c[*]');
>     [3, 4, 5]
```

来自JSON数组的范围。可以使用带有to关键字的范围来指定JSON数组的子集。

```
mysql> SELECT JSON_EXTRACT('[1, 2, 3, 4, 5]', '$[last-3 to last-1]');
>    [2, 3, 4]
```

如果针对不是数组的值计算路径，则评估结果与将值包装在单个元素数组中的结果相同。替换指定值因为[0] 和 [last] 都指向 “Sakila” ，所以该值被替换为10，返回结果为10；$[其它值] 找不到可替

的值，所以最终返回原值

```
mysql>    SELECT JSON_REPLACE('"Sakila"', '$[0]',10) zero, JSON_REPLACE('"Sakila"', '$[last]',10
 last, JSON_REPLACE('"Sakila"', '$[1]',10) other;
*************************** 1. row ***************************
zero: 10
last: 10
other: "Sakila"
```

你可以将column-> path 与 JSON列标识符和 JSON路径表达式一起用作JSON_EXTRACT (列，路径的同义词。

某些函数采用现有的JSON文档，以某种方式对其进行修改，并返回生成的修改后的文档。路径表达指示文档中的更改位置。例如：JSON_SET(), JSON_INSERT(), JSON_REPLACE() 函数各自采用JSO文档，以及一个或多个路径值对，这些路径值对描述了修改文档的位置和要使用的值。

这些函数在处理文档中的现有值和不存在值方面有所不同。

考虑这个文档：

```
mysql> SET @j = '["a", {"b": [true, false]}, [10, 20]]';
```

JSON_SET() 替换存在的路径的值，并为不存在的路径添加值。[1].b[0] 原本的true被替换为1， [2][ ] 原本是不存在的，现在新增了个2

```
mysql> SELECT JSON_SET(@j, '$[1].b[0]', 1, '$[2][2]', 2);
>  ["a", {"b": [1, false]}, [10, 20, 2]]
```

JSON_INSERT() 添加新值但不替换现有值

```
mysql> SELECT JSON_INSERT(@j, '$[1].b[0]', 1, '$[2][2]', 2);
>     ["a", {"b": [true, false]}, [10, 20, 2]]
```

JSON_REPLACE() 替换现有的值但忽略新值

```
mysql> SELECT JSON_REPLACE(@j, '$[1].b[0]', 1, '$[2][2]', 2);
>  ["a", {"b": [1, false]}, [10, 20]]
```

路径值对从左到右进行评估。通过评估一对产生的文档成为下一个评估的源值。（类似于递归）

JSON_RMOVE() 接受一个JSON文档和一个或多个指定要从文档中删除的值的路径。返回值是原始文减去文档中存在的路径选择的值：

```
mysql> SELECT JSON_REMOVE(@j, '$[2]', '$[1].b[1]', '$[1].b[1]');
> ["a", {"b": [true]}]
```

JSON路径语法

MySQL中的很多JSON函数需要路径表达式来标识JSON文档中的特定元素。一个路径由路径范围后一个或多个分支路径组成。对于MySQL JSON函数使用的路径，范围始终是正在搜索或以其它方式操的文档，由前导 $ 字符表示。路径分支用 . 字符分隔。数组中的元素用 [N] 表示，其中N是非负整数秘钥的名称必须是双引号字符串或有效的ECMAScript标识符（请参阅http://www.ecma-internation l.org/ecma-262/5.1/#sec-7.6）。路径表达式（如JSON文本）应使用ascii，utf8或utf8mb4字符集行编码。其它字符编码被隐式强行转换为utf8mb4。

```
pathExpression:
    scope[(pathLeg)*]

pathLeg:
    member | arrayLocation | doubleAsterisk

member:
    period ( keyName | asterisk )

arrayLocation:
    leftBracket ( nonNegativeInteger | asterisk ) rightBracket

keyName:
    ESIdentifier | doubleQuotedString

doubleAsterisk:
    '**'

period:
    '.'

asterisk:
    '*'

leftBracket:
    '['

rightBracket:
    ']'
```

如上所述，在MySQL中，路径的范围始终是正在操作的文档，表示为。你可以使用 '' 字符作为JSON路径表达式中文档的别名。

JSON值的比较和排序

JSON值可以使用 =, <, <=, >, >=, <>, != 和 <=> 运算符进行比较。

其中

<=>：NULL安全相等。此运算符执行与 = 号运算符类似的相等比较，但如果两个操作数均为NULL则返回1而不是NULL；如果一个操作数为NULL，则返回0而不是NULL。

<=> 等同于标准SQL 中 is not distinct from 运算符

```
mysql> SELECT 1 <=> 1, NULL <=> NULL, 1 <=> NULL;
       -> 1, 1, 0
mysql> SELECT 1 = 1, NULL = NULL, 1 = NULL;
       -> 1, NULL, NULL
```

对于行比较，(a, b) <=> (x, y) 等效于（a <=> x) AND (b <=> y)

JSON值尚不支持以下比较运算符和函数：

- BETWEEN

- IN()
- GREATEST()
- LEAST()

为了使用上述列出的比较运算符和函数，可以将JSON值转换为本机MySQL数值或字符串数据类型，便它们具有一致的非JSON标量类型。

JSON值的比较发生在两个级别。

第一级比较基于比较值的JSON类型。如果类型不同，则比较结果仅由哪种类型具有更高优先级来确。

如果这两个值具有相同的JSON类型，则使用特定于类型的规则进行第二级比较。

下面列出了JSON类型的优先级，从最高优先级到最低优先级。（类型名称是JSON_TYPE（）函数返的类型名称。）同一行显示的类型优先级相同。列表中前面列出的任何具有JSON类型的值都比列表稍后列出的具有JSON类型的任何值都要大。

BLOB
BIT
OPAQUE
DATETIME
TIME
DATE
BOOLEAN
ARRAY
OBJECT
STRING
INTEGER, DOUBLE
NULL

对于具有相同优先级的JSON值，比较规则是特定于类型的：

- BLOB

比较两个值的前N个字节，其中N是较短值中的字节数。如果两个值的前N个字节相同，则将较短值在前面。

- BIT

与 BLOB规则相同

- OPAQUE

与 BLOB 规则相同。OPAQUE 值是未归类为其他类型之一的值。

- DATETIME

表示较早时间点的值排在表示较晚时间点的值之前。如果两个值最初分别来自MySQL DATETIME 和 TMESTAMP类型，且它们表示相同的时间点，则它们相等。

- TIME

两个时间值中较小的一个排在较大的前面。

- DATE

较早的日期排在最近的日期之前。

- ARRAY

如果两个JSON数组具有相同的长度并且数组中相应位置的值相等，则它们是相等的。

如果数组不相等，则它们的顺序由第一个不同的元素来确定。首先排序在该位置具有较小值的数组。

如果较短数组中的所有值都等于较长数组中的相应值，则首先排序较短的数组。

例：

```
[] < ["a"] < ["ab"] < ["ab", "cd", "ef"] < ["ab", "ef"]
```

● BOOLEAN

为 false的JSON值小于 为true的JSON值。

● OBJECT

如果它们有相同的键值对，它们就是相等的。键的顺序不需要完全相同。

```
{"a": 1, "b": 2} = {"b": 2, "a": 1}
```

两个不相等的对象的顺序是未指定的但是确定性的。官方文档上只有这一句。

我自己测试了一下, 可以看到，JSON_OBJECT() 在处理JSON文档时会将key排序，所以键值的顺序于对象的比较没有影响。关于排序的结果，可以看到，就是按照key1,value1,key2,value2...这样的顺来比较，比较规则与 ARRAY 相同。

```
mysql>    select JSON_OBJECT('a',3, 'b', 2) = JSON_OBJECT('b',2, 'a', 3);
>    1

mysql>    select JSON_OBJECT('b',2, 'a', 2);
>    {"a": 2, "b": 2}

mysql>    select JSON_OBJECT('a',3, 'b', 2) < JSON_OBJECT('b',2, 'a', 2);
>    0

mysql>    select JSON_OBJECT('a',3, 'b', 2) < JSON_OBJECT('b',2, 'a', 3, 'c',3);
>    1
```

● STRING

字符串比较时会按照utf8mb4来比较。规则是前N个字节上按字母顺序排序，其中N是较短字符串的度。如果两个字符串的前N个字节相同，则较短的字符串小于较长的字符串。

```
"a" < "ab" < "b" < "bc"
```

这种排序等同于使用排序规则utf8mb4_bin 的SQL字符串的排序。因为utf8mb4_bin是二进制排序规，所以JSON值的比较区分大小写

```
"A" < "a"
```

● INTEGER,DOUBLE

JSON值可以包含精确值数字和近似值数字。需要注意的是，比较JSON值中的数字规则与比较MySQ数值类型的规则有所不同。

● 在使用本机MySQL INT和DOUBLE数字类型进行比较时，已知所有比较都涉及int和double，此对于所有行，int 将转化为 double。也就是说精确值数字被转化为近似值数字。

● 另一方面，如果查询比较包含数字的两个JSON列，则无法事先知道数字是 int 还是 double。了在所有行中提供最一致的行为，MySQL将近似值数字转化为精确值数字。生成的顺序是一致的，且不会丢失精确值数字的精度。

```
9223372036854775805 < 9223372036854775806 < 9223372036854775807
```

< 9.223372036854776e18 = 9223372036854776000 < 9223372036854776001

如果JSON比较使用非JSON数字比较规则，则可能发生不一致的排序。通常数字按MySQL排序规则是下列结果：

- Integer 比较

9223372036854775805 < 9223372036854775806 < 9223372036854775807

- Double 比较

9223372036854775805 = 9223372036854775806 = 9223372036854775807 = 9.22337203685776e18

任何JSON值与 SQL的NULL值比较，结果都是UNKNOWN。

为了比较JSON值和非JSON值，非JSON值会按照下列规则转化为JSON值，然后按照之前提到的规做比较。

JSON值和非JSON值相互转换

下表提供了MySQL在JSON值和其它类型值之间进行转换时遵循的规则的摘要：

其它类型	CAST(其它类型转换为JSON)	AST(JSON 转换为其它类型)
JSON	不改变	不改变
utf8 character type (utf8mb4,utf8, ascii)	字符串解析为JSON值。	SON值被序列化为utf8mb4字符串.
Other character types	其它字符编码隐式转换为utf8mb4,并按照u JSON值被序列化为utf8mb4字符串，	f8字符类型的描述进行处理. 后转换为其它字符编码。结果可能没有意义（乱码）.
NULL	结果是值为NULL的JSON.	可用.
Geometry types	The geometry value is converted into a JSON	document by callingST_AsGeoJSON(). Illegal operation. Worka ound: Pass the result of CAST(*json_val*AS CHAR) to ST_GeomFromGeoJSON().
All other types	Results in a JSON document consisting of a si	gle scalar value. Succeeds if the JSON document consists of a s ngle scalar value of the target type and that scalar value can be cast to the target type. Other ise, returns NULL and produces a warning.

JSON值的ORDER BY 和 GROUP BY 按照以下原则工作：

- 标量JSON值的排序使用上文所述规则
- 对于升序排序，SQL NULL排在所有JSON值之前，包括JSON空文字；对于降序排序，SQL NULL在所有JSON值之后。
- JSON的键排序时，需要 max_sort_length 系统变量来确定参与比较的字符数，比如 ‘abc’ 和 ‘bcdefg’ 比较时，如果 max_sort_length = 3，那么参与比较的字符其实是 ‘abc’ 和 ‘abc’,即 ‘abc’ = ‘abcdefg’；

如果max_sort_length = 4, 那么参与比较的字符其实是 'abc' 和 'abcd',即 ‘abc’ < 'abcdefg';

- 目前不支持对非标量值进行排序，会发出警告

对于排序，将JSON标量转换为其它一些本地MySQL类型可能是有益的。例如，如果名为jdoc的列包具有由id键和非负值组成的成员的JSON对象，请使用此表达式按id值排序：

```
ORDER BY CAST(JSON_EXTRACT(jdoc, '$.id') AS UNSIGNED)
```

如果确定生成的列定义为使用与ORDER BY中相同的表达式，则MySQL优化器会识别并考虑使用索引行查询计划。

JSON值的聚合

对于JSON值的聚合，SQL NULL 会像其它数据类型一样被忽略。非空值转换为数字类型并聚合，但 MN(), MAX(), GROUP_CONCAT() 除外。转换为数字应该会为JSON值产生一个有意义的结果，这些值数值标量，尽管（取决于值）可能会发生截断和精度损失。转换为的数字的JSON值也可能是没有意的结果。