Table of Contents

1. Testing Nulls

We have nulls in our data tables. A null represents a situation where a value is not known or is not applicable. We have seen that nulls propagate in arithmetic, and also in string concatenation with MySQL. Some client display nulls as blank space; other as the literal "(null)". We may want to have more control over how nulls are handled and displayed in our SQL code. The function considered here are:

- `COALESCE()`
- `NULLIF()`
- `IFNULL()`
- `ISNULL()`

A common situation is that we have nulls in a table and want to substitute a different value for the null in the result. We have a function Coalesce which lets us do that. One major concern is what is the appropriate value to use for a data value if the actual data value is not known. This is **a business rule decision** that is not the choice of the SQL coder. It is *not* always, or even often, appropriate to substitute a value of 0 if a numeric value is missing.

Suppose that we have an order for high def wide screen TVs and for some reason there is no price in the database for this item. If you substitute a value of 0- you have just given away a rather expensive product for free. It is more helpful to reject that data as being invalid and it might be even a better idea to not let an item be put into the products table for sale until we have a price.

On the other hand, if we are adding up test scores for student and a student did not take a test, then substituting a zero may make sense rather than returning a null for the test total score.

The point is that the person writing the code does not make the decision as to the proper way to handle a null. If the business knowledge expert does not specify how to handle this situation, ask!

Some of these demos use a small table z_numbers, with 5 integer columns, which has a lot of nulls in it. See demo file for code.

```
select  * from z_numbers;
+------+------+------+------+------+------+
| A    | B    | C    | D    | E    | F    |
+------+------+------+------+------+------+
|    1 |   10 |   10 |   50 |   90 |   45 |
|    2 |   15 |    5 | NULL |   10 |    0 |
|    3 | NULL | NULL |   50 |   50 |   -1 |
|    4 | NULL | NULL | NULL | NULL | NULL |
|    5 |    0 |    0 |    0 |    0 |    0 |
|    6 |   10 |   10 |   10 |   10 |   10 |
|    7 |  -10 |   10 |    0 |  210 |   85 |
|    8 |  -10 |   -1 |    0 | -210 |   85 |
|    9 |  200 |   -1 |    0 |   -1 |   85 |
|   10 |  200 |  200 |    0 |   -1 |   46 |
+------+------+------+------+------+------+
```

1.1. Coalesce

COALESCE accepts a series of values and returns the first non-null value. If all values are null, then the function returns null. The coalesce function is available in Oracle, T-SQL and MySQL.

One thing you can do to improve the portability of your code is to use techniques and functions that work on multiple dbms.

Demo 01: coalesce. The second expression looks at column B and if it is null, is evaluated as 9999. The third and fourth expression look at multiple columns; the arguments are in different order- the first non-null argument is returned.

```
select A
, COALESCE(B,9999) as B_Col
, COALESCE(B,C,D,E) as Coalesce_BCDE
, COALESCE(E,D,C,B) as Coalesce_EDCB
from z_numbers;
+------+-------+--------------+---------------+
| A    | B_Col | Coalesce_BCDE | Coalesce_EDCB |
+------+-------+--------------+---------------+
|    1 |    10 |            10 |            90 |
|    2 |    15 |            15 |            10 |
|    3 |  9999 |            50 |            50 |
|    4 |  9999 |          NULL |          NULL |
|    5 |     0 |             0 |             0 |
|    6 |    10 |            10 |            10 |
|    7 |   -10 |           -10 |          -210 |
|    8 |   -10 |           -10 |          -210 |
|    9 |   200 |           200 |           200 |
|   10 |   200 |           200 |             5 |
+------+-------+--------------+---------------+
```

Do not simply coalesce numeric columns to 0 or string columns to ZLS; this is generally incorrect. In the following query you cannot tell the difference in the result between a value of 0 that was inserted into the column directly in the Insert statement, and a 0 that comes from coalesce. Just because you do not know what a value is does not mean it is zero. (I do not yet know your midterm exam score, but I do not think you want to me to assume it is 0.)

```
select  A, COALESCE(B,0), COALESCE(D,0)
from z_numbers;
+------+---------------+---------------+
| A    | COALESCE(B,0) | COALESCE(D,0) |
+------+---------------+---------------+
|    1 |            10 |            50 |
|    2 |            15 |             0 |
|    3 |             0 |            50 |
|    4 |             0 |             0 |
|    5 |             0 |             0 |
|    6 |            10 |            10 |
|    7 |           -10 |             0 |
|    8 |           -10 |             0 |
|    9 |           200 |             0 |
|   10 |           200 |             0 |
+------+---------------+---------------+
```

1.2. IFNULL

IFNULL accepts two values. If the first argument is not null, then the first argument is returned. If the first argument is null then the second argument is returned.

The difference between Coalesce and IfNull is that Coalesce accepts more than two parameters; IfNull accepts only two parameters; IfNull is a MySQL function.

IFNull **(p_1, p_Null)** is executed as

```
if p_1 is null then
   return p_null
else
   return p_1
end if
```

Demo 02: IfNull. The expression for the column B-Null says that if column B is null , then use 87 instead. The expression for the column D_Null says that if column D is null , then use 0 instead. Notice that in the last column, you cannot tell the difference between values of 0 that come from the original data in column D and the values of O that come from the IfNull expression.

```
select  A
, B, IFNULL(B,87)  as B_Null
, D, IFNULL(D,0)   as D_Null
from z_numbers
;
+------+------+--------+------+--------+
| A    | B    | B_Null | D    | D_Null |
+------+------+--------+------+--------+
|    1 |   10 |     10 |   50 |     50 |
|    2 |   15 |     15 | NULL |      0 |
|    3 | NULL |     87 |   50 |     50 |
|    4 | NULL |     87 | NULL |      0 |
|    5 |    0 |      0 |    0 |      0 |
|    6 |   10 |     10 |   10 |     10 |
|    7 |  -10 |    -10 |    0 |      0 |
|    8 |  -10 |    -10 |    0 |      0 |
|    9 |  200 |    200 |    0 |      0 |
|   10 |  200 |    200 |    0 |      0 |
+------+------+--------+------+--------+
```

Demo 03: To create the coalesce example using `IfNull`, use the following which nests the `IfNull` function calls. Now doesn't coalesce look simpler.

```
select  A
, IFNULL(B, IFNULL(C, IFNULL(D, E) ) ) as Result
from z_numbers;
+------+--------+
| A    | Result |
+------+--------+
|    1 |     10 |
|    2 |     15 |
|    3 |     50 |
|    4 |   NULL |
|    5 |      0 |
|    6 |     10 |
|    7 |    -10 |
|    8 |    -10 |
|    9 |    200 |
|   10 |    200 |
+------+--------+
```

1.3. NULLIF

NULLIF (value1, value2) If value1 = value2, then NULLIF returns null, otherwise it returns value1.

NULLIF (p_1, p_2) is executed as

```
if  p_1 = p_2 then
   return null
else
   return p_1
end if
```

Demo 04: NullIF- if column B and C have the same value, this returns a null.

```
select  A, B, C, nullif(B,C)
from z_numbers
;
+------+------+------+-------------+
| A    | B    | C    | nullif(B,C) |
+------+------+------+-------------+
|    1 |   10 |   10 |        NULL |
|    2 |   15 |    5 |          15 |
|    3 | NULL | NULL |        NULL |
|    4 | NULL | NULL |        NULL |
|    5 |    0 |    0 |        NULL |
|    6 |   10 |   10 |        NULL |
|    7 |  -10 |   10 |         -10 |
|    8 |  -10 |   -1 |         -10 |
|    9 |  200 |   -1 |         200 |
|   10 |  200 |  200 |        NULL |
+------+------+------+-------------+
```

This can be used as a cleanup function. Suppose you inherit a table from someone who does not believe in allowing nulls and they used the flag value -1 to indicate that this value should have been a null. Since you want to use code that works with nulls, you can use the nullif function.

Demo 05: NullIF to clean up flags used instead of nulls This demo returns a null if the column passed as the first argument contain a -1; an actual null in that column also return a null.

```
select  A, F, NULLIF(F, -1)
from z_numbers
;
+------+------+---------------+
| A    | F    | NULLIF(F, -1) |
+------+------+---------------+
|    1 |   45 |            45 |
|    2 |    0 |             0 |
|    3 |   -1 |          NULL |
|    4 | NULL |          NULL |
|    5 |    0 |             0 |
|    6 |   10 |            10 |
|    7 |   85 |            85 |
|    8 |   85 |            85 |
|    9 |   85 |            85 |
|   10 |   46 |            46 |
+------+------+---------------+
```

In the following query, the third column, NULLIF(B, 200), returns all values of column B except for 200: The value 200 gets nulled out.

The fifth column, NULLIF(85, F), nulls out any original value of 85 in column F and returns the value 85 for the rest.

Demo 06: NullIF To null out values

```
select  A, B
, NULLIF(B, 200)
, D
, NULLIF(85, F)
from z_numbers;
+------+------+----------------+------+---------------+
| A    | B    | NULLIF(B, 200) | D    | NULLIF(85, F) |
+------+------+----------------+------+---------------+
|    1 |   10 |             10 |   50 |            85 |
|    2 |   15 |             15 | NULL |            85 |
|    3 | NULL |           NULL |   50 |            85 |
|    4 | NULL |           NULL | NULL |            85 |
|    5 |    0 |              0 |    0 |            85 |
|    6 |   10 |             10 |   10 |            85 |
|    7 |  -10 |            -10 |    0 |          NULL |
|    8 |  -10 |            -10 |    0 |          NULL |
|    9 |  200 |           NULL |    0 |          NULL |
|   10 |  200 |           NULL |    0 |            85 |
+------+------+----------------+------+---------------+
```

NULLIF(0, X-X) returns a null if X has a value. If X has a value, then X-X is 0 and the two arguments have the same value.

If X is null, then the function returns a 0. If X is null, the X –X is null and the two arguments do not have the same value.

Demo 07: NullIF to flip nulls and non-nulls

```
select  A, B
, NULLIF(0, B - B) as Flipped
from z_numbers;
+------+------+---------+
| A    | B    | Flipped |
+------+------+---------+
|    1 |   10 |    NULL |
|    2 |   15 |    NULL |
|    3 | NULL |       0 |
|    4 | NULL |       0 |
|    5 |    0 |    NULL |
|    6 |   10 |    NULL |
|    7 |  -10 |    NULL |
|    8 |  -10 |    NULL |
|    9 |  200 |    NULL |
|   10 |  200 |    NULL |
+------+------+---------+
```

Demo 08: Using nullif in an aggregate. This uses the average function - which we get to next week. Column D has some nulls and some zero values. Suppose that people who were entering data used a value of zero when they should have used a null.

```
select  D from z_numbers;
+------+
| D    |
+------+
|   50 |
| NULL |
|   50 |
| NULL |
|    0 |
|   10 |
|    0 |
```

```
|    0 |
|    0 |
|    0 |
+------+
10 rows in set (0.05 sec)

select  D
from z_numbers
where D is not null;
+------+
| D    |
+------+
|   50 |
|   50 |
|    0 |
|   10 |
|    0 |
|    0 |
|    0 |
|    0 |
+------+
8 rows in set (0.00 sec)
```

The average function skips any null values; but we want it to also skip any zero values. We can use nullif to say if the value is zero use a null instead.

The first column calculates the average including the zero values. The second column effectively says to ignore the zero values in the average.

```
select  avg(D), avg( nullif(D,0))
from z_numbers;
+---------+-------------------+
| avg(D)  | avg( nullif(D,0)) |
+---------+-------------------+
| 13.7500 |           36.6667 |
+---------+-------------------+
```

NULLIF is one of those functions that you think you will never need-until some day when you find yourself writing code to handle these situation.

1.4. ISNULL

ISNULL accepts one value. The function returns 1 if the argument is null and 0 if it is not

Demo 09: Displaying the value of IsNull

```
select  A, B, IsNull(B)
from z_numbers;
+------+------+-----------+
| A    | B    | IsNull(B) |
+------+------+-----------+
|    1 |   10 |         0 |
|    2 |   15 |         0 |
|    3 | NULL |         1 |
|    4 | NULL |         1 |
|    5 |    0 |         0 |
|    6 |   10 |         0 |
|    7 |  -10 |         0 |
|    8 |  -10 |         0 |
|    9 |  200 |         0 |
|   10 |  200 |         0 |
+------+------+-----------+
```

Demo 10: ISNULL

```
select  A, B
from z_numbers
where IsNull(B);
+------+------+
| A    | B    |
+------+------+
|    3 | NULL |
|    4 | NULL |
+------+------+
```

You get the same result with the IS Null test.

```
select  A, B
from z_numbers
where B is null;
```

When you use the IsNull function in the Where clause it returns a value of 0 or 1. MySQL uses the convention that in a test the value 0 is equivalent to False and a non-zero value is equivalent to True.

You could write the following queries- but don't. But you need to be aware of this convention when you are looking at mysql code.

Demo 11: Using an Integer as a Truth value

```
select  A, B
from z_numbers
where 56;

select  A, B
from z_numbers
where 0;
```

1.5. Testing with a numeric filter

In the animals table in a_vets we have some animals with names, some animals with a null for a name. I am going to insert one animal that has zls (zero-length-string) for a name.

```
Insert into a_vets.vt_animals values (40001, 'cat','', '2014-06-30', 411)
```

Demo 12: First just display the data from the animals table. I have 33 rows in the result set

```
select  an_id, an_name, an_type
from a_vets.vt_animals
order by an_name;
+-------+-----------------+-----------+
| an_id | an_name         | an_type   |
+-------+-----------------+-----------+
| 11025 | NULL            | bird      |
| 11029 | NULL            | bird      |
| 21007 | NULL            | snake     |
| 40001 |                 | cat       |
| 17025 | 25              | lizard    |
| 17027 | 3P#_25          | lizard    |
| 17026 | 3P#_26          | lizard    |
| 21314 | Adalwine        | cat       |
. . . rows omitted
| 21318 | Waldrom         | cat       |
| 21001 | Yoggie          | hedgehog  |
+-------+-----------------+-----------+
32 rows in set (0.00 sec)
```

Demo 13: This version uses the length function to get the number of characters in the animal name attribute. The function is used in the filter and each animal with a name that has at least one character is returned. I got 29 rows in the result set- 4 rows - the three rows with a null an_name and the one row with a ZLS for the an_name- are not returned.
I think this is poor style but it is common in MySQL so you need to recognize it.

```
select an_id, an_name
from a_vets.vt_animals
where length(an_name)
order by an_name;
+-------+-----------------+
| an_id | an_name         |
+-------+-----------------+
| 17025 | 25              |
| 17027 | 3P#_25          |
| 17026 | 3P#_26          |
| 21314 | Adalwine        |
. . . rows omitted
| 21318 | Waldrom         |
| 21001 | Yoggie          |
+-------+-----------------+
29 rows in set (0.00 sec)
```

Demo 14: What does the length function return for a Null and what does it return for a ZLS?

```
select an_id, an_name, length(an_name)
from a_vets.vt_animals
where an_id in (11025,11029,21007,40001)
order by an_name;
+-------+---------+-----------------+
| an_id | an_name | length(an_name) |
+-------+---------+-----------------+
| 11025 | NULL    |            NULL |
| 11029 | NULL    |            NULL |
| 21007 | NULL    |            NULL |
| 40001 |         |               0 |
+-------+---------+-----------------+
```

So why did the filter above- Where length(an_name) - filter out the rows with either a null or a zls for the an_name? In MySQL, the value 0 is treated as False if it is used as a test in a Where clause- that is why an_id 40001 is not returned. The other three rows have a null as the value of the test in the Where clause and that is treated as UnKnown. The Where clause only returns rows with a value of True.

Demo 15: If we want to get all of the missing or zls name we can filter for these

```
select  an_id, an_name
from a_vets.vt_animals
where an_name is null or NOT length(an_name)
order by an_name;
+-------+---------+
| an_id | an_name |
+-------+---------+
| 11025 | NULL    |
| 11029 | NULL    |
| 21007 | NULL    |
| 40001 |         |
+-------+---------+
```

If you added that new row you can remove it now.

```
Delete from a_vets.vt_animals where an_id = 40001;
```