A COMPARATIVE ANALYSIS OF THE USE OF REGEXP AND LIKE OPERATOR IN MYSQL

Dušan Stefanović
Academy of Applied Technical and Preschool Studies, Nis, Serbia,
dusan.stefanovic@akademijanis.edu.rs
Nikola Vukotić
Academy of Applied Technical and Preschool Studies, Nis, Serbia, nikola.vukotic@akademijanis.edu.rs
Slavimir Stošović
Academy of Applied Technical and Preschool Studies, Nis, Serbia,
slavimir.stosovic@akademijanis.edu.rs
Goran Milosavljević
Academy of Applied Technical and Preschool Studies, Nis, Serbia,
goran.milosavljevic@akademijanis.edu.rs

Abstract: This research paper compares the performance of RegExp and LIKE operators in MySQL for searching text data. Research shows which of these two techniques provides better search performance and efficiency. The paper includes an overview of RegExp and LIKE operators, their syntax and characteristics. The research methodology includes the definition of test scenarios, the selection of test data and the implementation of the experimental environment. Parameters considered include execution time as well as implementation complexity. Analysis of the results includes comparing the advantages, disadvantages and functionality of both approaches. Different test scenarios were tested on databases of ten thousand, one hundred thousand and one million rows of user data, in order to evaluate how the number of data being searched affects execution times. The aim of the research is to provide guidelines for choosing the most efficient tool for searching textual data in MySQL. The results of this research will be useful to developers in making decisions about the selection of textual data search tools.
Keywords: RegExp, LIKE operator, MySQL, performance, BRE, ERE.

1. INTRODUCTION

In today's era of big data, the increasing emphasis is on the speed and efficiency of searching big data. Tools such as RegExp (Regular Expressions) and LIKE operators play a key role in data filtering. RegExp allows data to be searched based on various patterns, while the LIKE operator performs the search using the so-called wildcart syntax. The challenge is to determine which of these approaches provides better runtime performance in MySQL databases. The research was conducted with a focus on comparing the performance of RegExp and the LIKE operator using databases with ten thousand rows, one hundred thousand rows, and one million rows. By applying these techniques to the data and recording the execution time for each of the techniques, the obtained results are presented in tabular format. This kind of research has a practical application because it makes it possible to make a decision on the choice of a technique for searching data in the MySQL database. The work will include a detailed description of RegExp and LIKE operators, implementation of tests, analysis of performed tests and comparison of performance in different scenarios.

2. REGEXP

Regular expressions, often known as RegExp (Barnett, 2023; Erwig & Gopinath, 2012; Kahrs, 1999), is a language used to create patterns, find and manipulate text based on mathematically defined sets of symbols and rules for combining them. They are applied in a variety of contexts, including programming languages, command-line interfaces, and text editors (Friedl, 2006). There are basic regular expressions BRE (Basic Regular Expressions) and extended regular expressions ERE (Extended Regular Expressions). It is possible to match the functionality of basic and extended regular expressions, and in some implementations it is practical to use a hybrid syntax that supports elements of both types of regular expressions (Walilko, 2023). A RegExp expression can be composed of the metacharacters shown in Table I, the quantifiers shown in Table II, the ranges shown in Table III, and the character classes shown in Table IV.

TABLE I: REGEXP METACHARACTERS

Template	Template description
.	Matches a single, any character, except a newline
^	Match the beginning of a line or text
$	Match the end of a line or text
\	Neutralization of special meanings of signs
|	Matching character from left or right. If the character on one side matches, the other is ignored

TABLE II: REGEXP QUANTIFIERS

Template	Template description
?	A set of characters that appear once or never
*	A set of characters that appear never, once, or more than once
+	A set of characters that appear one or more times
{x}	The number of times the character is repeated X times
{x, }	The number of times the character is repeated X or more times
{x,y}	Number of repetitions ranging from X to Y times

TABLE III: REGEXP RANGES

Template	Template description
( )	Character grouping
{ }	Defining the number of characters to match
[ ]	Defining the characters to match
[pqr]	Matches any character listed in square brackets
[pqr][xy]	Match p, q or r, followed by x or y
[…]	Any character enclosed in square brackets
[^ …]	Any character not enclosed in square brackets
[az]	A set of characters in the range a to z
[AZ]	A set of characters in the range A to Z
[0-9]	A set of characters representing a digit from 0 to 9

TABLE IV: REGEXP CHARACTER CLASSES

Template	Template description
\s	Any whitespace character (space, tab, newline)
\S	Any character that is not a whitespace character
\w	Any character from the set [a-zA-Z0-9_]
\W	Any non-set character [a-zA-Z0-9_]

\d	Any character representing a digit from [0-9]
\D	Any non-digit character [0-9]

2.1. RegExp for email addresses

$$[\backslash w\backslash.]*@(\backslash w*\backslash.)\{1,10\}\backslash w* \tag{1}$$

In expression 1 $[\backslash w\backslash\backslash.]*$ it finds the characters representing the username, consisting of the characters [a-zA-Z0-9_] and a dot that can be part of the username. In expression 1 the symbol $@$ that separates the username from the domain name. Part of the statement $(\backslash w*\backslash\backslash.)\{1,10\}$ represents a domain name without a Top Level Domain. The top level domain is represented by $\backslash w*$ (Vukićević, 2010).

2.2. RegExp for phone numbers

$$0[0-9]\{2\}[/-][0-9]\{3,4\}-?[0-9]\{3,4\} \tag{2}$$

In expression 2, 0 it indicates that the character string must contain the digit zero at that position. The next two characters $[0-9]\{2\}[/-]$ must be the digits 0 to 9, followed by the "-" sign. After that, a character set $[0-9]\{3,4\}$. represents a group of three or four digits. In the part of the expression $-?$ the sign "-" may or may not appear and $[0-9]\{3,4\}$ it is a group of characters of three or four digits.

3. LIKE OPERATOR

The LIKE operator (Smith, 2023) is a logical operator, designed to determine whether a given string of characters conforms to a specific pattern. It is typically used within a WHERE clause to facilitate searching for a specific pattern within a column's values. Because the LIKE operator finds specific pattern it could be interesting to compare it with RegExp (SyBooks, 2012; Barnhill, 2019). The LIKE operator can be used alone or in combination with wildcards.

3.1. Wildcard %

Within MySQL, the percent symbol (%) functions as a wildcard character, used in conjunction with the LIKE operator to effectively match any character string consisting of zero or more characters and can be used to search for any number of characters at a given position (Nichter, 2022).
The use of the wildcard % with the LIKE operator is shown in Table V.

TABLE V: LIKE OPERATOR WITH WILDCARD %

Template	Template description
'a%'	Finds a value that starts with the character 'a', followed by any other character
'%a	Finds a value that ends with the character 'a'
'%a%	Finds a value that contains the character 'a' at any position
'a%b'	Finds a value that starts with the character 'a' and ends with the character 'b'

3.2. Wildcard _

The _ wildcard character in MySQL is used in conjunction with the LIKE operator to replace just one of any single characters in place of the wildcard character. If it is necessary to replace a large number of characters, it is possible to add a large number of wildcard characters.
The use of the _ wildcard character with the LIKE operator is shown in Table VI.

TABLE VI: LIKE OPERATOR WITH WILDCARD _

Template	Template description
'_a'	Finds a value that contains two characters of which the character in the second position is 'a'
'a_'	Finds a value containing two characters of which the character in the first position is 'a'
'_ _'	Finds any value composed of any two characters

TABLE VII: LIKE OPERATOR WITH WILDCARDS

Template	Template description
'_a%'	Finds any value that has the character 'a' in the second position
'a_%'	Finds a loyalty that has a minimum of two characters and the character 'a' is in the first position
'_ _ %'	Finds a loyalty that consists of at least three or more characters.

3.3. Wildcard combination

If it is necessary to perform a more advanced form search, it is possible to combine wildcard characters in the work with the LIKE operator.
An example of a combination of wildcard characters with the LIKE operator is shown in Table VII.

4. COMPARING PERFORMANCE

In order to determine and compare the performance between RegExp and the LIKE operator, a test was conducted on databases of different sizes and data from columns with different numbers of characters. Testing was performed on databases containing ten thousand rows, one hundred thousand rows, and one million rows. Performance was compared when executing simple queries using only RegExp and LIKE operators, and when using complex queries that, in addition to RegExp and LIKE operators, contain additional operators such as AND, OR and string functions. The simulation included an environment that was identical for each scenario.

4.1. Database with 10 thousand rows

Several test scenarios were conducted over the column 'username', which contains up to ten characters, and 'password', which contains up to 30 characters. The comparison of execution time for the test example over the "username" column is shown in Table VIII, and over the "password" column in Table IX.

TABLE VIII: USERNAME COLUMN

Test scenario	RegExp	LIKE
Search by specific word	3ms	2ms
Search by initial character	5 ms	4ms
Search by last character	4ms	4ms
Search by character at a specific position	6ms	5 ms
Search by pattern that can be in any position	3ms	2ms

TABLE IX: PASSWORD COLUMN

Test scenario	RegExp	LIKE
Search by initial character	4ms	3ms
Search by last character	5 ms	3ms
Search by character at a specific position	5 ms	3ms
Search by pattern that can be in any position	5 ms	4ms

In all the mentioned test examples, over a base of ten thousand rows, the LIKE operator shows slightly better performance of 1ms in terms of execution time, compared to RegExp.

4.2. Database with 100 thousand rows

Several test scenarios were conducted over the column 'username', which contains up to ten characters, and 'password', which contains up to 30 characters. The comparison of the execution time for the test example over the "username" column is shown in Table X, and over the "password" column in Table XI.

TABLE X: USERNAME COLUMN

Test scenario	RegExp	LIKE
Search by specific word	48ms	41ms
Search by initial character	55 ms	45 ms
Search by last character	55 ms	48ms
Search by character at a specific position	55 ms	52ms
Search by pattern that can be in any position	49ms	43ms

TABLE XI: PASSWORD COLUMN

Test scenario	RegExp	LIKE
Search by initial character	49ms	46ms
Search by last character	51 ms	48ms
Search by character at a specific position	55 ms	48ms
Search by pattern that can be in any position	53 ms	46ms

In all the mentioned test examples, over a base of one hundred thousand rows, the LIKE operator shows better performance, by 3 ms to 10 ms in terms of execution time, compared to RegExp.

4.3. Testing with simple queries over a database of one million rows

Since the comparison of performance and execution time, in the case of using a database with ten thousand rows and a database with one hundred thousand rows, gives similar results, testing was carried out on a database with one million rows using simple and complex queries. Several test scenarios were conducted over the 'username' column. A comparison of execution times is shown in Table XII.

TABLE XII: USERNAME COLUMN

Test scenario	RegExp	LIKE
Search by specific word	424 ms	352 ms
Search by initial character	493 ms	408 ms
Search by last character	497 ms	424 ms
Search by character at a specific position	504 ms	410 ms
Search by pattern that can be in any position	434 ms	383 ms

TABLE XIII: PASSWORD COLUMN

Test scenario	RegExp	LIKE
Search by initial character	431 ms	389 ms
Search by last character	451 ms	408 ms
Search by character at a specific position	442 ms	411 ms
Search by pattern that can be in any position	446 ms	420 ms

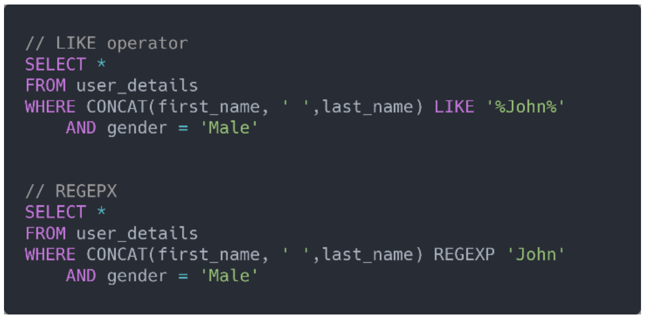
Several test scenarios were conducted over the 'password' column. A comparison of execution times is shown in Table XIII.

It can be concluded that in all the mentioned test examples, over a database with one million rows when using simple queries, the LIKE operator shows better performance in terms of execution time, from 26 ms to 96 ms, compared to RegExp. Based on time averaging over the "username" column, using the LIKE operator is faster by 75 ms (15.9%), and over the "password" column by 35.5 ms (8.02%).

4.4. Testing with complex queries over a million-row database

Testing on a database with a million rows was performed using RegExp and LIKE operators in combination with additional MySQL operators such as AND, OR and string functions.

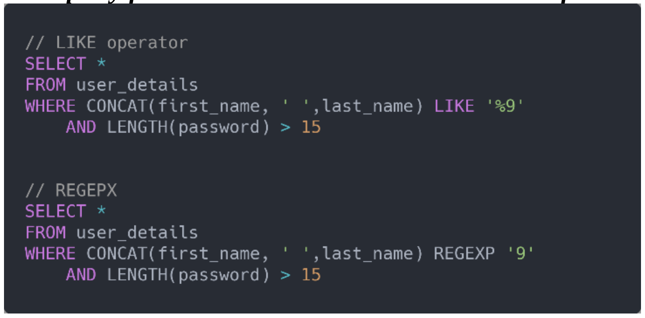
The query for the first test example is shown in Fig. 4.1. The query for the second test example is shown in Fig. 4.2. The query for the third test example is shown in Fig. 4.3.



```
// LIKE operator
SELECT *
FROM user_details
WHERE CONCAT(first_name, ' ',last_name) LIKE '%John%'
    AND gender = 'Male'


// REGEPX
SELECT *
FROM user_details
WHERE CONCAT(first_name, ' ',last_name) REGEXP 'John'
    AND gender = 'Male'
```

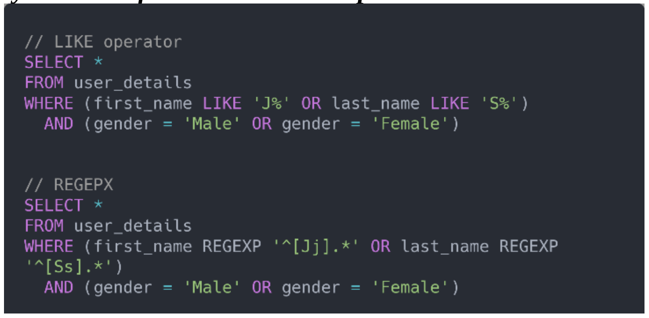
Fig. 4.1. Search query for all male users with "John" in their first and last name



```
// LIKE operator
SELECT *
FROM user_details
WHERE CONCAT(first_name, ' ',last_name) LIKE '%9'
    AND LENGTH(password) > 15


// REGEPX
SELECT *
FROM user_details
WHERE CONCAT(first_name, ' ',last_name) REGEXP '9'
    AND LENGTH(password) > 15
```

Fig. 4.2. Query to search for users who use a password with more than 15 characters



```
// LIKE operator
SELECT *
FROM user_details
WHERE (first_name LIKE 'J%' OR last_name LIKE 'S%')
  AND (gender = 'Male' OR gender = 'Female')


// REGEPX
SELECT *
FROM user_details
WHERE (first_name REGEXP '^[Jj].*' OR last_name REGEXP
'^[Ss].*')
  AND (gender = 'Male' OR gender = 'Female')
```

Fig. 4.3. Query to search for male or female users using a combination of AND and OR operators

After executing all three complex test cases, Table XIV shows the query execution times using RegExp and LIKE operators.

TABLE XIV: COMPLEX QUERIES

Test scenario	RegExp	LIKE
Test example 1	7.1s	7s
Test example 2	13s	11s
Test example 3	31s	30s

Based on time averaging over a million row database using more complex queries, using the LIKE operator is faster by 1.03ms (6.05%).

5. CONCLUSION

Based on research and conducted testing, it was found that the LIKE operator shows better performance compared to RegExp in the processed scenarios. The heighest acceleration 15.09% is in the processing of a larger number of data and simpler queries, while the smallest difference is in the database with ten thousand rows. The LIKE operator provides a faster and more efficient data search when it comes to less complex queries, and also when it comes to more complex queries with a combination of some other operators.

It is important to note that these results may vary depending on the specific context, hardware resources and configuration of the MySQL server.

Related to the testing conducted, MySQL developers who want to optimize their code and search better can get usefull information and guidelines.

REFERENCES

Barnett, B. (2023, July 25). Regular Expressions, Retrieved from https://www.grymoire.com/Unix/Regular.html

Barnhill, B. (2019, December 9). How Regex in SQL Works. Retrieved from https://dataschool.com/how-to-teach-people-sql/how-regex-works-in-sql/

Erwig, M. & Gopinath, R. (2012). Explanations for Regular Expressions. 7212. 394-408. 10.1007/978-3-642-28872-2_27.

Friedl, J. (2006). Mastering Regular Expressions: Understand Your Data and Be More Productive. O'Reilly Media

Kahrs, S. (1999). Regular Expressions - a Graphical User Interface.

Nichter, D. (2022). Efficient MySQL Performance: Best Practices and Techniques. O'Reilly Media

Smith, N. (2023, April 27). MySQL LIKE Operator: 7 Examples and Best Practices. Retrieved from https://blog.devart.com/mysql-like-tutorial.html

SyBooks Online (2012, January 1). SQL Anyware 12. Retrieved from https://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.help.sqlanywhere.12.0.1/dbreference/like-regexp-similarto.html

Vukićević, N. (2010, October 23). Regularni izrazi - Napredna pretraga teksta. Retrieved from https://www.codeblog.rs/clanci.php?p=regularni_izrazi

Walilko, A. (2023, August 21). What are Regular Expressions. Retrieved from https://www.liquidweb.com/kb/what-are-regular-expressions/