Table of Contents

1. What is an Intrinsic Function

One of the strengths of a dbms is the number and variety of single-row supplied functions. Functions do tasks such as rounding numbers to the desired accuracy, locating a string pattern within a target string, and dealing with nulls. A function returns a value and we can use that value in a Select clause, in a Where clause- and later we will use them in action statements.

One of the things you have to consider is that these functions are MySQL supplied functions; these are often called intrinsic functions since they are built into the dbms.. Other dbms will have similar functions but they might not use the same function name and the same arguments. When you start using supplied functions, your code is less portable. MySQL has situations where they have more than one function that does the same thing; sometimes MySQL has a function that looks like an Oracle function and another function that looks like it came from T-SQL to accomplish the same goal.

The value returned by a function can be passed to another function; this is called nesting functions and we will see examples of this as we work through various functions.

To use a function you need to know the name of the function and the type of arguments it needs. You also need to know the data type of the value returned by the function. You also need to consider (or test) what happens with a function if you pass it a null as one of the arguments, or pass it a value that is not appropriate for the function.

A function does not change the value of the arguments that were passed to it. For example, if you use the round function to round a numeric column in a Select query, the function does not change the data stored in the table; it simply returns the rounded version of the numbers.

These functions are referred to as **single-row functions.** They take one or more arguments, typically from one row in a table, and return a value. When the function is used in a Select statement, it returns one value for each row in the table. We will later discuss multi-row functions which take a group of rows and return a single value for the group.

We will not cover every function and I do not expect you to remember them all and all of their variations. But you should be able to quickly find and use functions from these documents and your book in doing assignments and during exams.

We will categorize the functions in the areas of

- functions that work with numbers
- functions that work with strings
- operators that work with regular expressions (these are not functions but it makes sense to discuss them here)
- functions that convert data from one format to another
- functions that do some sort of logic
- functions that work with temporal values

Some functions are hard to classify and may be discussed in more than one category.

2. Demonstrating Functions

Functions are best explained with examples. For many of the functions, I will show you examples of the use of the function with literal values- such as the Upper function which accepts a string and returns the string with all letters converted to upper case letters. You would not normally pass a literal string to this function but it is easier to show you examples using this technique.

You can test the function examples by running them with a select. This demonstrates the Upper function. The mysql client will display the expression as the column header if there is no column alias.

```
select UPPER( '       This is MY sTrInG') ;
+-----------------------------------+
| UPPER( '       This is MY sTrInG') |
+-----------------------------------+
|        THIS IS MY STRING          |
+-----------------------------------+
```

A more typical use is to change the display with a select that gets data from a table. Note the column aliases.

```
select an_id, upper(an_type), upper(an_type) as an_type
from vt_animals
order by an_id limit 5;
+-------+----------------+---------+
| an_id | upper(an_type) | an_type |
+-------+----------------+---------+
| 10002 | CAT            | CAT     |
| 11015 | SNAKE          | SNAKE   |
| 11025 | BIRD           | BIRD    |
| 11029 | BIRD           | BIRD    |
| 12035 | BIRD           | BIRD    |
+-------+----------------+---------+
```

2.1. Nesting Functions

This is an example of a nested function. This is an expression that uses two functions; one function returns a value that is used as an argument to the second function. The value that is returned by the upper function is passed to the Ltrim function which removes any spaces at the Left edge of the string.

```
Select LTRIM(UPPER( '       This is MY sTrInG')) ;
+----------------------------------------+
| LTRIM(UPPER( '       This is MY sTrInG')) |
+----------------------------------------+
| THIS IS MY STRING                      |
+----------------------------------------+
```

When you work with functions, as for the assignments, you need to work methodically. This example will make more sense after you work through the other documents in this unit. So make a note to come back to this.

3. Developing an expression using functions

Suppose, for some reason, we want to take the product descriptions in the Altgeld mart products table, throw away all of the spaces in the description- including the spaces between words, and then get the first 10 characters in the description in uppercase letters. You could use this expression in a select statement that you run against that products table. Before you start to think about the expressions, think about possible description that you should test and what the results should be:

- A description that is longer than 10 characters, after removing blanks:
  `'Bird cage- simple; wire frame two feeder trays'`
  Should return: `'BIRDCAGE-S'`
- A description that is shorter than 10 characters, after removing blanks:
  `'Red Mixer'`
  Should return: `'REDMIXER'`

- A description that has no blanks:
  'Discovery'
  Should return: 'DISCOVERY'
- A description that is all blanks:
  ' '
  Should return: '' (this is an empty- zero-length- length string)
- A description that is null:
  null
  Should return: null

When you first start to think about creating these expressions, you probably don't think about all of these possibilities- but they could happen. So how would you test your expression? Instead of running it against the products table, set up a variable; give it the value to be tested and run your expression against the variable. Then you can change the value in the variable and rerun the expression.

Suppose we did this with the nested function example:

```
Select LTRIM(UPPER( '       This is MY sTrInG')) ;
```

We could do:

```
set @v :=  '       This is MY sTrInG';
Select LTRIM(UPPER( @v)) ;
+--------------------+
| LTRIM(UPPER( @v)) |
+--------------------+
| THIS IS MY STRING |
+--------------------+

set @v := '       What about digits 2 3 4 and punctuation !#%$';
Select LTRIM(UPPER( @v)) ;
+-----------------------------------------------+
| LTRIM(UPPER( @v))                             |
+-----------------------------------------------+
| WHAT ABOUT DIGITS 2 3 4 AND PUNCTUATION !#%$ |
+-----------------------------------------------+
```

Back to our function expression: This is going to use three functions that are discussed in this unit's notes. The functions are

- UPPER which returns the upper cased string
- REPLACE which changes a character to another character
- LEFT which returns a set number of characters from the left edge of the string

We are going to need to use these three functions together. We could start working with the description in the products table, but it can be easier to start with a literal value that we can control. And we can set this up as a variable and change the value to test other possible strings.

First get rid of the blanks using replace; I am replacing each space (' ') with nothing- the ZLS (').

```
set @v := 'Bird cage- simple; wire frame two feeder trays';

Select replace(@v, ' ', '');
+---------------------------------------+
| replace(@v, ' ', '')                  |
+---------------------------------------+
| Birdcage-simple;wireframetwofeedertrays |
+---------------------------------------+
```

Now we can change this to upper case letters. Use the replace function expression as an argument to the upper function.

```
set @v := 'Bird cage- simple; wire frame two feeder trays';
Select replace(@v, ' ', '');
Select upper(replace(@v, ' ', ''));
+----------------------------------------+
| upper(replace(@v, ' ', ''))            |
+----------------------------------------+
| BIRDCAGE-SIMPLE;WIREFRAMETWOFEEDERTRAYS |
+----------------------------------------+
```

Now we want the first 10 characters; that can be done with the Left function.

```
set @v := 'Bird cage- simple; wire frame two feeder trays';
select left(upper(replace(@v, ' ', '')), 10) ;
+--------------------------------------+
| left(upper(replace(@v, ' ', '')), 10) |
+--------------------------------------+
| BIRDCAGE-S                           |
+--------------------------------------+
```

So far this looks good. But test the other values we considered at the start of this discussion.

The data you are testing could be null, you can test that

```
set @v := null;
```

If you are happy with the function, then you can use that expression in a query against the products table, replacing the @v variable with the column name.

```
Select left(upper(replace(prod_desc, ' ', '')), 10)
From product.products;
```

This approach lets you control the data you need to test; lets you work a little at a time which helps with the parentheses problem and helps you think methodically.

If you have used other DBMS you should note some features of MySQL

- MySQL functions will commonly return a null or a zero instead of an error message for inappropriate arguments.
- MySQL has expected the application programs to check arguments for validity and MySQL functions take a different attitude towards error checking than some other dbms.
- MySQL functions may have several aliases.
- MySQL functions are apt to change somewhat with different releases of the software.
- **Do not have any blanks between the name of the function and the opening parenthesis**

For the Oracle folk: MySQL allows the use of dual as a dummy table token; this is not quite the same as the Oracle dual table but it does let you write sql such as

```
Select UPPER( '       This is MY sTrInG') from dual;
```