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, but not in string concatenation with Oracle. Some clients 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()
- NVL()
- NULLIF()
- NVL2()

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_tst_numbers, which has integer columns and which has a lot of nulls in it: I am inserting a <n> for the nulls in **this** display. Code for this table is in the demo.

A	B	C	D	E	F
1	10	10	50	90	45
2	15	5	<n>	10	0
3	<n>	<n>	50	50	-1
4	<n>	<n>	<n>	<n>	<n>
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. Coalesce can cause an error if you use arguments of different data types and they cannot be cast to the same type. The coalesce function is an ansi standard function 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. You will see the NVL function, which is an Oracle specific function, used in a lot of older code.

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_tst_numbers
;
```

A	B_COL	COALESCE_BCDE	COALESCE_EDCB
1	10	10	90
2	15	15	10
3	9999	50	50
4	9999		
5	0	0	0
6	10	10	10
7	-10	-10	-210
8	-10	-10	-210
9	200	200	-1
10	200	200	-1

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) as BCol
, COALESCE(D,0) as DCol
from z_tst_numbers
;
```

A	BCOL	DCOL
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. NVL

NVL(exp, sub_value) If the expression is null, then NVL returns sub_value; otherwise it returns the value of the expression.

The difference between Coalesce and NVL is that Coalesce accepts more than two parameters; Nvl accepts only two parameters; NVL is an Oracle proprietary function.

NVL (p_1, p_Null) is executed as

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

Demo 02: NVL. The expression for the column B_Null says that if column B is null use 87 instead. The expression for the column D_Null says that if column D is null use 0 instead. Notice that in this column you cannot tell the difference between the values of 0 that come from the original data in column D and the values of 0 that come from the NVL expression

```
select A
, B, NVL(B,87) as B_Null
, D, NVL(D,0) as D_Null
, Coalesce (D,0) as D_null_Coalesce
from z_tst_numbers;
```

A	B	B_NULL	D	D_NULL	D_NULL_COALESCE
1	10	10	50	50	50
2	15	15		0	0
3		87	50	50	50
4		87		0	0
5	0	0	0	0	0
6	10	10	10	10	10
7	-10	-10	0	0	0
8	-10	-10	0	0	0
9	200	200	0	0	0
10	200	200	0	0	0

Demo 03: This is how you write the query demo 01 using nvl and nested functions

```
select
  A
, NVL(B,9999) as B_Col
, NVL(B, NVL(C, NVL(D,E))) as NVL_BCDE
, NVL(E, NVL(D, NVL(C,B))) as NVL_EDCB
from z_tst_numbers
;
```

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 nullif (B,C) returns a null if B and C have the same value

```
select A, B, C, nullif(B,C)
from z_tst_numbers;
```

A	B	C	NULLIF(B,C)
1	10	10	
2	15	5	15
3			
4			
5	0	0	
6	10	10	
7	-10	10	-10
8	-10	-1	-10
9	200	-1	200
10	200	200	

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

```
select A, C, NULLIF(C, -1)
from z_tst_numbers;
```

A	C	NULLIF(C,-1)
1	10	10
2	5	5
3		
4		
5	0	0
6	10	10
7	10	10
8	-1	
9	-1	
10	200	200

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 of the rows.

Demo 06: NULLIF to null out values

```
select A
, B, NULLIF(B, 200)
, F, NULLIF(85, F)
from z_tst_numbers;
```

A	B	NULLIF(B,200)	F	NULLIF(85,F)
1	10	10	45	85
2	15	15	0	85
3			-1	85
4				85
5	0	0	0	85
6	10	10	10	85
7	-10	-10	85	
8	-10	-10	85	
9	200		85	
10	200		46	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)
from z_tst_numbers
;
```

```
         A          B NULLIF(0,B-B)
---------- ---------- -------------
         1         10
         2         15
         3                        0
         4                        0
         5          0
         6         10
         7        -10
         8        -10
         9        200
        10        200
```

Demo 08: Using Null if to avoid divide by zero error

```
-- this is a regular division that has no surprises
variable num number;
exec  :num := 5;
select 20/:num from dual
;
```

```
  20/:NUM
----------
         4
```

```
-- if we try to divide by zero we get an error message
exec  :num := 0;
select 20/:num from dual;
```

```
ERROR at line 1:
ORA-01476: divisor is equal to zero
```

We can use nullif to return anull divisor instead of dividing by zero. Of course you need to decide if a null makes sense in terms of what you are trying to do. We now have one expression that executes with a zero or a non-zero value.

```
exec  :num := 5;
select 20 / (nullif(:num,0)) as Quotient
from dual;
```

```
  QUOTIENT
----------
         4
```

```
exec  :num := 0;
/
```

```
  QUOTIENT
----------
           -- this is the null return
```

Demo 09: Using nullif in an aggregate. This uses the average function - which we get to in another unit. 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_tst_numbers;
```

```
          D
-----------
         50

         50

          0
         10
          0
          0
          0
          0

10 rows selected.
```

```
select D from z_tst_numbers
where D is not null;
```

```
          D
-----------
         50
         50
          0
         10
          0
          0
          0
          0

8 rows selected.
```

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_tst_numbers;
```

```
      AVG(D) AVG(NULLIF(D,0))
---------- ----------------
     13.75       36.6666667
```

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. NVL2

The NVL2 function returns an If-then test. It looks at the first value and then returns one of two values depending on whether or not the first parameter is null. If you have done programming, this is the logic for nvl2.

NVL2(exp, sub1_value, sub2_value) If the expression is not null, then NVL2 returns sub1_value, otherwise it returns sub2_value.

NVL2 (p_1, p_NotNull, p_Null) is executed as

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

Demo 10: NVL2

```
select A
, B, NVL2(B, 300, 600)
from z_tst_numbers;
```

A	B	NVL2(B,300,600)
1	10	300
2	15	300
3		600
4		600
5	0	300
6	10	300
7	-10	300
8	-10	300
9	200	300
10	200	300