14.8.2 Regular Expressions

Table 14.14 Regular Expression Functions and Operators

Name	Description
NOT REGEXP	Negation of REGEXP
REGEXP	Whether string matches regular expression
REGEXP_INSTR()	Starting index of substring matching regular expression
REGEXP_LIKE()	Whether string matches regular expression
REGEXP_REPLACE()	Replace substrings matching regular expression
REGEXP_SUBSTR()	Return substring matching regular expression
RLIKE	Whether string matches regular expression

A regular expression is a powerful way of specifying a pattern for a complex search. This section discusses the functions and operators available for regular expression matching and illustrates, with examples, some of the special characters and constructs that can be used for regular expression operations. See also Section 5.3.4.7, “Pattern Matching”.

MySQL implements regular expression support using International Components for Unicode (ICU), which provides full Unicode support and is multibyte safe. (Prior to MySQL 8.0.4, MySQL used Henry Spencer's implementation of regular expressions, which operates in byte-wise fashion and is not multibyte safe. For information about ways in which applications that use regular expressions may be affected by the implementation change, see Regular Expression Compatibility Considerations.)

Prior to MySQL 8.0.22, it was possible to use binary string arguments with these functions, but they yielded inconsistent results. In MySQL 8.0.22 and later, use of a binary string with any of the MySQL regular expression functions is rejected with ER_CHARACTER_SET_MISMATCH.

- Regular Expression Function and Operator Descriptions
- Regular Expression Syntax
- Regular Expression Resource Control
- Regular Expression Compatibility Considerations

Regular Expression Function and Operator Descriptions

- ***expr* NOT REGEXP *pat***, ***expr* NOT RLIKE *pat***

  This is the same as `NOT (`***expr*` REGEXP `***pat***`)`.

- ***expr* REGEXP *pat***, ***expr* RLIKE *pat***

  Returns 1 if the string ***expr*** matches the regular expression specified by the pattern ***pat***, 0 otherwise. If ***expr*** or ***pat*** is `NULL`, the return value is `NULL`.

  `REGEXP` and `RLIKE` are synonyms for `REGEXP_LIKE()`.

  For additional information about how matching occurs, see the description for `REGEXP_LIKE()`.

```
mysql> SELECT 'Michael!' REGEXP '.*';
+-----------------------+
| 'Michael!' REGEXP '.*' |
+-----------------------+
|                     1 |
+-----------------------+
mysql> SELECT 'new*\n*line' REGEXP 'new\\*.\\*line';
+---------------------------------------+
| 'new*\n*line' REGEXP 'new\\*.\\*line' |
+---------------------------------------+
|                                     0 |
+---------------------------------------+
mysql> SELECT 'a' REGEXP '^[a-d]';
+--------------------+
| 'a' REGEXP '^[a-d]' |
+--------------------+
|                  1 |
+--------------------+
```

- REGEXP_INSTR(***expr*, *pat*[, *pos*[, *occurrence*[, *return_option*[, *match_type*]]]])**

  Returns the starting index of the substring of the string ***expr*** that matches the regular expression specified by the pattern ***pat***, 0 if there is no match. If ***expr*** or ***pat*** is `NULL`, the return value is `NULL`. Character indexes begin at 1.

  `REGEXP_INSTR()` takes these optional arguments:

  - ***pos***: The position in ***expr*** at which to start the search. If omitted, the default is 1.

- ***occurrence***: Which occurrence of a match to search for. If omitted, the default is 1.
- ***return_option***: Which type of position to return. If this value is 0, REGEXP_INSTR() returns the position of the matched substring's first character. If this value is 1, REGEXP_INSTR() returns the position following the matched substring. If omitted, the default is 0.
- ***match_type***: A string that specifies how to perform matching. The meaning is as described for REGEXP_LIKE().

For additional information about how matching occurs, see the description for REGEXP_LIKE().

```
mysql> SELECT REGEXP_INSTR('dog cat dog', 'dog');
+-----------------------------------+
| REGEXP_INSTR('dog cat dog', 'dog') |
+-----------------------------------+
|                                 1 |
+-----------------------------------+
mysql> SELECT REGEXP_INSTR('dog cat dog', 'dog', 2);
+--------------------------------------+
| REGEXP_INSTR('dog cat dog', 'dog', 2) |
+--------------------------------------+
|                                    9 |
+--------------------------------------+
mysql> SELECT REGEXP_INSTR('aa aaa aaaa', 'a{2}');
+-----------------------------------+
| REGEXP_INSTR('aa aaa aaaa', 'a{2}') |
+-----------------------------------+
|                                  1 |
+-----------------------------------+
mysql> SELECT REGEXP_INSTR('aa aaa aaaa', 'a{4}');
+-----------------------------------+
| REGEXP_INSTR('aa aaa aaaa', 'a{4}') |
+-----------------------------------+
|                                  8 |
+-----------------------------------+
```

- REGEXP_LIKE(***expr***, ***pat***[, ***match_type***])

  Returns 1 if the string ***expr*** matches the regular expression specified by the pattern ***pat***, 0 otherwise. If ***expr*** or ***pat*** is NULL, the return value is NULL.

  The pattern can be an extended regular expression, the syntax for which is discussed in Regular Expression Syntax. The pattern need not be a literal string. For example, it can be specified as a string expression or table column.

The optional ***match_type*** argument is a string that may contain any or all the following characters specifying how to perform matching:

- `c`: Case-sensitive matching.
- `i`: Case-insensitive matching.
- `m`: Multiple-line mode. Recognize line terminators within the string. The default behavior is to match line terminators only at the start and end of the string expression.
- `n`: The `.` character matches line terminators. The default is for `.` matching to stop at the end of a line.
- `u`: Unix-only line endings. Only the newline character is recognized as a line ending by the `.`, `^`, and `$` match operators.

If characters specifying contradictory options are specified within ***match_type***, the rightmost one takes precedence.

By default, regular expression operations use the character set and collation of the ***expr*** and ***pat*** arguments when deciding the type of a character and performing the comparison. If the arguments have different character sets or collations, coercibility rules apply as described in Section 12.8.4, “Collation Coercibility in Expressions”. Arguments may be specified with explicit collation indicators to change comparison behavior.

```
mysql> SELECT REGEXP_LIKE('CamelCase', 'CAMELCASE');
+--------------------------------------+
| REGEXP_LIKE('CamelCase', 'CAMELCASE') |
+--------------------------------------+
|                                    1 |
+--------------------------------------+
mysql> SELECT REGEXP_LIKE('CamelCase', 'CAMELCASE' COLLATE utf8mb4_0900_as_cs);
+-------------------------------------------------------------------+
| REGEXP_LIKE('CamelCase', 'CAMELCASE' COLLATE utf8mb4_0900_as_cs) |
+-------------------------------------------------------------------+
|                                                                 0 |
+-------------------------------------------------------------------+
```

match_type may be specified with the `c` or `i` characters to override the default case sensitivity. Exception: If either argument is a binary string, the arguments are handled in case-sensitive fashion as binary strings, even if ***match_type*** contains the `i` character.

> **Note**
>
> MySQL uses C escape syntax in strings (for example, `\n` to represent the newline character). If you want your ***expr*** or ***pat*** argument to contain a literal `\`, you must double it. (Unless the `NO_BACKSLASH_ESCAPES` SQL mode is enabled, in which case no escape character is used.)

```
mysql> SELECT REGEXP_LIKE('Michael!', '.*');
+------------------------------+
| REGEXP_LIKE('Michael!', '.*') |
+------------------------------+
|                            1 |
+------------------------------+
mysql> SELECT REGEXP_LIKE('new*\n*line', 'new\\*.\\*line');
+--------------------------------------------+
| REGEXP_LIKE('new*\n*line', 'new\\*.\\*line') |
+--------------------------------------------+
|                                          0 |
+--------------------------------------------+
mysql> SELECT REGEXP_LIKE('a', '^[a-d]');
+---------------------------+
| REGEXP_LIKE('a', '^[a-d]') |
+---------------------------+
|                         1 |
+---------------------------+
```

```
mysql> SELECT REGEXP_LIKE('abc', 'ABC');
+--------------------------+
| REGEXP_LIKE('abc', 'ABC') |
+--------------------------+
|                        1 |
+--------------------------+
mysql> SELECT REGEXP_LIKE('abc', 'ABC', 'c');
+-------------------------------+
| REGEXP_LIKE('abc', 'ABC', 'c') |
+-------------------------------+
|                             0 |
+-------------------------------+
```

- `REGEXP_REPLACE(`***expr***`, `***pat***`, `***repl***`[, `***pos***`[, `***occurrence***`[, `***match_type***`]]])`

Replaces occurrences in the string ***expr*** that match the regular expression specified by the pattern ***pat*** with the replacement string ***repl***, and returns the resulting string. If ***expr***, ***pat***, or ***repl*** is `NULL`, the return value is `NULL`.

`REGEXP_REPLACE()` takes these optional arguments:

- ***pos***: The position in ***expr*** at which to start the search. If omitted, the default is 1.
- ***occurrence***: Which occurrence of a match to replace. If omitted, the default is 0 (which means "replace all occurrences").
- ***match_type***: A string that specifies how to perform matching. The meaning is as described for `REGEXP_LIKE()`.

Prior to MySQL 8.0.17, the result returned by this function used the `UTF-16` character set; in MySQL 8.0.17 and later, the character set and collation of the expression searched for matches is used. (Bug #94203, Bug #29308212)

For additional information about how matching occurs, see the description for `REGEXP_LIKE()`.

```
mysql> SELECT REGEXP_REPLACE('a b c', 'b', 'X');
+--------------------------------+
| REGEXP_REPLACE('a b c', 'b', 'X') |
+--------------------------------+
| a X c                          |
+--------------------------------+
mysql> SELECT REGEXP_REPLACE('abc def ghi', '[a-z]+', 'X', 1, 3);
+-------------------------------------------------+
| REGEXP_REPLACE('abc def ghi', '[a-z]+', 'X', 1, 3) |
+-------------------------------------------------+
| abc def X                                       |
+-------------------------------------------------+
```

- `REGEXP_SUBSTR(`***expr***`, `***pat***`[, `***pos***`[, `***occurrence***`[, `***match_type***`]])`

Returns the substring of the string ***expr*** that matches the regular expression specified by the pattern ***pat***, `NULL` if there is no match. If ***expr*** or ***pat*** is `NULL`, the return value is `NULL`.

`REGEXP_SUBSTR()` takes these optional arguments:

- ***pos***: The position in ***expr*** at which to start the search. If omitted, the default is 1.
- ***occurrence***: Which occurrence of a match to search for. If omitted, the default is 1.

- ***match_type***: A string that specifies how to perform matching. The meaning is as described for `REGEXP_LIKE()`.

Prior to MySQL 8.0.17, the result returned by this function used the `UTF-16` character set; in MySQL 8.0.17 and later, the character set and collation of the expression searched for matches is used. (Bug #94203, Bug #29308212)

For additional information about how matching occurs, see the description for `REGEXP_LIKE()`.

```
mysql> SELECT REGEXP_SUBSTR('abc def ghi', '[a-z]+');
+----------------------------------------+
| REGEXP_SUBSTR('abc def ghi', '[a-z]+') |
+----------------------------------------+
| abc                                    |
+----------------------------------------+
mysql> SELECT REGEXP_SUBSTR('abc def ghi', '[a-z]+', 1, 3);
+-------------------------------------------------+
| REGEXP_SUBSTR('abc def ghi', '[a-z]+', 1, 3) |
+-------------------------------------------------+
| ghi                                             |
+-------------------------------------------------+
```

Regular Expression Syntax

A regular expression describes a set of strings. The simplest regular expression is one that has no special characters in it. For example, the regular expression `hello` matches `hello` and nothing else.

Nontrivial regular expressions use certain special constructs so that they can match more than one string. For example, the regular expression `hello|world` contains the `|` alternation operator and matches either the `hello` or `world`.

As a more complex example, the regular expression `B[an]*s` matches any of the strings `Bananas`, `Baaaaas`, `Bs`, and any other string starting with a `B`, ending with an `s`, and containing any number of `a` or `n` characters in between.

The following list covers some of the basic special characters and constructs that can be used in regular expressions. For information about the full regular expression syntax supported by the ICU library used to implement regular expression support, visit the International Components for Unicode web site.

- ^

  Match the beginning of a string.

```
mysql> SELECT REGEXP_LIKE('fo\nfo', '^fo$');                    -> 0
mysql> SELECT REGEXP_LIKE('fofo', '^fo');                       -> 1
```

- `$`

  Match the end of a string.

```
mysql> SELECT REGEXP_LIKE('fo\no', '^fo\no$');                  -> 1
mysql> SELECT REGEXP_LIKE('fo\no', '^fo$');                     -> 0
```

- `.`

  Match any character (including carriage return and newline, although to match these in the middle of a string, the `m` (multiple line) match-control character or the `(?m)` within-pattern modifier must be given).

```
mysql> SELECT REGEXP_LIKE('fofo', '^f.*$');                     -> 1
mysql> SELECT REGEXP_LIKE('fo\r\nfo', '^f.*$');                 -> 0
mysql> SELECT REGEXP_LIKE('fo\r\nfo', '^f.*$', 'm');            -> 1
mysql> SELECT REGEXP_LIKE('fo\r\nfo', '(?m)^f.*$');            -> 1
```

- `a*`

  Match any sequence of zero or more a characters.

```
mysql> SELECT REGEXP_LIKE('Ban', '^Ba*n');                      -> 1
mysql> SELECT REGEXP_LIKE('Baaan', '^Ba*n');                    -> 1
mysql> SELECT REGEXP_LIKE('Bn', '^Ba*n');                       -> 1
```

- `a+`

  Match any sequence of one or more a characters.

```
mysql> SELECT REGEXP_LIKE('Ban', '^Ba+n');                      -> 1
mysql> SELECT REGEXP_LIKE('Bn', '^Ba+n');                       -> 0
```

- `a?`

Match either zero or one a character.

```
mysql> SELECT REGEXP_LIKE('Bn', '^Ba?n');                    -> 1
mysql> SELECT REGEXP_LIKE('Ban', '^Ba?n');                   -> 1
mysql> SELECT REGEXP_LIKE('Baan', '^Ba?n');                  -> 0
```

- `de|abc`

  Alternation; match either of the sequences `de` or `abc`.

```
mysql> SELECT REGEXP_LIKE('pi', 'pi|apa');                   -> 1
mysql> SELECT REGEXP_LIKE('axe', 'pi|apa');                  -> 0
mysql> SELECT REGEXP_LIKE('apa', 'pi|apa');                  -> 1
mysql> SELECT REGEXP_LIKE('apa', '^(pi|apa)$');              -> 1
mysql> SELECT REGEXP_LIKE('pi', '^(pi|apa)$');               -> 1
mysql> SELECT REGEXP_LIKE('pix', '^(pi|apa)$');              -> 0
```

- `(abc)*`

  Match zero or more instances of the sequence `abc`.

```
mysql> SELECT REGEXP_LIKE('pi', '^(pi)*$');                  -> 1
mysql> SELECT REGEXP_LIKE('pip', '^(pi)*$');                 -> 0
mysql> SELECT REGEXP_LIKE('pipi', '^(pi)*$');                -> 1
```

- `{1}`, `{2,3}`

  Repetition; `{`***n***`}` and `{`***m***`,`***n***`}` notation provide a more general way of writing regular expressions that match many occurrences of the previous atom (or “piece”) of the pattern. ***m*** and ***n*** are integers.

  - `a*`

    Can be written as `a{0,}`.

  - `a+`

    Can be written as `a{1,}`.

  - `a?`

    Can be written as `a{0,1}`.

To be more precise, `a{n}` matches exactly *n* instances of `a`. `a{n,}` matches *n* or more instances of `a`. `a{m,n}` matches *m* through *n* instances of `a`, inclusive. If both *m* and *n* are given, *m* must be less than or equal to *n*.

```
mysql> SELECT REGEXP_LIKE('abcde', 'a[bcd]{2}e');             -> 0
mysql> SELECT REGEXP_LIKE('abcde', 'a[bcd]{3}e');             -> 1
mysql> SELECT REGEXP_LIKE('abcde', 'a[bcd]{1,10}e');          -> 1
```

- `[a-dX]`, `[^a-dX]`

  Matches any character that is (or is not, if `^` is used) either `a`, `b`, `c`, `d` or `X`. A `-` character between two other characters forms a range that matches all characters from the first character to the second. For example, `[0-9]` matches any decimal digit. To include a literal `]` character, it must immediately follow the opening bracket `[`. To include a literal `-` character, it must be written first or last. Any character that does not have a defined special meaning inside a `[]` pair matches only itself.

  ```
  mysql> SELECT REGEXP_LIKE('aXbc', '[a-dXYZ]');                -> 1
  mysql> SELECT REGEXP_LIKE('aXbc', '^[a-dXYZ]$');              -> 0
  mysql> SELECT REGEXP_LIKE('aXbc', '^[a-dXYZ]+$');             -> 1
  mysql> SELECT REGEXP_LIKE('aXbc', '^[^a-dXYZ]+$');            -> 0
  mysql> SELECT REGEXP_LIKE('gheis', '^[^a-dXYZ]+$');           -> 1
  mysql> SELECT REGEXP_LIKE('gheisa', '^[^a-dXYZ]+$');          -> 0
  ```

- `[=character_class=]`

  Within a bracket expression (written using `[` and `]`), `[=character_class=]` represents an equivalence class. It matches all characters with the same collation value, including itself. For example, if `o` and `(+)` are the members of an equivalence class, `[[=o=]]`, `[[=(+)=]]`, and `[o(+)]` are all synonymous. An equivalence class may not be used as an endpoint of a range.

- `[:character_class:]`

  Within a bracket expression (written using `[` and `]`), `[:character_class:]` represents a character class that matches all characters belonging to that class. The following table lists the standard class names. These names stand for the character classes defined in the `ctype(3)` manual page. A particular locale may provide other class names. A character class may not be used as an endpoint of a range.

Character Class Name	Meaning
`alnum`	Alphanumeric characters
`alpha`	Alphabetic characters
`blank`	Whitespace characters
`cntrl`	Control characters
`digit`	Digit characters
`graph`	Graphic characters
`lower`	Lowercase alphabetic characters
`print`	Graphic or space characters
`punct`	Punctuation characters
`space`	Space, tab, newline, and carriage return
`upper`	Uppercase alphabetic characters
`xdigit`	Hexadecimal digit characters

```
mysql> SELECT REGEXP_LIKE('justalnums', '[[:alnum:]]+');       -> 1
mysql> SELECT REGEXP_LIKE('!!!', '[[:alnum:]]+');              -> 0
```

Because ICU is aware of all alphabetic characters in `utf16_general_ci`, some character classes may not perform as quickly as character ranges. For example, `[a-zA-Z]` is known to work much more quickly than `[[:alpha:]]`, and `[0-9]` is generally much faster than `[[:digit:]]`. If you are migrating applications using `[[:alpha:]]` or `[[:digit:]]` from an older version of MySQL, you should replace these with the equivalent ranges for use with MySQL 8.0.

To use a literal instance of a special character in a regular expression, precede it by two backslash (\) characters. The MySQL parser interprets one of the backslashes, and the regular expression library interprets the other. For example, to match the string `1+2` that contains the special + character, only the last of the following regular expressions is the correct one:

```
mysql> SELECT REGEXP_LIKE('1+2', '1+2');                       -> 0
mysql> SELECT REGEXP_LIKE('1+2', '1\+2');                      -> 0
mysql> SELECT REGEXP_LIKE('1+2', '1\\+2');                     -> 1
```

Regular Expression Resource Control

`REGEXP_LIKE()` and similar functions use resources that can be controlled by setting system variables:

- The match engine uses memory for its internal stack. To control the maximum available memory for the stack in bytes, set the `regexp_stack_limit` system variable.
- The match engine operates in steps. To control the maximum number of steps performed by the engine (and thus indirectly the execution time), set the `regexp_time_limit` system variable. Because this limit is expressed as number of steps, it affects execution time only indirectly. Typically, it is on the order of milliseconds.

Regular Expression Compatibility Considerations

Prior to MySQL 8.0.4, MySQL used the Henry Spencer regular expression library to support regular expression operations, rather than International Components for Unicode (ICU). The following discussion describes differences between the Spencer and ICU libraries that may affect applications:

- With the Spencer library, the `REGEXP` and `RLIKE` operators work in byte-wise fashion, so they are not multibyte safe and may produce unexpected results with multibyte character sets. In addition, these operators compare characters by their byte values and accented characters may not compare as equal even if a given collation treats them as equal.

  ICU has full Unicode support and is multibyte safe. Its regular expression functions treat all strings as `UTF-16`. You should keep in mind that positional indexes are based on 16-bit chunks and not on code points. This means that, when passed to such functions, characters using more than one chunk may produce unanticipated results, such as those shown here:

```
mysql> SELECT REGEXP_INSTR('🍣🍣b', 'b');
+--------------------------+
| REGEXP_INSTR('??b', 'b') |
+--------------------------+
|                        5 |
+--------------------------+
1 row in set (0.00 sec)

mysql> SELECT REGEXP_INSTR('🍣🍣bxxx', 'b', 4);
+--------------------------------+
| REGEXP_INSTR('??bxxx', 'b', 4) |
+--------------------------------+
|                              5 |
```

```
+------------------------------+
1 row in set (0.00 sec)
```

Characters within the Unicode Basic Multilingual Plane, which includes characters used by most modern languages, are safe in this regard:

```
mysql> SELECT REGEXP_INSTR('бжb', 'b');
+----------------------------+
| REGEXP_INSTR('бжb', 'b')   |
+----------------------------+
|                          3 |
+----------------------------+
1 row in set (0.00 sec)

mysql> SELECT REGEXP_INSTR('עבb', 'b');
+----------------------------+
| REGEXP_INSTR('עבb', 'b')   |
+----------------------------+
|                          3 |
+----------------------------+
1 row in set (0.00 sec)

mysql> SELECT REGEXP_INSTR('µå周çб', '周');
+-----------------------------------+
| REGEXP_INSTR('µå周çб', '周')       |
+-----------------------------------+
|                                 3 |
+-----------------------------------+
1 row in set (0.00 sec)
```

Emoji, such as the “sushi” character 🍣 (U+1F363) used in the first two examples, are not included in the Basic Multilingual Plane, but rather in Unicode's Supplementary Multilingual Plane. Another issue can arise with emoji and other 4-byte characters when REGEXP_SUBSTR() or a similar function begins searching in the middle of a character. Each of the two statements in the following example starts from the second 2-byte position in the first argument. The first statement works on a string consisting solely of 2-byte (BMP) characters. The second statement contains 4-byte characters which are incorrectly interpreted in the result because the first two bytes are stripped off and so the remainder of the character data is misaligned.

```
mysql> SELECT REGEXP_SUBSTR('周周周周', '.*', 2);
+----------------------------------------+
| REGEXP_SUBSTR('周周周周', '.*', 2)     |
+----------------------------------------+
```

```
| 周周周                                  |
+----------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT REGEXP_SUBSTR('🍣🍣🍣🍣', '.*', 2);
+--------------------------------+
| REGEXP_SUBSTR('????', '.*', 2) |
+--------------------------------+
| ?恭揖恭揖恭揖                  |
+--------------------------------+
1 row in set (0.00 sec)
```

- For the `.` operator, the Spencer library matches line-terminator characters (carriage return, newline) anywhere in string expressions, including in the middle. To match line terminator characters in the middle of strings with ICU, specify the `m` match-control character.

- The Spencer library supports word-beginning and word-end boundary markers (`[[:<:]]` and `[[:>:]]` notation). ICU does not. For ICU, you can use `\b` to match word boundaries; double the backslash because MySQL interprets it as the escape character within strings.

- The Spencer library supports collating element bracket expressions (`[.characters.]` notation). ICU does not.

- For repetition counts (`{n}` and `{m,n}` notation), the Spencer library has a maximum of 255. ICU has no such limit, although the maximum number of match engine steps can be limited by setting the `regexp_time_limit` system variable.

- ICU interprets parentheses as metacharacters. To specify a literal open `(` or close parenthesis `)` in a regular expression, it must be escaped:

```
mysql> SELECT REGEXP_LIKE('(', '(');
ERROR 3692 (HY000): Mismatched parenthesis in regular expression.
mysql> SELECT REGEXP_LIKE('(', '\\(');
+-------------------------+
| REGEXP_LIKE('(', '\\(') |
+-------------------------+
|                       1 |
+-------------------------+
mysql> SELECT REGEXP_LIKE(')', ')');
ERROR 3692 (HY000): Mismatched parenthesis in regular expression.
mysql> SELECT REGEXP_LIKE(')', '\\)');
+-------------------------+
| REGEXP_LIKE(')', '\\)') |
+-------------------------+
```

```
|                        1 |
+--------------------------+
```

- ICU also interprets square brackets as metacharacters, but only the opening square bracket need be escaped to be used as a literal character:

```
mysql> SELECT REGEXP_LIKE('[', '[');
ERROR 3696 (HY000): The regular expression contains an
unclosed bracket expression.
mysql> SELECT REGEXP_LIKE('[', '\\[');
+--------------------------+
| REGEXP_LIKE('[', '\\[') |
+--------------------------+
|                        1 |
+--------------------------+
mysql> SELECT REGEXP_LIKE(']', ']');
+-----------------------+
| REGEXP_LIKE(']', ']') |
+-----------------------+
|                     1 |
+-----------------------+
```