存储在JSON列中的任何JSON文档的大小限于[max\_allowed\_packet](https://dev.mysql.com/doc/refman/5.7/en/server-system-variables.html#sysvar_max_allowed_packet)系统变量的值是很重要的。（当服务器在内存中操作JSON值时，它可以大于这个值;当服务器存储它时，这个限制就适用。

JSON列不能有默认值。

与JSON数据类型一起，可以使用一组SQL函数来对JSON值进行操作，例如创建，操作和搜索。

还可以使用一组用于GeoJSON值的空间函数。

JSON列，如其他二进制类型的列，不直接索引; 相反，您可以在生成的列上创建索引，从列中提取标量值 JSON。

1. 创建JSON值

JSON数组包含由逗号分隔的值的列表，并包含在内[] 字符中：

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

JSON对象包含一组内由逗号分隔的和封闭的键-值对{ }字符：

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

在JSON数组元素和JSON对象键值中允许嵌套：

[99, {"id": "HK500", "cost": 75.99}, ["hot", "cold"]]

{"k1": "value", "k2": [10, 20]}

mysql> CREATE TABLE t1 (jdoc JSON);

mysql> INSERT INTO t1 VALUES('{"key1": "value1", "key2": "value2"}');

json\_type(json) 显示json类型

mysql> SELECT JSON\_TYPE('["a", "b", 1]');

+----------------------------+

| JSON\_TYPE('["a", "b", 1]') |

+----------------------------+

| ARRAY |

+----------------------------+

MySQL使用utf8mb4字符集和utf8mb4\_bin归类来处理JSON上下文中使用的 字符串 。其他字符集中的字符串将utf8mb4根据需要进行转换。

1. 从组件元素编写JSON值的函数

[JSON\_ARRAY()](https://dev.mysql.com/doc/refman/5.7/en/json-creation-functions.html#function_json-array)接受一个（可能是空的）值列表并返回一个包含这些值的JSON数组：

mysql> SELECT JSON\_ARRAY('a', 1, NOW());

+----------------------------------------+

| JSON\_ARRAY('a', 1, NOW()) |

+----------------------------------------+

| ["a", 1, "2015-07-27 09:43:47.000000"] |

+----------------------------------------+

[JSON\_OBJECT()](https://dev.mysql.com/doc/refman/5.7/en/json-creation-functions.html#function_json-object) 需要一个（可能是空的）键值对列表并返回一个包含这些对的JSON对象：

mysql> SELECT JSON\_OBJECT('key1', 1, 'key2', 'abc');

+---------------------------------------+

| JSON\_OBJECT('key1', 1, 'key2', 'abc') |

+---------------------------------------+

| {"key1": 1, "key2": "abc"} |

+---------------------------------------+

[JSON\_MERGE()](https://dev.mysql.com/doc/refman/5.7/en/json-modification-functions.html#function_json-merge) 需要两个或多个JSON文档并返回组合结果：

mysql> SELECT JSON\_MERGE('["a", 1]', '{"key": "value"}');

+--------------------------------------------+

| JSON\_MERGE('["a", 1]', '{"key": "value"}') |

+--------------------------------------------+

| ["a", 1, {"key": "value"}] |

+--------------------------------------------+

JSON值可以分配给用户定义的变量：

mysql> SET @j = JSON\_OBJECT('key', 'value');

但是，用户定义的变量不能是 JSON数据类型，因此尽管 @j在前面的示例中看起来像JSON值，并且具有与JSON值相同的字符集和归类，但它不具有 JSON数据类型。相反，[JSON\_OBJECT()](https://dev.mysql.com/doc/refman/5.7/en/json-creation-functions.html#function_json-object)分配给变量的结果 将转换为字符串。

通过转换JSON值生成的字符串具有以下字符集utf8mb4和排序规则 utf8mb4\_bin：

mysql> SELECT CHARSET(@j), COLLATION(@j);

+-------------+---------------+

| CHARSET(@j) | COLLATION(@j) |

+-------------+---------------+

| utf8mb4 | utf8mb4\_bin |

+-------------+---------------+

因为utf8mb4\_bin是二进制排序规则，所以JSON值的比较区分大小写。

区分大小写也适用于JSON  null，true和 false文字，它们总是必须以小写字母写入：

mysql> SELECT JSON\_VALID('null'), JSON\_VALID('Null'), JSON\_VALID('NULL');

+--------------------+--------------------+--------------------+

| JSON\_VALID('null') | JSON\_VALID('Null') | JSON\_VALID('NULL') |

+--------------------+--------------------+--------------------+

| 1 | 0 | 0 |

+--------------------+--------------------+--------------------+

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

+----------------------+

| CAST('null' AS JSON) |

+----------------------+

| null |

+----------------------+

将mascot: The MySQL mascot is a dolphin named "Sakila".插入表中：

使用MySQL [JSON\_OBJECT()](https://dev.mysql.com/doc/refman/5.7/en/json-creation-functions.html#function_json-object)函数。在这种情况下，您必须使用反斜杠来转义每个引号字符，如下所示

mysql> INSERT INTO facts VALUES

> (JSON\_OBJECT("mascot", "Our mascot is a dolphin named \"Sakila\"."));

如果将该值作为JSON对象文本插入，则不能以相同的方式工作，在这种情况下，必须使用双反斜杠转义序列，如下所示：

mysql> INSERT INTO facts VALUES

> ('{"mascot": "Our mascot is a dolphin named \\"Sakila\\"."}');

要查找这个特定的句子 mascot作为关键字，可以使用列路径运算符 [->](https://dev.mysql.com/doc/refman/5.7/en/json-search-functions.html#operator_json-column-path)，如下所示：

mysql> SELECT col->"$.mascot" FROM qtest;

+---------------------------------------------+

| col->"$.mascot" |

+---------------------------------------------+

| "Our mascot is a dolphin named \"Sakila\"." |

+---------------------------------------------+

这使反斜杠和周围的引号完好无损。要使用mascot键显示所需的值 ，但不包括周围的引号或任何转义符[->>](https://dev.mysql.com/doc/refman/5.7/en/json-search-functions.html#operator_json-inline-path)，请使用内联路径运算符 ，如下所示：

mysql> SELECT sentence->>"$.mascot" FROM facts;

+-----------------------------------------+

| sentence->>"$.mascot" |

+-----------------------------------------+

| Our mascot is a dolphin named "Sakila". |

+-----------------------------------------+

1. JSON值的规范化，合并和自动套接

#### JSON规范化

1.如果在插入json格式数据时 后面出现的key与前面出现的key相同时会忽略后面出现的；

2.MySQL还会丢弃原始JSON文档中的键，值或元素之间的额外空白。为了使查找效率更高，它还对JSON对象的键进行排序。

3.合并时多个对象产生单个对象。如果多个对象具有相同的键，则生成的合并对象中该键的值是包含键值的数组：

#### 合并JSON值

在组合多个数组的上下文中，通过将稍后命名的数组连接到第一个数组的末尾，将这些数组合并成一个数组。在以下示例中， [JSON\_MERGE()](https://dev.mysql.com/doc/refman/5.7/en/json-modification-functions.html#function_json-merge)将其参数合并到单个数组中：mysql> SELECT JSON\_MERGE('[1, 2]', '["a", "b"]', '[true, false]');

+-----------------------------------------------------+

| JSON\_MERGE('[1, 2]', '["a", "b"]', '[true, false]') |

+-----------------------------------------------------+

| [1, 2, "a", "b", true, false] |

+-----------------------------------------------------+

合并时多个对象产生单个对象。如果多个对象具有相同的键，则生成的合并对象中该键的值是包含键值的数组：

mysql> SELECT JSON\_MERGE('{"a": 1, "b": 2}', '{"c": 3, "a": 4}');

+----------------------------------------------------+

| JSON\_MERGE('{"a": 1, "b": 2}', '{"c": 3, "a": 4}') |

+----------------------------------------------------+

| {"a": [1, 4], "b": 2, "c": 3} |

+----------------------------------------------------+

需要数组值的上下文中使用的非数组值是自动绑定的：该值被包围，[ 并将]其转换为数组。在下面的语句中，每个参数都以数组（[1]，[2]）的形式自动包装。这些然后合并产生一个单一的结果数组：

mysql> SELECT JSON\_MERGE('1', '2');

+----------------------+

| JSON\_MERGE('1', '2') |

+----------------------+

| [1, 2] |

+----------------------+

数组和对象值通过将对象作为数组自动套装并合并两个数组进行合并：

mysql> SELECT JSON\_MERGE('[10, 20]', '{"a": "x", "b": "y"}');

+------------------------------------------------+

| JSON\_MERGE('[10, 20]', '{"a": "x", "b": "y"}') |

+------------------------------------------------+

| [10, 20, {"a": "x", "b": "y"}] |

+------------------------------------------------+

4.搜索和修改JSON值

[***column***->***path***](https://dev.mysql.com/doc/refman/5.7/en/json-search-functions.html#operator_json-column-path)[JSON\_EXTRACT(***column***, ***path***)](https://dev.mysql.com/doc/refman/5.7/en/json-search-functions.html#function_json-extract)

JSON路径表达式path在JSON文档中选择一个值。路径语法使用一个前导$字符来表示正在考虑的JSON文档，可选地，随后是选择器，它们依次指示文档的更多特定部分：

1. 后面跟一个键名的句点用给定的键名称对象中的成员。如果不带引号的名称在路径表达式中不合法（例如，如果包含空格），则必须在双引号内指定密钥名称。
2. [***N***]附加到一个***path***选择一个数组名称的值在***N*** 数组中的位置。阵列位置是从零开始的整数。
3. 路径可以包含\*或 \*\*通配符：

* .[\*] 评估JSON对象中所有成员的值。
* [\*] 计算出JSON数组中所有元素的值。
* ***prefix***\*\****suffix*** 评估所有以该命名前缀开头并以命名后缀结束的路径。

1. 文档中不存在的路径（评估为不存在的数据）评估为NULL。

让我们$引用这个包含三个元素的JSON数组：[3, {"a": [5, 6], "b": 10}, [99, 100]]

* $[0]评估3。
* $[1]评估{"a": [5, 6], "b": 10}。
* $[2]评估[99, 100]。
* $[3]评估为NULL （它指的是不存在的第四个数组元素）。

因为$[1]并$[2] 计算非标量值，所以可以将它们用作选择嵌套值的更具体的路径表达式的基础。例子：

* $[1].a评估[5, 6]。
* $[1].a[1]评估 6。
* $[1].b评估 10。
* $[2][0]评估 99。

1. 如果路径表达式中未加引号的键名称不合法，则必须引用名称键的路径组件。让我们$参考这个值：{"a fish": "shark", "a bird": "sparrow"}

键都包含一个空格，必须引用：

* $."a fish"评估 shark。
* $."a bird"评估 sparrow。

1. 使用通配符的路径评估可以包含多个值的数组：

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

+---------------------------------------------------------+

| 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[\*]');

+------------------------------------------------------------+

| JSON\_EXTRACT('{"a": 1, "b": 2, "c": [3, 4, 5]}', '$.c[\*]') |

+------------------------------------------------------------+

| [3, 4, 5] |

+------------------------------------------------------------+

在以下示例中，路径将$\*\*.b 评估为多个路径（$.a.b和 $.c.b），并生成一个匹配路径值的数组：

mysql> SELECT JSON\_EXTRACT('{"a": {"b": 1}, "c": {"b": 2}}', '$\*\*.b');

+---------------------------------------------------------+

| JSON\_EXTRACT('{"a": {"b": 1}, "c": {"b": 2}}', '$\*\*.b') |

+---------------------------------------------------------+

| [1, 2] |

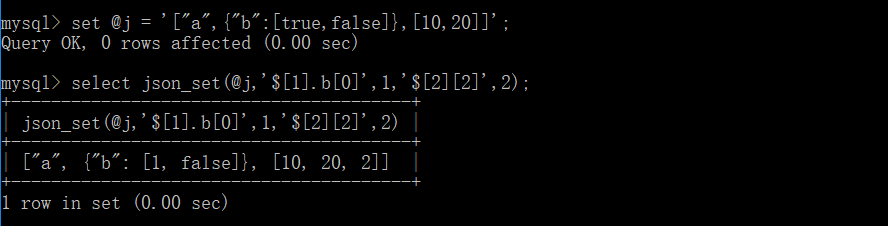
+---------------------------------------------------------+

\*\*\*\*\*\*\*\*\*某些函数使用现有的JSON文档，以某种方式对其进行修改，然后返回生成的修改文档。路径表达式指示要在文档中进行更改的位置。例如，[JSON\_SET()](https://dev.mysql.com/doc/refman/5.7/en/json-modification-functions.html#function_json-set)，[JSON\_INSERT()](https://dev.mysql.com/doc/refman/5.7/en/json-modification-functions.html" \l "function_json-insert)，和 [JSON\_REPLACE()](https://dev.mysql.com/doc/refman/5.7/en/json-modification-functions.html#function_json-replace)功能各采取JSON文档，再加上描述其中修改文档和使用的值的一个或多个路径/值对。这些函数在处理文档中现有和不存在的值方面有所不同。

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

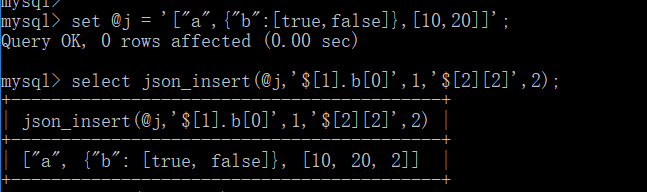
[JSON\_SET()](https://dev.mysql.com/doc/refman/5.7/en/json-modification-functions.html#function_json-set) 替换存在的路径的值，并为不存在的路径添加值：

mysql> SELECT JSON\_SET(@j, '$[1].b[0]', 1, '$[2][2]', 2);

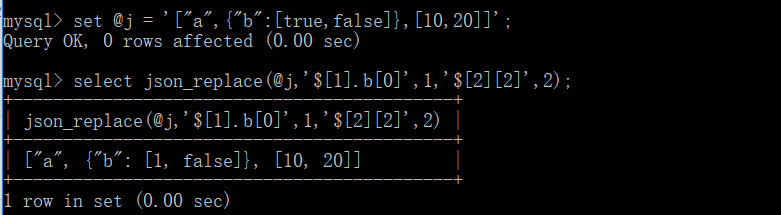


[JSON\_INSERT()](https://dev.mysql.com/doc/refman/5.7/en/json-modification-functions.html#function_json-insert) 增加新的值，但不会取代现有的值：

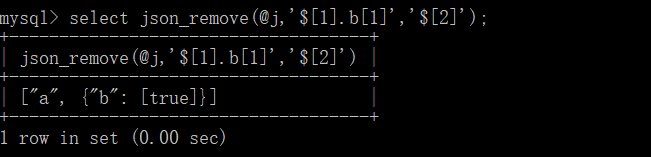
SELECT JSON\_INSERT(@j, '$[1].b[0]', 1, '$[2][2]', 2);



[JSON\_REPLACE()](https://dev.mysql.com/doc/refman/5.7/en/json-modification-functions.html#function_json-replace) 替换现有值并忽略新值：



JSON\_REMOVE()需要一个JSON文档和一个或多个指定要从文档中删除值的路径。返回值是原始文档减去文档中存在的路径选择的值：



5.JSON值的比较和排序

1.JSON值可以使用[=](https://dev.mysql.com/doc/refman/5.7/en/comparison-operators.html" \l "operator_equal)， [<](https://dev.mysql.com/doc/refman/5.7/en/comparison-operators.html#operator_less-than)， [<=](https://dev.mysql.com/doc/refman/5.7/en/comparison-operators.html#operator_less-than-or-equal)， [>](https://dev.mysql.com/doc/refman/5.7/en/comparison-operators.html#operator_greater-than)， [>=](https://dev.mysql.com/doc/refman/5.7/en/comparison-operators.html#operator_greater-than-or-equal)， [<>](https://dev.mysql.com/doc/refman/5.7/en/comparison-operators.html#operator_not-equal)， [!=](https://dev.mysql.com/doc/refman/5.7/en/comparison-operators.html#operator_not-equal)，和 [<=>](https://dev.mysql.com/doc/refman/5.7/en/comparison-operators.html#operator_equal-to) 进行比较。

2.JSON值尚不支持以下比较运算符和函数：

* [BETWEEN](https://dev.mysql.com/doc/refman/5.7/en/comparison-operators.html#operator_between)
* [IN()](https://dev.mysql.com/doc/refman/5.7/en/comparison-operators.html#function_in)
* [GREATEST()](https://dev.mysql.com/doc/refman/5.7/en/comparison-operators.html#function_greatest)
* [LEAST()](https://dev.mysql.com/doc/refman/5.7/en/comparison-operators.html#function_least)

这些比较运算符和函数的解决方法是将JSON值转换为本机MySQL数值或字符串数​​据类型，以便它们具有一致的非JSON标量类型。

\*JSON值的比较发生在两个层次上。第一级比较是基于比较值的JSON类型。如果类型不同，则比较结果仅由具有更高优先级的类型确定。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。

以下列表显示了从最高优先级到最低优先级的JSON类型 （类型名称是由[JSON\_TYPE()](https://dev.mysql.com/doc/refman/5.7/en/json-attribute-functions.html#function_json-type) 函数返回的类型）

BLOB

BIT

OPAQUE

DATETIME

TIME

DATE

BOOLEAN

ARRAY

OBJECT

STRING

INTEGER, DOUBLE

NULL

对于相同优先级的JSON值，比较规则是类型特定的（举一些比较特殊的例子）：

ARRAY

如果两个JSON数组的长度相同，并且数组中相应位置的值相等，则两个JSON数组相等。

如果数组不相等，则它们的顺序由第一个位置中存在差异的元素确定。首先排序具有较小值的数组。如果较短数组的所有值都等于较长数组中的相应值，则较短的数组首先排序

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

OBJECT

如果两个JSON对象具有相同的一组键，并且每个键在两个对象中都具有相同的值，则两个JSON对象相等。

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

两个不相等的对象的顺序是不确定的。

1. 在JSON和非JSON值之间转换

先通过cast（json）函数进行转化，将json类型数据转化为mysql可以比较的其他类型，然后再进行比较排序。 也可以通过cast（）函数将其他数据类型转化为json数据类型。

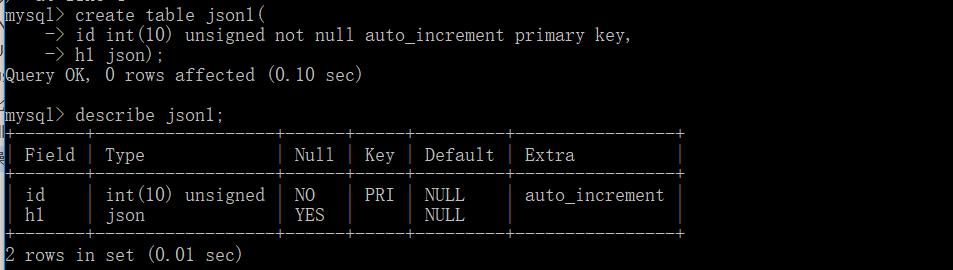
* ORDER BY GROUP BY对于JSON值则按照以下原则工作：标量JSON值的排序使用与前面的讨论相同的规则。
* 对于升序排序，SQL NULL在所有JSON值（包括JSON空文本）之前排序; 对于降序排序，NULL所有JSON值（包括JSON空文本）之后的SQL 顺序。
* 对于JSON值的排序键由[max\_sort\_length](https://dev.mysql.com/doc/refman/5.7/en/server-system-variables.html" \l "sysvar_max_sort_length)系统变量的值绑定 ，因此只有在第一个[max\_sort\_length](https://dev.mysql.com/doc/refman/5.7/en/server-system-variables.html" \l "sysvar_max_sort_length)字节比较为相等之后才有所不同 。
* 当前不支持排序非标量值，并发生警告。

对于排序，将JSON标量转换为其他本地MySQL类型可能会有所帮助。例如，如果一个名为 jdocJSON的对象包含一个由一个id键和一个非负值组成的成员，可以使用这个表达式按id 值排序：

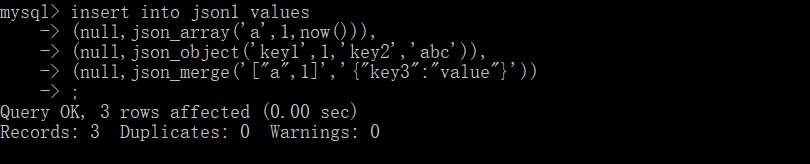
ORDER BY CAST(JSON\_EXTRACT(jdoc, '$.id') AS UNSIGNED)

8.JOSN数据操作截图

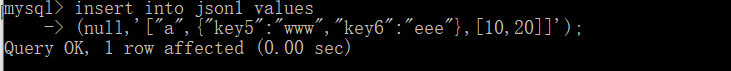
创建：



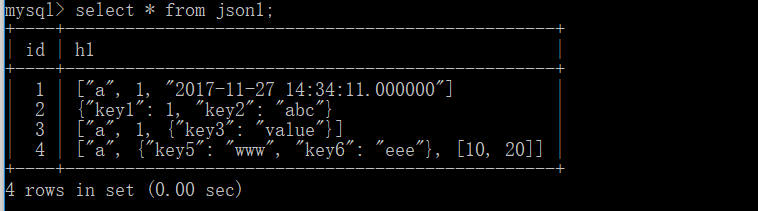
插入数据：json\_array(),json\_object(),json\_merge()



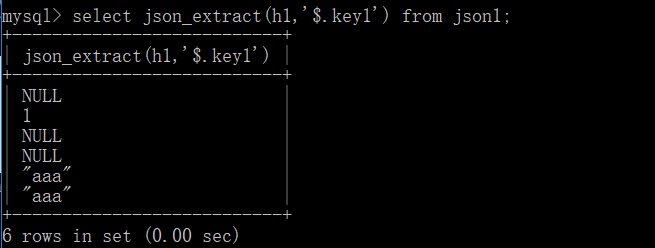
文本格式插入：

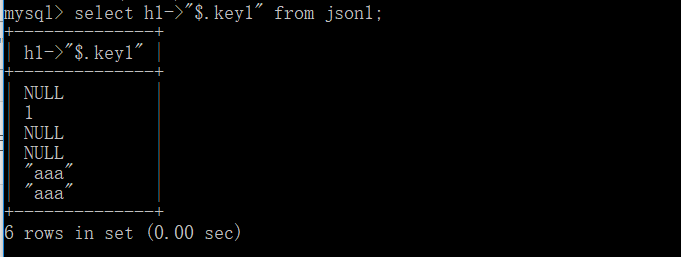


搜索json的值：



查找主键名为key1的表中所有的值：





查找json数组中的值：

