Testing the Execution Speed in MySQL – query vs view

Vladimir Milenković
Akademija strukovnih studija Šumadija
Odsek Trstenik
Trstenik, Srbija
vlmilenkovic@yahoo.com

Goran Miodragović
Academy of Professional Studies Sumadija
Department in Trstenik,
Trstenik, Serbia
gmiodragovic@asss.edu.rs

Abstract—**This paper presents research on the topic of query execution speed in MySQL database. We will answer the question of whether there is a faster way of obtaining the desired results than the classic query execution, or whether there is a better alternative solution. Our scenario consists of three phases: running a specific query in three ways, recording the results of each of the variations, and discussing the results obtained. The three ways mentioned will be views, queries, and queries with indexed columns.**

Keywords- view; query; index

I. INTRODUCTION (HEADING 1)

MySQL as a member of OpenSource products is characterized as open code software. The term open source refers to software that can be freely executed, copied, distributed and modified. Quality of OpenSource products is complitly heterogeneous. A database is one of the most important components of almost every application today. When working with databases, you have to take care of a lot of things. Some examples include designing and creating the database, maintaining it, managing backup and recovery, and measuring performance. MySQL, although the world's most popular relational database, is not sufficiently optimized. The most common choice of every user is to not change anything and work on already predefined settings without any deeper interest in any changes. That changes to MySQL database settings or ways that it works would improve the operation of the database itself or speed up the execution of given queries, which would greatly shorten data processing time. By doing so, they would also save the resources that enable this type of processing and use them in some other way.

Further on in the paper, we will test the speed of execution of given queries on tables with smaller and larger amounts of data, using views, queries and queries with indexed columns of the mentioned tables. As a relevant query execution time, we will monitor reports through phpMyAdmin after the success of query completion.

II. EASE OF USE

A paper with related work on Optimizing MySQL Queries and Combining DDL Queries [1] describes optimizing the configuration of MySQL server to improve its performance. One of the ways is to manually set it up out of the box, if we add to the configuration file:

Innodb_buffer_pool_size = 1G # (change to around 50%-70% of full RAM-a amount) innodb_log_file_size = 256M innodb_flush_log_at_trx_commit = 1 # can change to 2 or 0 innodb_flush_method = O_DI

Furder on word is about help optimization tools like Variable Inspector for Ubuntu Systems, MySQL Tuner: Percona Toolkit to Identify Duplicate Indexes and for unused indexes. If we are logging slow queries, we can run the utility and it will check if these queries are using indexes on the tables and how. It also describes how to detect and track database bottlenecks through slow query logging:

pt-index-usage /var/log/mysql/mysql-slow.log

slow_query_log = /var/log/mysql/mysql-slow.log long_query_time = 1 log-query-not-using-indexes = 1

The MySQL optimizer will use statistics based on table queries to select the best index to execute the query. It works quite simply, based on built-in statistical logic. To ensure that the correct (or incorrect) key is used, we can use the FORCE INDEX, USE INDEX, and IGNORE INDEX keywords in our query. To see how fast our MySQL is, we can enable profiling to take a closer look at MySQL queries. This can be done by issuing the command set profiling=1, after which we have to run show profiles to see the result. We can also enable and configure caching.

Finally, regarding index level measurement, it concludes that the use of indexes in tables significantly increases the speed of processing and generation of DBMS responses to entered queries. MySQL constantly "measures" the level of

index and key usage in each database. To get the given value, use the query:

SHOW STATUS AS "handler_read%"

SHOW STATUS AS "handler_read%"

As a result, we are interested in the value in the Handler_read_key row. If the number indicated there is small, it means that indexes are almost never used in this database. And this is bad.

Another related work is about „Results of a comparison of the responsiveness of open source and commercial methods of database organization“ [2]. Response speeds between centralized and distributed databases were tested. In both cases of the mentioned databases, the commercial solution shows significantly better performance. The speed ratio of open source and commercial solutions is: 3.78:1.

Based on all the measurements and the analysis, several conclusions can be drawn:

- Centralized databases generally have an advantage in terms of performance since data is pulled from only one server.
- The weakness of centralized databases is security. The entire database is in one place, so regular back-up is necessary to ensure the security of the database.
- Distributed databases reduce the individual load on the servers themselves, but, on the other hand, it takes a little longer time to display the data, which depends on the network environment and network load.
- Distributed systems are much more powerful in terms of capacity than centralized ones. Expanding system capacity is reduced to adding a new node to a distributed system, which is a much simpler and cheaper procedure than replacing a centralized system with a larger one.
- Open source solutions are, from the economic point of view, more profitable, but during testing the speed was lower compared to Microsoft solutions.
- Also, commercial solutions offer the possibility of technical support, which is not the case with open source.

III. Experimental Evaluation Query Vs. View

The hardware and software characteristics of the machine on which the queries were run are:

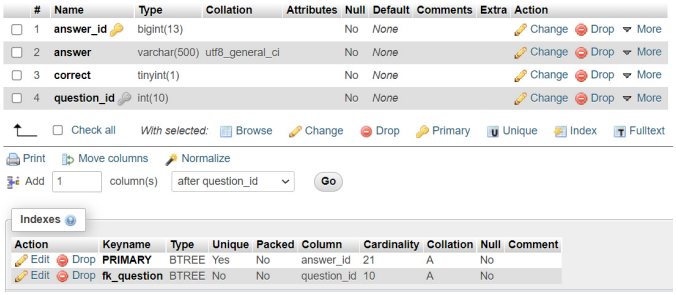
Intel(R) Pentium(R) CPU 4417U @ 2.30GHz 2.30 GHz, 16.0 GB (15.9 GB usable),

Windows 11 Enterprise 64-bit operating system, x64-based processor.

The basic idea behind this experiment is to systematically reduce data processing times, either by classic execution of queries or in some alternative way, by all means, through a series of cyclical reexamination of parts of the execution process itself. In further work, we will use the tables from the moodle database, which is described in my master work on the topic „Design and implementation of the database of the e-learning platform“. For the experimental environment, we will use tables with a smaller amount of content, and later we will monitor the execution time on tables with a larger amount of data. With classical queries for data selection in several ways, following the times of their execution, we will determine which way is the fastest and most rewarding to use.

A. Small table

As a first example, we will use the answers table, whose structure (Figure 1) and data (Figure 2) are as follows:



#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	answer_id	bigint(13)			No	None			Change Drop More
2	answer	varchar(500)	utf8_general_ci		No	None			Change Drop More
3	correct	tinyint(1)			No	None			Change Drop More
4	question_id	int(10)			No	None			Change Drop More

Check all With selected: Browse Change Drop Primary Unique Index Fulltext

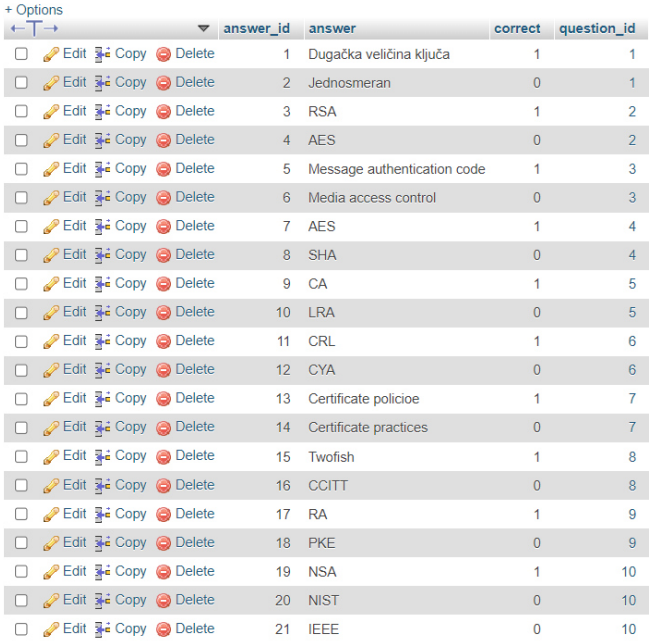
Print Move columns Normalize

Add 1 column(s) after question_id Go

Indexes

Action	Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
Edit Drop	PRIMARY	BTREE	Yes	No	answer_id	21	A	No	
Edit Drop	fk_question	BTREE	No	No	question_id	10	A	No	

Figure 1. Table structure



+ Options

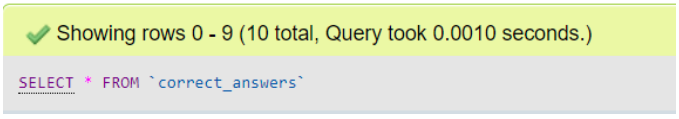
	answer_id	answer	correct	question_id
Edit Copy Delete	1	Dugačka veličina ključa	1	1
Edit Copy Delete	2	Jednosmeran	0	1
Edit Copy Delete	3	RSA	1	2
Edit Copy Delete	4	AES	0	2
Edit Copy Delete	5	Message authentication code	1	3
Edit Copy Delete	6	Media access control	0	3
Edit Copy Delete	7	AES	1	4
Edit Copy Delete	8	SHA	0	4
Edit Copy Delete	9	CA	1	5
Edit Copy Delete	10	LRA	0	5
Edit Copy Delete	11	CRL	1	6
Edit Copy Delete	12	CYA	0	6
Edit Copy Delete	13	Certificate policioe	1	7
Edit Copy Delete	14	Certificate practices	0	7
Edit Copy Delete	15	Twofish	1	8
Edit Copy Delete	16	CCITT	0	8
Edit Copy Delete	17	RA	1	9
Edit Copy Delete	18	PKE	0	9
Edit Copy Delete	19	NSA	1	10
Edit Copy Delete	20	NIST	0	10
Edit Copy Delete	21	IEEE	0	10

Figure 2. Tabel data

We can create the view with the following simple code:

```
CREATE
VIEW `correct_answer_2` AS
SELECT * FROM `answers` WHERE `correct` = 1
```

Now that we have the view created, by simply clicking on it we get the results of the query execution:



Showing rows 0 - 9 (10 total, Query took 0.0010 seconds.)

```
SELECT * FROM `correct_answers`
```

Figure 3. Result of SELECT query

We can see that the query execution time [3] through the view is 10 milliseconds.

And now let's look at the execution of the query through MySQL syntax:

```
SELECT * FROM `answers`
WHERE `correct` = 1
```

The result of execution is:

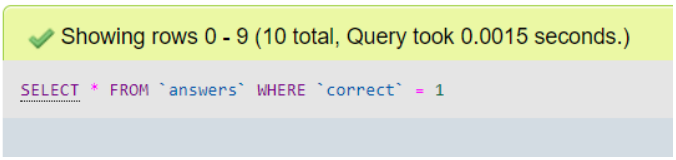


Figure 4. Result of execution

We will notice that the query execution time has increased to 15 milliseconds. It doesn't seem like a significant difference to the eye, but if we consider the existence of tables with millions of records, the percentage difference has a very big meaning when it comes to the need for faster application response.

Further on, the question arises, can this work even faster? Now let's look at the description of the used table with the following code:

```
EXPLAIN SELECT * FROM `answers` WHERE `correct` = 1
```

The result is:

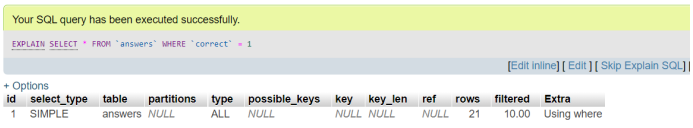


Figure 5. Result of execution

It's time to introduce the concept of index [4]. Indexes have a big impact on query execution speed, just assigning a PRIMARY key is not enough. To add an index to MySQL [5], we use the CREATE INDEX command. There are several types of indexes, for example, FULLTEXT is used for full-text searches, while UNIQUE is used for specific stored data.

We add indexes as follows using the syntax:

```
CREATE INDEX index_correct ON answers(correct);
```

This is how we create an index on the answers table on the *correct* column.

If we now check with the aforementioned EXPLAIN command we will get the following:

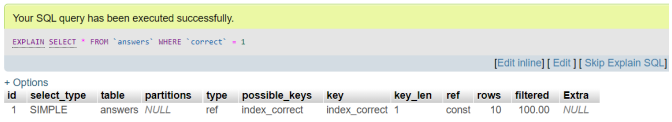


Figure 6. Result of execution

Now if we check the execution time of the query on the indexed table we will notice the following:

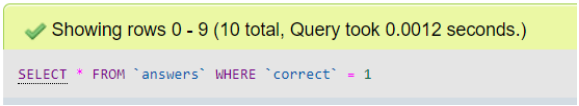


Figure 7. Result of execution

We can see that the query was executed faster using the index on the table. The time of execution now is 12 milliseconds.

And what about those "bigger" tables? Is the situation the same? In order to answer these questions let's move forward to out next stage.

B. Large table

In our database, we will create another table called *students1*, which is structurally a copy of the *students* table, and import a larger amount of data (students data in our case).

EXPLAIN command will show us following:

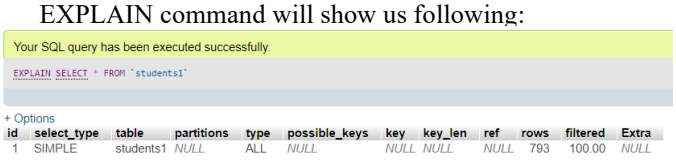


Figure 8. Result of execution

We see that the mentioned table contains 793 rows. We will connect the *students1* table with foreign keys to the table *courses* in order to use the following code to test the execution speed of queries related to which students have chosen courses on their study years [6].

```
ELECT `index_id`, `sname`, `ssurname`, `cname` FROM
`students1`
INNER JOIN `courses` ON `courses`.`study_year` =
`students1`.`study_year`
WHERE `chosen` = 1 LIMIT 200;
```

We will also create a corresponding View:

```
CREATE VIEW chosen_courses
AS SELECT *
FROM students1, courses
WHERE students1.study_year=courses.study_year
AND chosen=1;
```

whose execution time is:

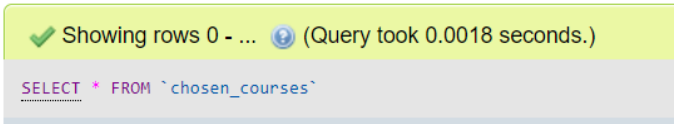


Figure 9. Result of execution

Query we will use is:

```
SELECT `index_id`, `sname`, `ssurname`, `cname`
FROM `students1` INNER JOIN `courses` ON
`courses`.`study_year` = `students1`.`study_year` WHERE
`chosen` = 1
LIMIT 200
```

The execution time of a classic database query is:

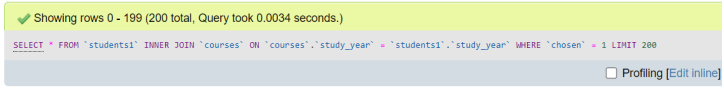


Figure 10. Result of execution

Based on the previous example, if we index the data column with the size of 4 first syllables:

ALTER TABLE `moodle`.`students1` ADD INDEX `index_sname` (`sname`(4));

we will get the execution time:

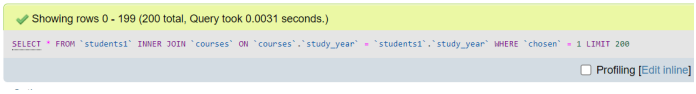


Figure 11. Result of execution

IV. EXPERIMERNTAL RESULTS

We can visually display the results obtained from tables with a smaller amount of data as follows through Figure 12. :

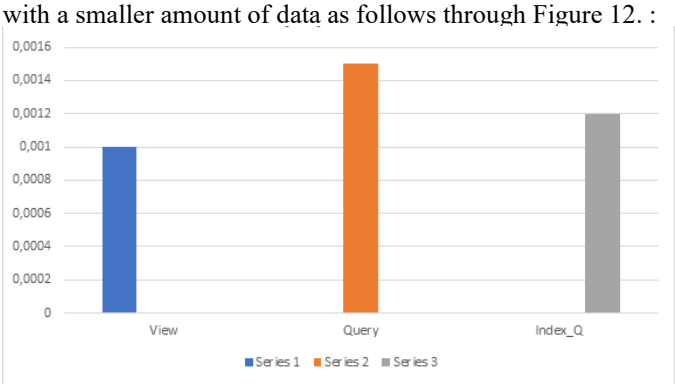


Figure 12. Small table results in sec

Now let's test the multiple execution of the same queries. We will take as a test example 10 executions each as can be seen in Table 1. and then look at the Figure 13. for results of the query execution speed.

TABLE I. EXECUTION TIME IN SEC

	1	2	3	4	5	6	7	8	9	10
View	0,0008	0,0009	0,0009	0,0015	0,0013	0,0010	0,0009	0,0010	0,0016	0,0009
Query	0,0030	0,0014	0,0014	0,0013	0,0012	0,0027	0,0015	0,0012	0,0016	0,0014
Index_Q	0,0011	0,0014	0,0016	0,0013	0,0012	0,0009	0,0008	0,0013	0,0010	0,0008

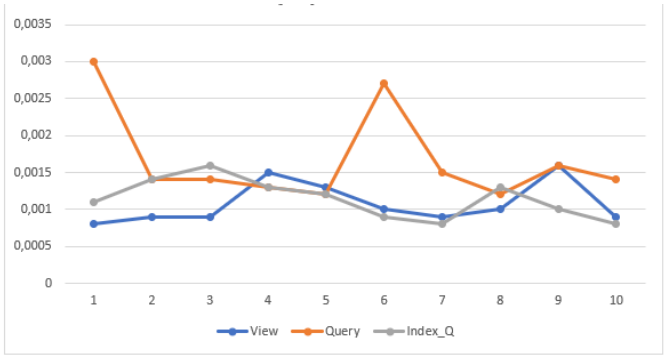


Figure 13. Example execution times in sec

Although there are variations in query execution times, we can generally conclude, as evidenced by the average time, that a View is the best option when it comes to tables with a small amount of data.

As for tables with a larger amount of data, the obtained results are as follows on Figure 14:

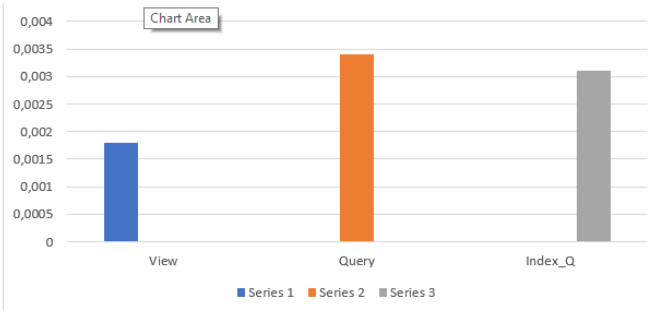


Figure 14. Large table in sec

Now let's test the multiple execution of the same queries. We will take as a test example 10 executions each as can be seen in Table 2. and then look at the Figure 15. for results of the query execution speed.

TABLE II. EXECUTION TIME IN SEC

	1	2	3	4	5	6	7	8	9	10
View	0,0018	0,0025	0,0023	0,0022	0,0017	0,0027	0,0026	0,0019	0,0019	0,0018
Query	0,0034	0,0048	0,0046	0,0045	0,0067	0,0049	0,0031	0,0033	0,0044	0,0039
Index_Q	0,0031	0,0029	0,0035	0,0042	0,0019	0,0019	0,0033	0,0022	0,0025	0,0034

Based on the graphically displayed values, we can notice that the query execution time values from the View are generally significantly lower than the other two cases. Therefore, we can conclude that the display of queries created by the view of virtual tables has a faster response time, which means that our application will work faster and return the desired data to us faster.

CONCLUSION

Research has shown that MySQL is not only a technological product of a given moment, but also a product that is constantly evolving. Successful management of MySQL query performance can only be achieved with a proactive plan. Many problems can be identified and solutions determined in advance. Performance management is an iterative process, because after changing the values of the observed parameters, the operation of the system is monitored again. By measuring performance (benchmarking), it is possible to optimize tables and servers of the MySQL database. It is recommended that, for the needs of the considered system, changes and optimization of queries that last longer than a tenth of a second be made.

In MySQL, the simplest query cost metrics are execution time, number of rows examined and number of rows returned. None of these metrics is a perfect way to measure query cost, but they reflect roughly how much data MySQL must access internally to execute a query and translate approximately into how fast the query runs. All three metrics are logged in the slow query log, so looking at the slow query log is one of the best ways to find queries that examine too much data.

Mysql Indexes allow us tell Mysql databases to index specific columns. This works similar to how regular indexes work in books. Similarly, when DB tables are indexed, Mysql searches in the index column to return the data. This is faster since it does not crawl through each row, but instead uses binary search to get the results.

Views can perform well if you generally keep them simple. they are very good for further usage like doing

analysis, presenting data to user as you can use views to hide table columns from users by granting them access to the view and not to the table itself. This helps enhance database security and integrity. In terms of View performance, we could see that the times obtained in our environment were the best of the three query categories tested.

FURTER WORK

As for some future work, they could look at the benefits of stored procedures, functions and triggers. Repeating queries that do the same thing with variable parameters can be extremely tedious. Automation of the same processes can be the salvation that every IT expert strives for.

A stored procedure is a named database object and is stored on the server side where it is executed, and only the results are passed to the client. When granting privileges, it is sufficient to grant the privilege to start the procedure; it is not necessary to grant special authorizations for individual tables used within it. They are very powerful and all operations from DDL and DML can be performed through them, such as, for example, creating a table, executing an UPDATE statement over several tables, inserting, deleting data, but also setting values (SET) as well as accepting a transaction. (COMMIT) or restoring the database to its previous state (ROLLBACK).The procedure itself can return parameters, result set, code and create cursors. It can also contain input parameters, local variables (variables), numeric and character operations, assignment operations, SQL operations, and logic to control the flow of execution. What a stored procedure cannot do is that it cannot return the result in tabular form. Stored procedure has in and out parameters and that's it .. there is no "table" as a parameter. What can be used for the stored procedure to return the table is to create a temporary table in the stored procedure (create temporary table t1 ...) and then fill it with the result and then after calling the procedure drop the same (or at the beginning of the procedure put a drop it first if it exists).

Important features of Stored Procedure, SQL Function, and Trigger [7]:

A. *Executable*

- *Store procedure:* We can execute the stored procedures when required.
- *Function:* We can call a function whenever required. Function can't be executed because a function is not in pre-compiled form.
- *Trigger:* Trigger can be executed automatically on specified action on a table like, update, delete, or update.

B. *Calling*

Stored procedure: Stored Procedures can't be called from a function because functions can be called from a select statement and Stored Procedures can't be called from. But you can call Store Procedure from Trigger.

```
CREATE TRIGGER tri ON tbl FOR INSERT AS EXEC
mysp
```

Function: Function can be called from Store Procedure or Trigger.

```
CREATE TRIGGER bu_trigger
BEFORE UPDATE
ON tbl1 FOR EACH ROW
BEGIN
 CALL myFunction (*.* );
```

Trigger: Trigger can't be called from Store Procedure or Function.

C. *Parameter*

Store procedure: Stored Procedures can accept any type of parameter. Stored Procedures also accept out parameter.

Function: Function can accept any type of parameter. But function can't accept out parameter.

Trigger: We can't pass a parameter to trigger.

D. *Return*

Store procedure: Stored Procedures may or may not return any values (Single or table) on execution.

Function: Function must return any value.

Trigger: Trigger never return value on execution.

Proper use of these features of mysql can significantly contribute to the acceleration of work and business processes and save a lot of time for both developers and users. Although they initially take a little more time to make, in the long run the time spent pays off.

REFERENCES

[1] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982]. „Optimizing MySql queries. Combining DDL queries.", https://gtavrl.ru/bs/optimizaciya-mysql-zaprosov-obedinenie-ddl-zaprosov-kogda/ , accessed August 2022.

[2] Borivoje M., Srđan J., Lepomir P., Dejan K., „Results of a comparison of the responsiveness of open source and commercial methods of database organization", https://infoteh.etf.ues.rs.ba/zbornik/2012/radovi/RSS-3/RSS-3-3.pdf , accessed August 2022.

[3] SentinelOne, „How to Measure MySQL Query Time: A Detailed Look", https://www.sentinelone.com/blog/how-to-measure-mysql-query-time/ , accessed August 2022.

[4] TechnoTrax, „Mysql Index Tutorial", https://www.youtube.com/watch?v=MS70fZbOuqo , accessed August 2022.

[5] „MySQL indexes. Use of indexes in MySQL. Cycle on the principles of working with MySQL" , https://mirhat.ru/bs/dressing-room/mysql-indeksy-ispolzovanie-indeksov-v-mysql-cikl-po-principam/ , accessed August 2022.

[6] „Mysql multiple queries in one. Multiple SELECT COUNTs in a single MySQL query. Optimizing MySQL queries", https://sukachoff.ru/bs/virusy/mysql-neskolko-zaprosov-v-odnom-neskolko-select-count-v-odnom-zaprose-mysql-optimizaciya/ , accessed August 2022.

[7] Jangid, Gajendra, "Stored Procedure, SQL Function, Triggers In Brief", https://www.c-sharpcorner.com/blogs/about-store-proc-function-trigger-in-brif , accessed August 2022.