Table of Contents

1. Validating arguments 1
1.1. Validation rules 2
1.2. Another method for dealing with nulls 3
2. Interacting with Tables (Optional) 4
2.1. Using a Query to Place a Value into a Variable 4

The previous examples of functions assumed that the input arguments were reasonable values. You can do some simple validation of the parameters to handle the situation where a user supplied an argument that does not work for your function.

1. Validating arguments

What can we do in the function to check some of our arguments? Let's go back to the function new_salary_1

Demo 01: NewSalary_1 Everyone is going to get a $300 raise

```
Drop function if exists newsalary_1#

create function newsalary_1 (
   in_salary   decimal (9,2))
   returns decimal (10,2)
begin
  declare  c_increase_amount decimal (9,2) default 300;
  declare  v_new_salary      decimal (10,2);

  set v_new_salary  := in_salary + c_increase_amount;
   return v_new_salary ;
end;
#
```

This gives us a null return for a null argument and accepts a negative value for salary.

```
Select newsalary_1 (4500) as Result #
+---------+
| Result  |
+---------+
| 4800.00 |
+---------+


Select newsalary_1 (null) as Result #
+--------+
| Result |
+--------+
|   NULL |
+--------+


Select newsalary_1 (-4500) as Result #
+----------+
| Result   |
+----------+
| -4200.00 |
+----------+
```

1.1. Validation rules

Suppose we were told to follow the following rules for validation.

If the input is null, then the return should be the minimum increase (in this example $300.00)

If the input is negative, then the return should be 0

We can handle this logic with If tests

Demo 02:

```
Drop function if exists newsalary_1_V2#

create function newsalary_1_V2 (
   in_salary   decimal (9,2))
   returns decimal (10,2)
begin
   declare  c_increase_amount decimal (10,2) default 300;
   declare  v_new_salary      decimal (10,2);


   if in_salary is null then
      set v_new_salary  := c_increase_amount;
   elseif in_salary < 0 then
      set v_new_salary  := 0;
   else
      set v_new_salary  := in_salary + c_increase_amount;
   end if;

   return v_new_salary ;
end;
#
```

Demo 03: Testing

```
select
  Salary
, AnticipatedValue
, newsalary_1_V2 (salary) as "new salary"
, AnticipatedValue - NewSalary_1_v2(salary) as "problem"
from (
    select 10000 as Salary
         , 10300 as AnticipatedValue
  union all
    select null,   300
  union all
    select -500,  0
) as tstTbl#


Select
   newsalary_1_V2 (4500) as Result_1
,  newsalary_1_V2 (null) as Result_2
,  newsalary_1_V2 (-35)  as Result3#

+----------+----------+--------+
| Result_1 | Result_2 | Result |
+----------+----------+--------+
|  4800.00 |   300.00 |   0.00 |
+----------+----------+--------+
```

1.2. Another method for dealing with nulls

We want a function named **FutureDate** that has four parameters. The first is a datetime value, the 2nd, 3rd, and 4th are integer arguments that are expected to be the number of years, months, and days. The function will calculate and return a date that adds the indicated number of years, months and days. The function will use only the year-month-day components of the first parameter- ignoring any time components and returns a date type. If the first argument is null, then return a null. The other arguments can be positive or negative numbers; if they are null then do not add anything for that component. For example:

FutureDate(mydate, 2, 3, 4) will add 2 years, 3 months and 4 days to mydate

FutureDate(mydate, 2, null, 44) will add 2 years, no months and 44 days to mydate

Hopefully you are thinking of the Data_dd function which can add years, month and days to a date value.

We have two other considerations. (1) We need to ignore the time components. The easiest way to do that is assign the datetime parameters to a local date variable. (2) we need to handle any possible null parameter for the year, month, day. If the parameter for year is null, then we want to add 0 years. You might be thinking of If tests, but there is an easier way- use the coalesce function. set p_yr :=coalesce(p_yr, 0);

But what if the first parameter for the date is null; the Date_Add function will handle that and return a null if the date value passed to it is null; your code does not need to handle that.

Demo 04: FutureDate version 1

```
drop function if exists a_testbed.FutureDate#

create  function a_testbed.FutureDate (
  p_date  datetime
, p_yr  int
, p_mn  int
, p_day  int)
  returns date
BEGIN
    declare return_date date ;
    set return_date := p_date;

    set p_yr  :=coalesce(p_yr,  0);
    set p_mn  :=coalesce(p_mn,  0);
    set p_day :=coalesce(p_day, 0);

    set return_date := date_add(return_date, interval p_yr year);
    set return_date := date_add(return_date, interval p_mn month);
    set return_date := date_add(return_date, interval p_day day);

    return return_date;
    end;
#
```

Demo 05: If you are very comfortable with functions you can do this

```
drop function if exists a_testbed.FutureDate2#

create function a_testbed.FutureDate2 (
  p_date  datetime
, p_y  int
, p_m  int
, p_d  int)
```

```
    returns date
  BEGIN
     return p_date +
       interval coalesce(p_y, 0) year +
       interval coalesce(p_m, 0) month +
       interval coalesce(p_d, 0) day;
  END;
  #
```

2. Interacting with Tables (Optional)

2.1. Using a Query to Place a Value into a Variable

The SQL statement in the following routine will count the number of rows in the table. There is a new clause (`INTO variable`) that places the Count(*) value into a local MySQL variable. The Select statement can refer to MySQL variables in both the `INTO` clause and the `WHERE` clause.

These are not robust functions- they simply are designed to show you that we can use MySQL to retrieve data from a table.

Demo 06: Getting a single value from the query to place into a local MySQL variable and return it.

```
Drop function if exists DeptEmployeeCount #

Create function DeptEmployeeCount  (
   p_dept_id int)
   returns int
begin
    declare  v_row_count   int ;
    select count(*)
    into v_row_count
    from a_emp.employees
    where Dept_id = p_dept_id;

    return v_row_count;
end;
#
```

Demo 07: Testing this

```
select DeptEmployeeCount (80)
#
+-------------------------+
| DeptEmployeeCount (80) |
+-------------------------+
|                       3 |
+-------------------------+


select DeptEmployeeCount (1234)
#
+---------------------------+
| DeptEmployeeCount (1234) |
+---------------------------+
|                         0 |
+---------------------------+
```

Demo 08: This is a function that uses a join between two tables in the select query.

```
Drop function if exists empjobtitle#

create function empjobtitle
   (in_emp_id int)
   returns varchar(100)
begin
   declare  v_job_title  varchar(35);
   declare  v_message    varchar(100);

   select job_title
   into v_job_title
   from  a_emp.employees
   join a_emp.jobs using (job_id)
   where  emp_id = in_emp_id;

   set v_message := concat('employee ' , in_emp_id ,' has job ' , v_job_title);
   return v_message;
end;
#
```

Demo 09: Test this with a value that matches an employee

```
select  EmpJobTitle(6023)
#
+--------------------------+
| EmpJobTitle(103)         |
+--------------------------+
| employee 103 has job DBA |
+--------------------------+
```

Demo 10: Test this with a value that does not match an employee

```
select  EmpJobTitle(63) #
+-----------------+
| EmpJobTitle(63) |
+-----------------+
| NULL            |
+-----------------+
```

In this case the select query that gets a value for the job_title from the table cannot find a matching row and returns a Null. That null is assigned to the variable v_job_title and causes the concatenated string to be null.