Table of Contents

1. GREATEST and LEAST 1
1.1. Examples using the demo tables 2
2. IF 3

1. GREATEST and LEAST

GREATEST and LEAST return the largest and smallest value from the list of arguments. Notice what happens with nulls. If there is a null in the list then the functions treat this as an unknown value and therefore the function cannot "know" which value in the list is the largest.

Warning: in a future unit we discuss the functions Max and Min which sound very much like Greatest and Least. They are NOT the same. Greatest and Least. are row functions- they work on multiple columns in a single row of data. You can use them with literals.

Demo 01:

```
select A
, GREATEST(B,C,D,E,F)
, LEAST(B, C, D, E,F)
from a_testbed.z_tst_numbers;
+------+--------------------+---------------------+
| A    | GREATEST(B,C,D,E,F) | LEAST(B, C, D, E,F) |
+------+--------------------+---------------------+
|    1 |                 90 |                  10 |
|    2 |               NULL |                NULL |
|    3 |               NULL |                NULL |
|    4 |               NULL |                NULL |
|    5 |                  0 |                   0 |
|    6 |                 10 |                  10 |
|    7 |                210 |                 -10 |
|    8 |                 85 |                -210 |
|    9 |                200 |                  -1 |
|   10 |                200 |                  -1 |
+------+--------------------+---------------------+
```

Demo 02:

```
select greatest(4, 45.78, 9), least(4, 45.78, 9);
+----------------------+---------------------+
| greatest(4, 45.78, 9) | least (4, 45.78, 9) |
+----------------------+---------------------+
|                45.78 |                4.00 |
+----------------------+---------------------+
```

Data type Issues: A function returns a single value. GREATEST (list), LEAST (list) returns the largest, smallest in the list. Avoid mixing incompatible data types.

Demo 03: The string values are treated as numeric and evaluated to 0

```
select greatest( 'flea', 4, 45.78, 9, 'elephant');
+-------------------------------------------+
| greatest( 'flea', 4, 45.78, 9, 'elephant') |
+-------------------------------------------+
| 45.78                                     |
+-------------------------------------------+
1 row in set, 2 warnings (0.00 sec)

Warning (Code 1292): Truncated incorrect DOUBLE value: 'flea'
Warning (Code 1292): Truncated incorrect DOUBLE value: 'elephant'
```

MySQL's general approach is to treat type issues as something to be compensated for more than as an error. You need to develop the habit of paying attention to the warning messages.

Demo 04:

```
select least( 'flea', 4, 45.78, 9, 'elephant');
+-----------------------------------------+
| least( 'flea', 4, 45.78, 9, 'elephant') |
+-----------------------------------------+
| 0                                       |
+-----------------------------------------+
1 row in set, 2 warnings (0.00 sec)

Warning (Code 1292): Truncated incorrect DOUBLE value: 'flea'
Warning (Code 1292): Truncated incorrect DOUBLE value: 'elephant'
```

Demo 05: Data types matter; as numbers 25 is smaller than 125; as string '125' sorts before '25' and is considered least.

```
select least(25, 125) as Numbers, least('25', '125') as Strings;
+---------+---------+
| Numbers | Strings |
+---------+---------+
|      25 | 125     |
+---------+---------+
```

1.1. Examples using the demo tables

Demo 06: This query returns the largest of the two columns quoted_price and prod_list_price

```
select order_id
, prod_id
, quoted_price
, prod_list_price
, GREATEST(quoted_price, prod_list_price)  as HigherPrice
from orderEntry.orderDetails
join product.products using (prod_id)
where prod_id in (1000, 1010);
+----------+---------+-------------+-----------------+-------------+
| order_id | prod_id | quoted_price | prod_list_price | HigherPrice |
+----------+---------+-------------+-----------------+-------------+
|      115 |    1000 |       100.00 |          125.00 |      125.00 |
|      303 |    1000 |       125.00 |          125.00 |      125.00 |
|      313 |    1000 |       125.00 |          125.00 |      125.00 |
|      608 |    1000 |       100.00 |          125.00 |      125.00 |
|     3808 |    1000 |       100.00 |          125.00 |      125.00 |
|      105 |    1010 |       150.00 |          150.00 |      150.00 |
|      390 |    1010 |       175.00 |          150.00 |      175.00 |
|      395 |    1010 |       195.00 |          150.00 |      195.00 |
. . .
```

Demo 07: The Greatest function just returns a number. You probably want to know which is bigger; that uses a Case. (In our tables these price columns are not null, but that is not true for all tables. So you always have to think about those nulls.)

```
select order_id
, prod_id
, quoted_price as "quoted"
, prod_list_price  as "List"
```

```
, GREATEST(quoted_price, prod_list_price)  as "higher"
, case  when quoted_price = prod_list_price then 'same price'
        when quoted_price > prod_list_price then 'quoted is higher'
        when quoted_price < prod_list_price then 'list is higher'
        else 'one or more is null'
        end  "PriceComparison"
from orderEntry.orderDetails
join product.products using (prod_id)
where prod_id in (1000, 1010)
;
+--------+---------+--------+--------+--------+-----------------+
| order_id | prod_id | quoted | List   | higher | PriceComparison  |
+--------+---------+--------+--------+--------+-----------------+
|    115 |    1000 | 100.00 | 125.00 | 125.00 | list is higher   |
|    303 |    1000 | 125.00 | 125.00 | 125.00 | same price       |
|    313 |    1000 | 125.00 | 125.00 | 125.00 | same price       |
|    608 |    1000 | 100.00 | 125.00 | 125.00 | list is higher   |
|   3808 |    1000 | 100.00 | 125.00 | 125.00 | list is higher   |
|    105 |    1010 | 150.00 | 150.00 | 150.00 | same price       |
|    390 |    1010 | 175.00 | 150.00 | 175.00 | quoted is higher |
|    395 |    1010 | 195.00 | 150.00 | 195.00 | quoted is higher |
|    405 |    1010 | 150.00 | 150.00 | 150.00 | same price       |
|    528 |    1010 | 150.00 | 150.00 | 150.00 | same price       |
. . .
```

2. IF

The IF function takes three arguments; the first should be an expression that has a true/false value. If its value is true then the return value is the value of argument 2; if the value of the first expression is not true then the return value is the value of argument 3.

You can write this logic with a case to have more portable code and code that is easier to understand.

Demo 08:

```
select if(curdate() > '1888-08-08', 'passed the test', 'this is really old');
+-------------------------------------------------------------------------+
| if(curdate() > '1888-08-08', 'passed the test', 'this is really old') |
+-------------------------------------------------------------------------+
| passed the test                                                         |
+-------------------------------------------------------------------------+

select if(month(curdate()) in (6,7,8), 'Summer!', 'Not summer');
+-----------------------------------------------------------+
| if(month(curdate()) in (6,7,8), 'Summer!', 'Not summer') |
+-----------------------------------------------------------+
| Not Summer                                                |
+-----------------------------------------------------------+

select if( 5 > null, 'passed the test', 'nulls make unknown values');
+-------------------------------------------------------------+
| if(5 > null, 'passed the test', 'nulls make unknown values') |
+-------------------------------------------------------------+
| nulls make unknown values                                   |
+-------------------------------------------------------------+

Using case

select case when month(curdate()) in (6,7,8) then 'Summer!'
            else  'Not summer'
            end  as Message;
```

Demo 09: We want to give customers a 5% savings for each pet supply item or sporting goods item. As a first step we will determine the percent to apply to the price.

```
select catg_id, prod_id, prod_list_price
, if(catg_id IN('PET','SPG'), 0.95, 1) as "Price Multiplier"
from product.products products
order by prod_id;
```

Selected rows

```
+---------+---------+----------------+------------------+
| catg_id | prod_id | prod_list_price | Price Multiplier |
+---------+---------+----------------+------------------+
| HW      |    1000 |          125.00 |                1 |
| SPG     |    1010 |          150.00 |             0.95 |
| SPG     |    1050 |          269.95 |             0.95 |
| SPG     |    1060 |          255.95 |             0.95 |
| APL     |    1125 |          500.00 |                1 |
| APL     |    1130 |          149.99 |                1 |
| PET     |    1140 |           14.99 |             0.95 |
| PET     |    4577 |           29.95 |             0.95 |
```

Demo 10: The if test could be nested for another test. Suppose that APL were to get a 10% discount.

```
select catg_id, prod_id, prod_list_price
, if(catg_id IN('PET','SPG'), 0.95, if(catg_id IN('APL'), 0.90, 1))
     as "Price Multiplier"
from product.products products
order by prod_id;
```

Selected rows

```
+---------+---------+----------------+------------------+
| catg_id | prod_id | prod_list_price | Price Multiplier |
+---------+---------+----------------+------------------+
| HW      |    1000 |          125.00 |                1 |
| SPG     |    1010 |          150.00 |             0.95 |
| SPG     |    1050 |          269.95 |             0.95 |
| SPG     |    1060 |          255.95 |             0.95 |
| APL     |    1125 |          500.00 |             0.90 |
| APL     |    1130 |          149.99 |             0.90 |
| PET     |    1140 |           14.99 |             0.95 |
| PET     |    4577 |           29.95 |             0.95 |
```

Nesting IF tests like this quickly becomes hard to read and is error prone. For anything other than a simple single test, use a case expression. Also case expressions are standard SQL supported by most dbms; the If function is not widely supported.