mongoDB

$regex

On this page

Definition

`$regex`

Provides regular expression capabilities for pattern matching *strings* in queries. MongoDB uses Perl compatible regular expressions (i.e. "PCRE") version 8.41 with UTF-8 support.

To use `$regex`, use one of the following syntaxes:

```
{ <field>: { $regex: /pattern/, $options: '<options>' } }
{ <field>: { $regex: 'pattern', $options: '<options>' } }
{ <field>: { $regex: /pattern/<options> } }
```

In MongoDB, you can also use regular expression objects (i.e. `/pattern/`) to specify regular expressions:

```
{ <field>: /pattern/<options> }
```

For restrictions on particular syntax use, see $regex vs. /pattern/ Syntax.

`$options`

The following `<options>` are available for use with regular expression.

Option	Description	Syntax Restrictions

Option	Description	Syntax Restrictions
`i`	Case insensitivity to match upper and lower cases. For an example, see Perform Case-Insensitive Regular Expression Match.	
`m`	For patterns that include anchors (i.e. ^ for the start, $ for the end), match at the beginning or end of each line for strings with multiline values. Without this option, these anchors match at beginning or end of the string. For an example, see Multiline Match for Lines Starting with Specified Pattern. If the pattern contains no anchors or if the string value has no newline characters (e.g. `\n`), the `m` option has no effect.	
`x`	"Extended" capability to ignore all white space characters in the `$regex` pattern unless escaped or included in a character class. Additionally, it ignores characters in-between and including an un-escaped hash/pound (#) character and the next new line, so that you may include comments in complicated patterns. This only applies to data characters; white space characters may never appear within special character sequences in a pattern. The `x` option does not affect the handling of the VT character (i.e. code 11).	Requires `$regex` with `$options` syntax
`s`	Allows the dot character (i.e. `.`) to match all characters *including* newline characters. For an example, see Use the . Dot Character to Match New Line.	Requires `$regex` with `$options` syntax

Behavior

$regex vs. /pattern/ Syntax

`$in` Expressions

To include a regular expression in an `$in` query expression, you can only use JavaScript regular expression objects (i.e. `/pattern/`). For example:

```
{ name: { $in: [ /^acme/i, /^ack/ ] } }
```

You *cannot* use `$regex` operator expressions inside an `$in`.

Implicit `AND` Conditions for the Field

To include a regular expression in a comma-separated list of query conditions for the field, use the `$regex` operator. For example:

```
{ name: { $regex: /acme.*corp/i, $nin: [ 'acmeblahcorp' ] } }
{ name: { $regex: /acme.*corp/, $options: 'i', $nin: [ 'acmeblahcorp' ] } }
{ name: { $regex: 'acme.*corp', $options: 'i', $nin: [ 'acmeblahcorp' ] } }
```

`x` and `s` Options

To use either the `x` option or `s` options, you must use the `$regex` operator expression *with* the `$options` operator. For example, to specify the `i` and the `s` options, you must use `$options` for both:

```
{ name: { $regex: /acme.*corp/, $options: "si" } }
{ name: { $regex: 'acme.*corp', $options: "si" } }
```

PCRE vs JavaScript

To use PCRE supported features in the regex pattern that are unsupported in JavaScript, you must use the `$regex` operator expression with the pattern as a string. For example, to use `(?i)` in the pattern to turn case-insensitivity on for the remaining pattern and `(?-i)` to turn case-sensitivity on for the remaining pattern, you must use the `$regex` operator with the pattern as a string:

```
{ name: { $regex: '(?i)a(?-i)cme' } }
```

Index Use

For case sensitive regular expression queries, if an index exists for the field, then MongoDB matches the regular expression against the values in the index, which can be faster than a collection scan. Further optimization can occur if the regular expression is a "prefix expression", which means that all potential matches start with the same string. This allows MongoDB to construct a "range" from that prefix and only match against those values from the index that fall within that range.

A regular expression is a "prefix expression" if it starts with a caret (`^`) or a left anchor (`\A`), followed by a string of simple symbols. For example, the regex `/^abc.*/` will be optimized by matching only against the values from the index that start with `abc`.

Additionally, while `/^a/`, `/^a.*/`, and `/^a.*$/` match equivalent strings, they have different performance characteristics. All of these expressions use an index if an appropriate index exists; however, `/^a.*/`, and `/^a.*$/` are slower. `/^a/` can stop scanning after matching the prefix.

Case insensitive regular expression queries generally cannot use indexes effectively. The `$regex` implementation is not collation-aware and is unable to utilize case-insensitive indexes.

Examples

The following examples use a collection `products` with the following documents:

```
{ "_id" : 100, "sku" : "abc123", "description" : "Single line description." }
{ "_id" : 101, "sku" : "abc789", "description" : "First line\nSecond line" }
{ "_id" : 102, "sku" : "xyz456", "description" : "Many spaces before     line" }
{ "_id" : 103, "sku" : "xyz789", "description" : "Multiple\nline description" }
```

Perform a `LIKE` Match

The following example matches all documents where the `sku` field is like `"%789"`:

```
db.products.find( { sku: { $regex: /789$/ } } )
```

The example is analogous to the following SQL LIKE statement:

```
SELECT * FROM products
WHERE sku like "%789";
```

Perform Case-Insensitive Regular Expression Match

The following example uses the `i` option perform a *case-insensitive* match for documents with `sku` value that starts with `ABC`.

```
db.products.find( { sku: { $regex: /^ABC/i } } )
```

The query matches the following documents:

```
{ "_id" : 100, "sku" : "abc123", "description" : "Single line description." }
{ "_id" : 101, "sku" : "abc789", "description" : "First line\nSecond line" }
```

Multiline Match for Lines Starting with Specified Pattern

The following example uses the `m` option to match lines starting with the letter `S` for multiline strings:

```
db.products.find( { description: { $regex: /^S/, $options: 'm' } } )
```

The query matches the following documents:

```
{ "_id" : 100, "sku" : "abc123", "description" : "Single line description." }
{ "_id" : 101, "sku" : "abc789", "description" : "First line\nSecond line" }
```

Without the `m` option, the query would match just the following document:

```
{ "_id" : 100, "sku" : "abc123", "description" : "Single line description." }
```

If the `$regex` pattern does not contain an anchor, the pattern matches against the string as a whole, as in the following example:

```
db.products.find( { description: { $regex: /S/ } } )
```

Then, the `$regex` would match both documents:

```
{ "_id" : 100, "sku" : "abc123", "description" : "Single line description." }
{ "_id" : 101, "sku" : "abc789", "description" : "First line\nSecond line" }
```

Use the . Dot Character to Match New Line

The following example uses the `s` option to allow the dot character (i.e. `.`) to match all characters *including* new line as well as the `i` option to perform a case-insensitive match:

```
db.products.find( { description: { $regex: /m.*line/, $options: 'si' } } )
```

The query matches the following documents:

```
{ "_id" : 102, "sku" : "xyz456", "description" : "Many spaces before     line" }
{ "_id" : 103, "sku" : "xyz789", "description" : "Multiple\nline description" }
```

Without the `s` option, the query would have matched only the following document:

```
{ "_id" : 102, "sku" : "xyz456", "description" : "Many spaces before     line" }
```

Ignore White Spaces in Pattern

The following example uses the `x` option ignore white spaces and the comments, denoted by the `#` and ending with the `\n` in the matching pattern:

```
var pattern = "abc #category code\n123 #item number"
db.products.find( { sku: { $regex: pattern, $options: "x" } } )
```

The query matches the following document:

```
{ "_id" : 100, "sku" : "abc123", "description" : "Single line description." }
```