Table of Contents

A sampling of the MySQL string oriented functions.

Concat	RPad	Repeat	ELT
Concat_ws	LPad	Replace	Field
Upper	Substring	Reverse	Find_in_set
Lower	Substring_Index	Space	
RTrim	Instr	Ascii	
LTrim	Locate	Char	

1. Concat, Concat_ws

Demo 01: We have been using **concat** for several weeks.

```
+----------------------------------+
| Concat('C', 'AT', ' Fluff', 'y') |
+----------------------------------+
| CAT Fluffy                       |
+----------------------------------+
```

Demo 02: If you want a particular string placed between each item to be concatenated, use Concat_ws. The first argument is placed between the other items to be concatenated

```
+----------------------------------------+
| Concat_ws(' ', 'Fluffy', 'the', 'cat') |
+----------------------------------------+
| Fluffy the cat                         |
+----------------------------------------+

+-------------------------------------+
| Concat_ws(' is a ', 'Fluffy', 'cat') |
+-------------------------------------+
| Fluffy is a cat                     |
+-------------------------------------+

+-----------------------------------------------------+
| Concat_ws(' and ', 'Fluffy', 'Mopsy', 'Cottontail') |
+-----------------------------------------------------+
| Fluffy and Mopsy and Cottontail                     |
+-----------------------------------------------------+
```

It is always a good idea to check what functions do with null arguments.

First concat with a null; nulls propagate

```
select
  concat('a','b')
, concat(null,'b')
, concat('c', null,'b');
```

```
+-----------------+-----------------+----------------------+
| concat('a','b') | concat(null,'b') | concat('c', null,'b') |
+-----------------+-----------------+----------------------+
| ab              | NULL            | NULL                 |
+-----------------+-----------------+----------------------+
```

But we get a different results with concat_ws with nulls.

```
select
  concat_ws(' ', 'a','b')
, concat_ws(' ', null,'b')
, concat_ws(' ', 'c', null,'b');
+-----------------------+------------------------+------------------------------+
| concat_ws(' ','a','b') | concat_ws(' ',null,'b') | concat_ws(' ', 'c', null,'b') |
+-----------------------+------------------------+------------------------------+
| a b                   | b                      | c b                          |
+-----------------------+------------------------+------------------------------+

select concat_ws(null,'a','b');
+-------------------------+
| concat_ws(null,'a','b') |
+-------------------------+
| NULL                    |
+-------------------------+
```

Demo 03: **Concat_ws**. (The \G terminator changes the display to a vertical display. This terminator might not work in all clients.)

```
Select Concat_ws(' ', prod_desc, 'is in the', catg_desc,
       'group and has a list price of $',prod_list_price) as "Item Desc"
From product.products
join product.categories using( catg_id)
order by prod_list_price desc
limit 4\G
*************************** 1. row ***************************
Item Desc: Low Energy washer Dryer combo is in the APPLIANCES group and has a
list price of $ 850.00
*************************** 2. row ***************************
Item Desc: Satin four-poster cat bed is in the PET SUPPLIES group and has a
list price of $ 549.99
*************************** 3. row ***************************
Item Desc: Our highest end cat tree- you gotta see this is in the PET SUPPLIES
group and has a list price of $ 549.99
*************************** 4. row ***************************
Item Desc: Full-sized Washer is in the APPLIANCES group and has a list price of
$ 549.99
```

2. Binary

The default behavior for MySQL string comparisons is case insensitive. So the following query returns rows- a row with a match on 'armadillo' and a match on "Armadillo" and it would match " ArmAdIllO ", "ArmaDiLlO" etc. () there are2 inserts in the demos for animals 70 and 71)

Demo 04: Binary

```
select z_id, z_name, z_type
from a_testbed.zoo_2016
where z_type = 'armadillo'
;
```

```
+------+---------+-----------+
| z_id | z_name  | z_type    |
+------+---------+-----------+
|   25 | Abigail | Armadillo |
|   70 | Anders  | armadillo |
|   71 | Anne    | ARMADILLO |
+------+---------+-----------+
```

We may want to use case sensitive comparisons in a particular query. We can do this by placing the keyword BINARY before the springs being compared.

Demo 05: Binary

```
select z_id, z_name, z_type
from a_testbed.zoo_2016
where  BINARY z_type = 'armadillo';
+------+---------+-----------+
| z_id | z_name  | z_type    |
+------+---------+-----------+
|   70 | Anders  | armadillo |
+------+---------+-----------+
```

You can also use BINARY in the order by clause to get a case sensitive sort.

```
order by binary an_type;
```

3. Capitalization

Demo 06: **Upper** and **Lower** return a string in the specified case pattern. UCase and LCase are alternate function names..

```
+---------------------+-----------------------------------+---------------------+
| Upper( 'MY sTrInG') | Upper( '50 Phelan Ave SF 94112') | Lower( 'MY sTrInG') |
+---------------------+-----------------------------------+---------------------+
| MY STRING           | 50 PHELAN AVE SF 94112           | my string           |
+---------------------+-----------------------------------+---------------------+
```

4. Padding and Trimming strings

Demo 07: **RTRIM** and **LTRIM** remove blanks from the Right/Left side of the string. Note the nested functions for the third example.

```
select
  rtrim( '     San Francisco  CA   ') as Rtrim
, ltrim( '     San Francisco  CA   ') as Ltrim
, rtrim(ltrim( '     San Francisco  CA   ')) as  "R&LTrim";
+------------------------+---------------------+-------------------+
| Rtrim                  | Ltrim               | R&LTrim           |
+------------------------+---------------------+-------------------+
|      San Francisco  CA | San Francisco  CA   | San Francisco  CA |
+------------------------+---------------------+-------------------+
```

Demo 08: **RPAD** and **LPad** add characters to the edge of the string to the specified length and may truncate data.

```
select
  rpad( 'San Francisco', 15, '-') as RPAD
, lpad( 'San Francisco', 15, '-') as LPAD
, rpad( 'San Francisco',  5, '-') as "RPad_short";
+-----------------+----------------+------------+
| RPAD            | LPAD           | RPad_short |
+-----------------+----------------+------------+
| San Francisco-- | --San Francisco | San F      |
+-----------------+----------------+------------+
```

5. Parts of strings and matches within a string

Demo 09: **SUBSTRING**: returns part of a string. Substring (strExp1, pos_start, len) returns part of strExp1, starting from position pos_start and continuing for len characters. Substr , mid are aliases.

```
+-------------------------------+---------------------------------+
| SUBSTRING( 'ABCDEFGHIJK',1, 5) | SUBSTRING( 'ABCDEFGHIJK', 5, 3) |
+-------------------------------+---------------------------------+
| ABCDE                         | EFG                             |
+-------------------------------+---------------------------------+

+-------------------------------+---------------------------------+
| SUBSTRING( 'ABCDEFGHIJK',50, 5) | SUBSTRING( 'ABCDEFGHIJK', 5, 60) |
+-------------------------------+---------------------------------+
|                               | EFGHIJK                         |
+-------------------------------+---------------------------------+
```

Using a negative value for start_pos means that you count from the end of the string.

```
+-------------------------------+-----------------------------------+
| SUBSTRING( 'ABCDEFGHIJK',-5, 2) | SUBSTRING( 'ABCDEFGHIJK', -50, 20) |
+-------------------------------+-----------------------------------+
| GH                            |                                   |
+-------------------------------+-----------------------------------+
```

Demo 10: **LEFT** and **RIGHT** return the indicated number of characters from the Left or Right of the string.

```
+-----------------------+------------------------+-------------------------+
| left('ABCDEFGHIJK', 5) | RIGHT('ABCDEFGHIJK', 5) | RIGHT('ABCDEFGHIJK', 55) |
+-----------------------+------------------------+-------------------------+
| ABCDE                 | GHIJK                  | ABCDEFGHIJK             |
+-----------------------+------------------------+-------------------------+
```

Demo 11: **SUBSTRING_INDEX** (strExp1, delimiter, count) breaks the strExp1 into substrings using the delimiter and then returns the substring up to the count_th delimiter

This returns the substring up to the first comma.

```
select SUBSTRING_INDEX( 'Cat,Ant,Elephant, Blue Frog, Zebra', ',', 1);
+-------------------------------------------------------------+
| SUBSTRING_INDEX( 'Cat,Ant,Elephant, Blue Frog, Zebra', ',', 1) |
+-------------------------------------------------------------+
| Cat                                                         |
+-------------------------------------------------------------+
```

Demo 12: **This returns the substring up to the third comma.**

```
select SUBSTRING_INDEX( 'Cat,Ant,Elephant, Blue Frog, Zebra', ',', 3);
+-------------------------------------------------------------+
| SUBSTRING_INDEX( 'Cat,Ant,Elephant, Blue Frog, Zebra', ',', 3) |
+-------------------------------------------------------------+
| Cat,Ant,Elephant                                            |
+-------------------------------------------------------------+
```

Demo 13: **This returns the substring counting from the end of the string.**

```
select SUBSTRING_INDEX( 'Cat,Ant,Elephant, Blue Frog, Zebra', ',', -1);
+-------------------------------------------------------------+
| SUBSTRING_INDEX( 'Cat,Ant,Elephant, Blue Frog, Zebra', ',', -1) |
+-------------------------------------------------------------+
|  Zebra                                                      |
+-------------------------------------------------------------+
```

Demo 14: **This uses 'a' as the delimiter. This is case dependent..**

```
select SUBSTRING_INDEX( 'Cat,Ant,Elephant, Blue Frog, Zebra', 'a', 2);
+-------------------------------------------------------------+
| SUBSTRING_INDEX( 'Cat,Ant,Elephant, Blue Frog, Zebra', 'a', 2) |
+-------------------------------------------------------------+
| Cat,Ant,Eleph                                               |
+-------------------------------------------------------------+
```

Demo 15: **INSTR** returns the location of the substring in the target string.

```
+-----------------------------+-------------------------------+
| INSTR( 'ABCDEABCDE', 'CD' ) | INSTR(  'ABCDEABCDE','zebra') |
+-----------------------------+-------------------------------+
|                           3 |                             0 |
+-----------------------------+-------------------------------+
```

Demo 16: **LOCATE** also returns the location of the substring in the target string. With this function the search argument comes first. You can also add a third argument which states the position in which to start the search.

```
+-------------------------------+---------------------------------+
| LOCATE( 'CD', 'ABCDEABCDE'  ) | LOCATE( 'CD', 'ABCDEABCDE', 5) |
+-------------------------------+---------------------------------+
|                             3 |                               8 |
+-------------------------------+---------------------------------+
```

6. Changing the string contents

Demo 17: **REPLACE** replaces every occurrence of the second argument with the third argument.

```
+-------------------------------------+--------------------------------------+
| REPLACE('ABCDABCDABCD', 'B', 'cat') | REPLACE('ABCDABCDABCD', 'BCD', '-') |
+-------------------------------------+--------------------------------------+
| AcatCDAcatCDAcatCD                  | A-A-A-                               |
+-------------------------------------+--------------------------------------+
```

Demo 18: **You can use replace to remove patterns by using a zero length string for the third argument.**

```
+-----------------------------+
| REPLACE('ABCDABCD', 'CD', '') |
+-----------------------------+
| ABAB                        |
+-----------------------------+
```

Replace with a null third argument

```
+-------------------------------+--------------------------------+
| REPLACE('ABCDABCD', 'C', null) | REPLACE('ABCDABCD', 'X', null) |
+-------------------------------+--------------------------------+
| NULL                          | ABCDABCD                       |
+-------------------------------+--------------------------------+
```

Demo 19: **INSERT**(strExp1, pos_start, len, StrExp2,) starts at position pos_start in strExp1 and removes len characters. It then puts strExp2 in that place in the string. Len can be 0 which results in just an insert.

```
+---------------------------+---------------------------------+
| INSERT('abcdefgh',1,4,'X') | INSERT('abcdefgh',5,2,'xzyxzy') |
+---------------------------+---------------------------------+
| Xefgh                     | abcdxzyxzygh                    |
+---------------------------+---------------------------------+
```

7. Misc string functions

Demo 20: **Length**: Length(strExp) returns the number of characters in the expression. If the argument is a null string, Length returns null (not 0).

```
+----------------------+------------+--------------+
| length('   abc   ') | length('') | length(null) |
+----------------------+------------+--------------+
|                    9 |          0 |         NULL |
+----------------------+------------+--------------+
```

Demo 21: **REPEAT** duplicates the first argument the indicated number of times

```
+-------------------+
| repeat('*-* ',3) |
+-------------------+
| *-* *-* *-*      |
+-------------------+
```

Demo 22: **REVERSE** reverses the characters in the string

```
+----------------------+
| reverse('abcdefgh') |
+----------------------+
| hgfedcba            |
+----------------------+
```

Demo 23: **SPACE** creates a string of spaces of the indicated length

```
+------------------------------+
| concat('A' , space(5) , 'Z') |
+------------------------------+
| A     Z                      |
+------------------------------+
```

Demo 24: **ASCII** returns the ASCII number corresponding to the first character in the argument string

```
+--------------+--------------+--------------+-----------+-------------+
| ascii('Cat') | ascii('Dog') | ascii('dog') | ascii('') | ascii(null) |
+--------------+--------------+--------------+-----------+-------------+
|           67 |           68 |          100 |         0 |        NULL |
+--------------+--------------+--------------+-----------+-------------+
```

Demo 25: **CHAR** returns the character associated with an ASCII number

```
+----------+----------+----------+----------+-----------+-----------+
| char(68) | char(69) | char(70) | char(50) | char(123) | char(124) |
+----------+----------+----------+----------+-----------+-----------+
| D        | E        | F        | 2        | {         | |         |
+----------+----------+----------+----------+-----------+-----------+
```

Demo 26: **FIELD**

The Field function gets two or more arguments. The first argument is the value you are trying to match, the second and other arguments are possible matches. If the first argument is found, then the position of that argument is returned, otherwise 0.

```
select Field('cat', 'ant','bear','catfish','dog','cat','elk');
+-----------------------------------------------------------+
| Field('cat', 'ant','bear','catfish','dog','cat','elk') |
+-----------------------------------------------------------+
|                                                         5 |
+-----------------------------------------------------------+

select Field('moose', 'ant','bear','catfish','dog','cat','elk');
+-------------------------------------------------------------+
| Field('moose', 'ant','bear','catfish','dog','cat','elk') |
+-------------------------------------------------------------+
|                                                           0 |
+-------------------------------------------------------------+
```

Nulls always need testing; if the first argument is null, then the function returns 0, not a null.

```
select Field(null, 'ant');
+--------------------+
| Field(null, 'ant') |
+--------------------+
|                  0 |
+--------------------+

select Field('moose', null);
+----------------------+
| Field('moose', null) |
+----------------------+
|                    0 |
+----------------------+

select Field('moose', null, 'cat');
+-----------------------------+
| Field('moose', null, 'cat') |
+-----------------------------+
|                           0 |
+-----------------------------+

select Field('moose', null, 'cat', 'moose');
+-------------------------------------+
| Field('moose', null, 'cat', 'moose') |
+-------------------------------------+
|                                   3 |
+-------------------------------------+
```

Field will work with numbers; again you need to take care with this.

```
select Field(12,1002,120,2011,12, 2012,12);
+-------------------------------------+
| Field(12,1002,120,2011,12, 2012,12) |
+-------------------------------------+
|                                   4 |
+-------------------------------------+

select Field(0,1002,120,2011,12, 2012,12);
+-------------------------------------+
| Field(0,1002,120,2011,12, 2012,12) |
+-------------------------------------+
|                                  0 |
+-------------------------------------+
```

Be careful to avoid mixing types- why does this return 3?

```
select Field(0,1002,120,'ant',12, 2012,12);
+-------------------------------------+
| Field(0,1002,120,'ant',12, 2012,12) |
+-------------------------------------+
|                                   3 |
+-------------------------------------+
```

Demo 27: **ELT**

ELT is the complement of Field. The first argument is a number and the rest of the arguments are values- the function returned the value that corresponds to the first argument.

```
select ELT(2, 'ant', 'cat', 'dog', 'bird', 'hedgehog');
+-----------------------------------------------+
| ELT(2, 'ant', 'cat', 'dog', 'bird', 'hedgehog') |
+-----------------------------------------------+
| cat                                           |
+-----------------------------------------------+

select ELT(8, 'ant', 'cat', 'dog', 'bird', 'hedgehog');
+-----------------------------------------------+
| ELT(8, 'ant', 'cat', 'dog', 'bird', 'hedgehog') |
+-----------------------------------------------+
| NULL                                          |
+-----------------------------------------------+

select ELT(0, 'ant', 'cat', 'dog', 'bird', 'hedgehog');
+-----------------------------------------------+
| ELT(0, 'ant', 'cat', 'dog', 'bird', 'hedgehog') |
+-----------------------------------------------+
| NULL                                          |
+-----------------------------------------------+

select ELT(3.5, 'ant', 'cat', 'dog', 'bird', 'hedgehog');
+-------------------------------------------------+
| ELT(3.5, 'ant', 'cat', 'dog', 'bird', 'hedgehog') |
+-------------------------------------------------+
| bird                                            |
+-------------------------------------------------+
```

Demo 28: **FIND_IN_SET**

The FIND_IN_SET function gets two arguments, the first is a string and the second is a comma-separated list. (a set value). The function returns the number of the first element in the list that matches the first argument. Avoid spaces in the literals. The first argument should not contain a comma.

First- two examples with string literals.

```
select Find_In_set('cat', 'ant,bear,catfish,dog,cat,elk');
+-----------------------------------------------------+
| Find_In_set('cat', 'ant,bear,catfish,dog,cat,elk') |
+-----------------------------------------------------+
|                                                   5 |
+-----------------------------------------------------+

select Find_In_set('moose', 'ant,bear,catfish,dog,cat,elk');
+-----------------------------------------------------+
| Find_In_set('moose', 'ant,bear,catfish,dog,cat,elk') |
+-----------------------------------------------------+
|                                                   0 |
+-----------------------------------------------------+
```

Demo 29: How could we use this with a table?

Set a variable to the list of animal types we are considering.

```
set @list = 'cat,dog,bird';
```

Then use that variable as a function argument,

```
select an_type, Find_In_set(an_type, @list) as Found
from vt_animals;
+-----------+-------+
| an_type   | Found |
+-----------+-------+
| bird      |     3 |
| bird      |     3 |
| bird      |     3 |
| cat       |     1 |
| cat       |     1 |
| cat       |     1 |
| cat       |     1 |
| cat       |     1 |
| cat       |     1 |
| cat       |     1 |
| cat       |     1 |
| chelonian |     0 |
| chelonian |     0 |
| dog       |     2 |
| dog       |     2 |
| dog       |     2 |
| dormouse  |     0 |
```

We could use the function in the Where clause;

```
select an_name, an_type
from vt_animals
where Find_In_set(an_type, @list) >0;
+-----------------+---------+
| an_name         | an_type |
+-----------------+---------+
| Gutsy           | cat     |
| NULL            | bird    |
| NULL            | bird    |
| Mr Peanut       | bird    |
| Burgess         | dog     |
| Ursula          | cat     |
| Napper          | cat     |
| Pinkie          | dog     |
| Calvin Coolidge | dog     |
| Adalwine        | cat     |
| Baldric         | cat     |
| Etta            | cat     |
| Manfried        | cat     |
| Waldrom         | cat     |
+-----------------+---------+
```

Demo 30: Using an In list

We could do the same logic with an IN list. With the In List, each different animal type value has to be independently delimited. With the FIND_IN_SET function, we have one string. This approach is often easier when the list of animal types is coming from an external application. The list is a comma-separated values (CSV) string; csv strings occur in many programming situations.

```
Select an_name, an_type
From vt_animals
Where an_type in ('cat', 'dog', 'bird');
```