Table of Contents

1. Regular Expressions

Regular expressions allow you to do more complex pattern matching than wildcards. Wildcards are limited to matching any single character and any series of characters. With regular expressions we can build a pattern that specifies specific characters to search for, ranges of characters and repeating patterns of characters.

A regular expression is a string that expresses a pattern for text matching. The regular expression string can contain literal characters and metacharacters. For example the regular expression 'cat' matches the string 'cat and the regular expression 'p..t' matches a string that contains a 'p' followed by any two characters followed by a 't'.

One of the most important things to understand about regular expressions is that they can work very well for matching strings that have some sort of consistent pattern, but they are not a good for matching unstructured text. You may have to decide if you want to use a pattern that produces a lot of matches including undesired matches (false positives), or a pattern that produces fewer matches but misses some of the matches you want to find. You can find examples of regular expression patterns on various web sites but you need to test these- for example does the pattern for a phone number assume that you will use parentheses and a dash (415) 239-3768 or does it allow the format 415.239.3768; does it handle extensions?, country codes? Does the url pattern you found assume that the top level domain is two or three characters long? Does it assume English characters only?

If you have worked with regular expressions you know that they can get quite complex- we will stick with some simple patterns. MySQL follows one version of regular expressions; if you want to do complex regular expressions, read the manual and test your expression carefully.

1.1. Syntax Rules

MySQL uses the name RegExp for pattern matching . RLike is a synonym for RegExp.

You can negate this with Not- Not RegExp

Usage

```
Expr_1 RegExp pattern_1
```

- Returns 1 or 0; 1 if there is a match and 0 if there is no match
  If either Expr_1 or pattern_1 is null, then a null is returned.
- RegExp is not case specific except with binary strings

The manual will often put the regular expression example in the Select clause in order to do short focused tests; for a practical purpose you are more apt to use the regular expression in a Where clause. If the return value of regExp is 1, then that row passes the Where test.

For the demos we will start with some simple tests and also use a table set up for this purpose. The table has an id column and a name column; we will be testing against the name column.

Put this in the a_testbed database.

```
Create table z_reg ( id integer auto_increment primary key, name varchar(25));
-- Sample inserts
Insert into z_reg (name) values ('Fluffy'), ('goofy'), ('ursula'), ('greg');
Insert into z_reg (name) values ('pout'), ('Sam 415'), ('pretty bird');
Insert into z_reg (name) values ('pat'), ('peat'), ('Patricia'), ('Impromptu');
Insert into z_reg (name) values ('Pete'), ('pat the cat'), ('C3PO');
```

```
+----+-------------+
| id | name        |
+----+-------------+
|  1 | Fluffy      |
|  2 | goofy       |
|  3 | ursula      |
|  4 | greg        |
|  5 | pout        |
|  6 | Sam 415     |
|  7 | pretty bird |
|  8 | pat         |
|  9 | peat        |
| 10 | Patricia    |
| 11 | Impromptu   |
| 12 | Pete        |
| 13 | pat the cat |
| 14 | C3PO        |
| 15 | Mary Proud  |
| 16 | ptt         |
| 17 | pita        |
+----+-------------+
```

1.2. Demos

Demo 01: Identify test. This tests an expression 'goofy' against simple regular expressions that contain no special characters. The purpose of this is just to let you see the return value.

```
Select
  'goofy' Regexp 'goofy' as d_1a
, 'goofy' Regexp 'max'   as d_1b
;
+------+------+
| d_1a | d_1b |
+------+------+
|    1 |    0 |
+------+------+
```

Demo 02: This show that this is not case specific unless you include binary

```
Select
  'goofy' Regexp 'goofy' as d_2a
, 'goofy' Regexp 'Goofy' as d_2b
, binary 'goofy' Regexp 'Goofy' as d_2c
;
+------+------+------+
| d_2a | d_2b | d_2c |
+------+------+------+
|    1 |    1 |    0 |
+------+------+------+
```

Demo 03: These regular expressions are matched if the pattern occurs anywhere in the string. The first is just checking if there is a g anywhere in the string. The second checks for the pattern oo which is matched. The third is looking for the pattern oy- this is not matched. We have an o and a y but not

the pattern oy. Note that this is different than the way the LIKE operator works- Like tries to match the entire string; this version of regular expressions does not have to match the entire string.

```
Select
  'goofy' Regexp 'g'  as d_3a
, 'goofy' Regexp 'oo' as d_3b
, 'goofy' Regexp 'oy' as d_3c
;
+------+------+------+
| d_3a | d_3b | d_3c |
+------+------+------+
|    1 |    1 |    0 |
+------+------+------+
```

1.3. Metacharacters

To be more useful, we add metacharacters. The Like operator recognized the metacharacters % and _. Regular expressions have a much richer set of metacharacters. You can find a list of them in the MySQL documentation.

What the regex expressions provide is the ability to search for ranges/lists of characters. Instead of looking for any single character as with a wildcard, you could look for any of the listed characters [aeiouy]. You could also specify that you want to match exactly 6 of those characters in a row [aeiouy]{6}

You could match a pattern of as many 0s and 1s as occurs by using [01]*

There are also predefined patterns such as `[:lower:]` which stands for any lower case letter.

In this first set of demos we use the metacharacters

^	start of the string
$	end of the string
.	any single character
{n}	repetition; want exactly n of the previous character g{3} matches ggg
{n, m}	repetition; want between n and m of the previous character g{3,5} matches ggg, gggg, ggggg
*****	repetition; any number, including 0 of the preceding character
+	repetition; any number, but at least one, of the preceding character
?	repetition; zero or one of the preceding character
[abc…]	list - matches any one of the included characters
[a-p]	range - matches any one of the characters in the indicated range

Demo 04: The ^ character matches the start of the string.

```
Select *
From z_reg
Where name  RegExp '^g';
+----+-------+
| id | name  |
+----+-------+
|  2 | goofy |
|  4 | greg  |
+----+-------+
```

Demo 05: The $ character matches the end of the string.

```
Select *
From z_reg
Where name  RegExp 'g$';
```

```
+----+-------+
| id | name  |
+----+-------+
|  4 | greg  |
+----+-------+
```

Demo 06: The . (dot) character matches any single character. This pattern matches any string that starts with a p and ends with a t and has exactly one character between.

```
Select * From z_reg
Where name  RegExp '^p.t$';
+----+------+
| id | name |
+----+------+
|  8 | pat  |
| 16 | ptt  |
+----+------+
```

Demo 07: This pattern matches any string that starts with a p and ends with a t and has exactly two characters between .

```
Select * From z_reg
Where name  RegExp '^p..t$';
+----+------+
| id | name |
+----+------+
|  5 | pout |
|  9 | peat |
+----+------+
```

Demo 08: It can help to also display rows that do not match the pattern

```
Select * From z_reg
Where name not RegExp '^p..t$';
+----+-------------+
| id | name        |
+----+-------------+
|  1 | Fluffy      |
|  2 | goofy       |
|  3 | ursula      |
|  4 | greg        |
|  6 | Sam 415     |
|  7 | pretty bird |
|  8 | pat         |
| 10 | Patricia    |
| 11 | Impromptu   |
| 12 | Pete        |
| 13 | pat the cat |
| 14 | C3PO        |
| 15 | Mary Proud  |
| 16 | ptt         |
| 17 | pita        |
+----+-------------+
```

Demo 09: We can use {n} to indicate that we want exactly n of the preceding character. Note that I do not have the $ to tie this to the end of the string.

```
Select * From z_reg
Where name  RegExp '^p.{3}t';
```

```
+----+-------------+
| id | name        |
+----+-------------+
|  7 | pretty bird |
| 13 | pat the cat |
+----+-------------+
```

Demo 10: We can use {n, m} to indicate that we want between n and m of the preceding character.

```
Select * From z_reg
Where name  RegExp '^p.{4,7}a';
+----+----------+
| id | name     |
+----+----------+
| 10 | Patricia |
+----+----------+
```

Demo 11: We can use * to indicate that we will match any number, including 0, of the preceding character.

```
Select * From z_reg
Where name  RegExp '^p.*t';
+----+-------------+
| id | name        |
+----+-------------+
|  5 | pout        |
|  7 | pretty bird |
|  8 | pat         |
|  9 | peat        |
| 10 | Patricia    |
| 12 | Pete        |
| 13 | pat the cat |
| 16 | ptt         |
| 17 | pita        |
+----+-------------+
```

Demo 12: Use + to indicate that you must match at least one. Use ? for matching either 0 or 1 of the preceding.

```
Select * From z_reg
Where name  RegExp 'pr?o';
+----+------------+
| id | name       |
+----+------------+
|  5 | pout       |
| 11 | Impromptu  |
| 14 | C3PO       |
| 15 | Mary Proud |
+----+------------+
```

Demo 13: The [ ] allows for a list or a range of characters to match.

```
Select * From z_reg
Where name  RegExp '[aeiouy]$';
+----+-----------+
| id | name      |
+----+-----------+
|  1 | Fluffy    |
|  2 | goofy     |
|  3 | ursula    |
| 10 | Patricia  |
| 11 | Impromptu |
```

```
| 12 | Pete      |
| 14 | C3PO      |
| 17 | pita      |
+----+-----------+
```

Demo 14: The [ ] allows for a list or a range of characters to match. This matches an r followed by a single vowel followed by a character in the range a-m.

```
Select * From z_reg
Where name  RegExp 'r[aeiouy][a-m]';
+----+-----------+
| id | name      |
+----+-----------+
|  4 | greg      |
| 10 | Patricia  |
| 11 | Impromptu |
+----+-----------+
```

Demo 15: Regular expression also includes character classes. This matches strings that include any whitespace character

```
Select * From z_reg
Where name RegExp '[[:blank:]]';
+----+-------------+
| id | name        |
+----+-------------+
|  6 | Sam 415     |
|  7 | pretty bird |
| 13 | pat the cat |
| 15 | Mary Proud  |
+----+-------------+
```

Demo 16: This matches strings that include an upper case letter followed by a lower case letter.

```
Select * from z_reg
Where binary name RegExp '[[:upper:]][[:lower:]]';
+----+------------+
| id | name       |
+----+------------+
|  1 | Fluffy     |
|  6 | Sam 415    |
| 10 | Patricia   |
| 11 | Impromptu  |
| 12 | Pete       |
| 15 | Mary Proud |
+----+------------+
```

Demo 17: This matches any digit

```
Select * From z_reg
Where name RegExp '[[:digit:]]';
+----+---------+
| id | name    |
+----+---------+
|  6 | Sam 415 |
| 14 | C3PO    |
+----+---------+
```