本文总结了如何在CockroachDB SQL中处理NULL值。获取更多信息可通过 [built-in SQL client](http://doc.cockroachchina.baidu.com/#quick-start/learn-cockroachdb-sql/use-the-built-in-sql-client/).

使用内置客户端时，使用单词NULL表示NULL值， 这将它们与包含空字符串(“”)的字符字段区分开来。

NULL的简单比较

一个值和NULL的比较结果都为NULL。 其余例子将在下一个问题中描述。

此行为与PostgreSQL以及所有其他主要RDBMS一致。

> **CREATE** **TABLE** t1(

a INT,

b INT,

c INT

);

> **INSERT** **INTO** t1 **VALUES**(1, 0, 0);

> **INSERT** **INTO** t1 **VALUES**(2, 0, 1);

> **INSERT** **INTO** t1 **VALUES**(3, 1, 0);

> **INSERT** **INTO** t1 **VALUES**(4, 1, 1);

> **INSERT** **INTO** t1 **VALUES**(5, NULL, 0);

> **INSERT** **INTO** t1 **VALUES**(6, NULL, 1);

> **INSERT** **INTO** t1 **VALUES**(7, NULL, NULL);

> **SELECT** \* **FROM** t1;

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

| a | b | c |

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

| 1 | 0 | 0 |

| 2 | 0 | 1 |

| 3 | 1 | 0 |

| 4 | 1 | 1 |

| 5 | NULL | 0 |

| 6 | NULL | 1 |

| 7 | NULL | NULL |

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

> **SELECT** \* **FROM** t1 **WHERE** b < 10;

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

| a | b | c |

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

| 1 | 0 | 0 |

| 2 | 0 | 1 |

| 3 | 1 | 0 |

| 4 | 1 | 1 |

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

> **SELECT** \* **FROM** t1 **WHERE** **NOT** b > 10;

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

| a | b | c |

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

| 1 | 0 | 0 |

| 2 | 0 | 1 |

| 3 | 1 | 0 |

| 4 | 1 | 1 |

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

> **SELECT** \* **FROM** t1 **WHERE** b < 10 **OR** c = 1;

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

| a | b | c |

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

| 1 | 0 | 0 |

| 2 | 0 | 1 |

| 3 | 1 | 0 |

| 4 | 1 | 1 |

| 6 | NULL | 1 |

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

> **SELECT** \* **FROM** t1 **WHERE** b < 10 **AND** c = 1;

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

| a | b | c |

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

| 2 | 0 | 1 |

| 4 | 1 | 1 |

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

> **SELECT** \* **FROM** t1 **WHERE** **NOT** (b < 10 **AND** c = 1);

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

| a | b | c |

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

| 1 | 0 | 0 |

| 3 | 1 | 0 |

| 5 | NULL | 0 |

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

> **SELECT** \* **FROM** t1 **WHERE** **NOT** (c = 1 **AND** b < 10);

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

| a | b | c |

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

| 1 | 0 | 0 |

| 3 | 1 | 0 |

| 5 | NULL | 0 |

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

检查NULL值时，请使用IS NULL”或“IS NOT NULL子句。

> **SELECT** \* **FROM** t1 **WHERE** b **IS** NULL **AND** c **IS** **NOT** NULL;

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

| a | b | c |

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

| 5 | NULL | 0 |

| 6 | NULL | 1 |

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

NULL和条件运算符

条件运算符(包括IF，COALESCE，IFNULL)仅根据条件操作数的值来评估一些操作数，因此它们的结果并不总是NULL，具体取决于给定的操作数。

例如，即使第二个操作数为NULL，COALESCE(1，NULL)也将始终返回1。

NULL和三元逻辑

AND，OR和IS实现三元逻辑，如下所示。

| **Expression** | **Result** |
| --- | --- |
| FALSE AND FALSE | FALSE |
| FALSE AND TRUE | FALSE |
| FALSE AND NULL | FALSE |
| TRUE AND FALSE | FALSE |
| TRUE AND TRUE | TRUE |
| TRUE AND NULL | NULL |
| NULL AND FALSE | FALSE |
| NULL AND TRUE | NULL |
| NULL AND NULL | NULL |

| **Expression** | **Result** |
| --- | --- |
| FALSE OR FALSE | FALSE |
| FALSE OR TRUE | TRUE |
| FALSE OR NULL | NULL |
| TRUE OR FALSE | TRUE |
| TRUE OR TRUE | TRUE |
| TRUE OR NULL | TRUE |
| NULL OR FALSE | NULL |
| NULL OR TRUE | TRUE |
| NULL OR NULL | NULL |

| **Expression** | **Result** |
| --- | --- |
| FALSE IS FALSE | TRUE |
| FALSE IS TRUE | FALSE |
| FALSE IS NULL | FALSE |
| TRUE IS FALSE | FALSE |
| TRUE IS TRUE | TRUE |
| TRUE IS NULL | FALSE |
| NULL IS FALSE | FALSE |
| NULL IS TRUE | FALSE |
| NULL IS NULL | TRUE |

NULL和算术

涉及NULL值的算术运算将产生NULL结果。

> **SELECT** a, b, c, b\*0, b\*c, b+c **FROM** t1;

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

| a | b | c | b \* 0 | b \* c | b + c |

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

| 1 | 0 | 0 | 0 | 0 | 0 |

| 2 | 0 | 1 | 0 | 0 | 1 |

| 3 | 1 | 0 | 0 | 0 | 1 |

| 4 | 1 | 1 | 0 | 1 | 2 |

| 5 | NULL | 0 | NULL | NULL | NULL |

| 6 | NULL | 1 | NULL | NULL | NULL |

| 7 | NULL | NULL | NULL | NULL | NULL |

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

NULL和聚合函数

聚合函数是对一组rows进行操作并返回单个值的函数。 此处重复示例数据，以便更容易理解结果。

> **SELECT** \* **FROM** t1;

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

| a | b | c |

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

| 1 | 0 | 0 |

| 2 | 0 | 1 |

| 3 | 1 | 0 |

| 4 | 1 | 1 |

| 5 | NULL | 0 |

| 6 | NULL | 1 |

| 7 | NULL | NULL |

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

> **SELECT** **COUNT**(\*), **COUNT**(b), **SUM**(b), **AVG**(b), **MIN**(b), **MAX**(b) **FROM** t1;

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

| COUNT(\*) | COUNT(b) | SUM(b) | AVG(b) | MIN(b) | MAX(b) |

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

| 7 | 4 | 2 | 0.5000000000000000 | 0 | 1 |

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

请注意以下事项：

* NULL值不包含在列的COUNT()中。 COUNT(\*)返回7，而COUNT(b)返回4。
* NULL值在MIN()或MAX()中不被视为高值或低值。
* AVG(b)返回SUM(b)/COUNT(b)，它与AVG(\*)不同，因为NULL值不在COUNT(b)中考虑。 有关更多详细信息，请参见[NULLs as Other Values](https://www.cockroachlabs.com/docs/stable/null-handling.html#nulls-as-other-values)

NULL作为不同的值

NULL值被认为与其他值不同，并且包含在列的不同值列表中。

> **SELECT** **DISTINCT** b **FROM** t1;

+------+

| b |

+------+

| 0 |

| 1 |

| NULL |

+------+

但是，计算不同值的数量会排除NULL，这与COUNT()函数一致。

> **SELECT** **COUNT**(**DISTINCT** b) **FROM** t1;

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

| count(DISTINCT b) |

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

| 2 |

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

NULL作为其他值

在某些情况下，你可能希望在算术或聚合函数计算中包含NULL值。 为此，在计算过程中使用IFNULL()函数将值替换为NULL。

例如，假设你要计算列b的平均值为b中所有数字的SUM()除以总行数，无论b的值是否为NULL。 在这种情况下，你将使用AVG(IFNULL(b，0)，其中IFNULL(b，0)在计算过程中将zero(0)的值替换为NULL。

> **SELECT** **COUNT**(\*), **COUNT**(b), **SUM**(b), **AVG**(b), **AVG**(**IFNULL**(b, 0)), **MIN**(b), **MAX**(b) **FROM** t1;

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

| COUNT(\*) | COUNT(b) | SUM(b) | AVG(b) | AVG(IFNULL(b, 0)) | MIN(b) | MAX(b) |

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

| 7 | 4 | 2 | 0.5000000000000000 | 0.2857142857142857 | 0 | 1 |

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

NULL和设置操作

NULL值被视为UNION设置操作的一部分

> **SELECT** b **FROM** t1 **UNION** **SELECT** b **FROM** t1;

+------+

| b |

+------+

| 0 |

| 1 |

| NULL |

+------+

NULL和排序

在对包含NULL值的列进行排序时，CockroachDB首先使用ASC排序NULL值，最后使用DESC排序。 这与PostgreSQL不同，首先用DESC，再用ASC排序NULL值

请注意，ORDER BY子句的NULLS FIRST和NULLS LAST选项未在CockroachDB中实现，因此你无法更改排序顺序中出现NULL值的位置。

> **SELECT** \* **FROM** t1 **ORDER** **BY** b **ASC**;

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

| a | b | c |

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

| 6 | NULL | 1 |

| 5 | NULL | 0 |

| 7 | NULL | NULL |

| 1 | 0 | 0 |

| 2 | 0 | 1 |

| 4 | 1 | 1 |

| 3 | 1 | 0 |

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

> **SELECT** \* **FROM** t1 **ORDER** **BY** b **DESC**;

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

| a | b | c |

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

| 4 | 1 | 1 |

| 3 | 1 | 0 |

| 2 | 0 | 1 |

| 1 | 0 | 0 |

| 7 | NULL | NULL |

| 6 | NULL | 1 |

| 5 | NULL | 0 |

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

NULL和Unique约束

NULL值不被认为是唯一的。 因此，如果表在一个或多个可选（可为空）的列上具有唯一约束，则可以在这些列中插入多个具有NULL值的行，如下面的示例所示。

> **CREATE** **TABLE** t2(a INT, b INT **UNIQUE**);

> **INSERT** **INTO** t2 **VALUES**(1, 1);

> **INSERT** **INTO** t2 **VALUES**(2, NULL);

> **INSERT** **INTO** t2 **VALUES**(3, NULL);

> **SELECT** \* **FROM** t2;

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

| a | b |

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

| 1 | 1 |

| 2 | NULL |

| 3 | NULL |

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

NULL和CHECK约束

一个结果为NULL的Check约束表达式被认为是通过的，允许使用像discount <price这样的简洁表达式而不必担心要添加OR discount IS NULL子句。 当需要非空验证时，通常的Not Null约束可以与Check约束一起使用。

> **CREATE** **TABLE** products (id **STRING** **PRIMARY** **KEY**, price INT **NOT** NULL **CHECK** (price > 0), discount INT, **CHECK** (discount <= price));

> **INSERT** **INTO** products (id, price) **VALUES** ('ncc-1701-d', 100);

> **INSERT** **INTO** products (id, price, discount) **VALUES** ('ncc-1701-a', 100, 50);

> **SELECT** \* **FROM** products;

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

| id | price | discount |

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

| ncc1701a | 100 | 50 |

| ncc1701d | 100 | NULL |

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

> **INSERT** **INTO** products (id, price) **VALUES** ('ncc-1701-b', -5);

failed to satisfy **CHECK** **constraint** (price > 0)

> **INSERT** **INTO** products (id, price, discount) **VALUES** ('ncc-1701-b', 100, 150);

failed to satisfy **CHECK** **constraint** (discount <= price)