Table of Contents

[1. Concat, Concat\_ws 1](#_Toc363672556)

[2. Binary 2](#_Toc363672557)

[3. Capitalization 3](#_Toc363672558)

[4. Padding and Trimming strings 3](#_Toc363672559)

[5. Parts of strings and matches within a string 4](#_Toc363672560)

[6. Changing the string contents 5](#_Toc363672561)

[7. Misc string functions 6](#_Toc363672562)

A sampling of the MySQL string oriented functions.

Concat

Concat\_ws

Upper

Lower

RTrim

LTrim

RPad

LPad

Substring

Substring\_Index

Instr

Locate

Repeat

Replace

Reverse

Space

Ascii

Char

ELT

Field

Find\_in\_set

1. Concat, Concat\_ws
2. We have been using **concat** for several weeks.

+----------------------------------+

| Concat('C', 'AT', ' Fluff', 'y') |

+----------------------------------+

| CAT Fluffy |

+----------------------------------+

1. 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 |

+----------------------------------------+

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 |

+-------------------------+

1. **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, 'costs', prod\_list\_price,

'and has a warranty of',prod\_warranty\_period, 'months') as "Item Desc"

From a\_prd.products

limit 4\G

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Item Desc: Seven speed BnD hand mixer, black costs 125.00 and has a warranty of

12 months

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 2. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Item Desc: Set of 12 barbells 15 pounds costs 150.00 and has a warranty of 60 months

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 3. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Item Desc: Cork-backed dartboard with hanger costs 12.95 and has a warranty of 60 months

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 4. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Item Desc: Basketball costs 29.95 and has a warranty of 60 months

1. Binary

The default behavior for MySQL string comparisons is case insensitive. So the following query returns rows- a row with a match on 'dog' and a match on "Dog" and it would match "DOG", "DoG" etc.

1. Binary

Select an\_id, an\_name, an\_type

From zoo\_animals

Where an\_type = 'dog';

+-------+----------------+---------+

| an\_id | an\_name | an\_type |

+-------+----------------+---------+

| 119 | Randle Patrick | Dog |

| 1023 | Nukem | Dog |

| 101 | Nicholas | Dog |

| 102 | Cosette | Dog |

| 103 | Bernadette | Dog |

| 110 | cerberus | dog |

| 145 | Jack | Dog |

| 112 | sam | dog |

| 113 | sandy | dog |

| 205 | Lala | dog |

| 103 | Goofy | Dog |

+-------+----------------+---------+

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.

1. Binary

Select an\_id, an\_name, an\_type

From zoo\_animals

Where BINARY an\_type = 'dog';

+-------+----------+---------+

| an\_id | an\_name | an\_type |

+-------+----------+---------+

| 110 | cerberus | dog |

| 112 | sam | dog |

| 113 | sandy | dog |

| 205 | Lala | dog |

+-------+----------+---------+

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

order by binary an\_type;

1. Capitalization
2. **Upper** and **Lower** return a string in the specified case pattern. UCase and LCase are aliases.

+---------------------+----------------------------------+---------------------+

| Upper( 'MY sTrInG') | Upper( '50 Phelan Ave SF 94112') | Lower( 'MY sTrInG') |

+---------------------+----------------------------------+---------------------+

| MY STRING | 50 PHELAN AVE SF 94112 | my string |

+---------------------+----------------------------------+---------------------+

1. Padding and Trimming strings
2. **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 |

+------------------------+----------------------+-------------------+

1. **RPAD** and **LPad** add characters to the edge of the string to the specified length..

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 |

+-----------------+-----------------+------------+

1. Parts of strings and matches within a string
2. **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 | |

+---------------------------------+------------------------------------+

1. **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 |

+------------------------+-------------------------+--------------------------+

1. **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 |

+----------------------------------------------------------------+

1. **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 |

+----------------------------------------------------------------+

1. **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 |

+-----------------------------------------------------------------+

1. **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 |

+----------------------------------------------------------------+

1. **INSTR** returns the location of the substring in the target string.

+-----------------------------+-------------------------------+

| INSTR( 'ABCDEABCDE', 'CD' ) | INSTR( 'ABCDEABCDE','zebra') |

+-----------------------------+-------------------------------+

| 3 | 0 |

+-----------------------------+-------------------------------+

1. **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 |

+-------------------------------+--------------------------------+

1. Changing the string contents
2. **REPLACE** replaces every occurrence of the second argument with the third argument.

+-------------------------------------+-------------------------------------+

| REPLACE('ABCDABCDABCD', 'B', 'cat') | REPLACE('ABCDABCDABCD', 'BCD', '-') |

+-------------------------------------+-------------------------------------+

| AcatCDAcatCDAcatCD | A-A-A- |

+-------------------------------------+-------------------------------------+

1. **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 |

+--------------------------------+--------------------------------+

1. **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 |

+----------------------------+---------------------------------+

1. Misc string functions
2. **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 |

+---------------------+------------+--------------+

1. **REPEAT** duplicates the first argument the indicated number of times

+------------------+

| repeat('\*-\* ',3) |

+------------------+

| \*-\* \*-\* \*-\* |

+------------------+

1. **REVERSE** reverses the characters in the string

+---------------------+

| reverse('abcdefgh') |

+---------------------+

| hgfedcba |

+---------------------+

1. **SPACE** creates a string of spaces of the indicated length

+------------------------------+

| concat('A' , space(5) , 'Z') |

+------------------------------+

| A Z |

+------------------------------+

1. **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 |

+--------------+--------------+--------------+-----------+-------------+

1. **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 | { | | |

+----------+----------+----------+----------+-----------+-----------+

1. **FIELD**

The Field function gets two or more arguments. The first argument ids 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 returns, 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 |

+--------------------------------------+

Find 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 |

+-------------------------------------+

1. **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 |

+---------------------------------------------------+

1. **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 |

+----------------------------------------------------+

1. 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 |

+-----------------+---------+

1. 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');