Higher Education Diploma in Software Engineering with emphasis in Web Programming
June 2021

Filtering techniques

Let the user get the message in time

Areon Lundqvist

Faculty of Computing, Blekinge Institute of Technology, 371 79 Karlskrona, Sweden

This thesis is submitted to the Faculty of Computing at Blekinge Institute of Technology in partial fulfilment of the requirements for the degree of Higher Education Diploma in Software Engineering with emphasis in Web Programming. The thesis is equivalent to 20 weeks of full time studies.

The authors declare that they are the sole authors of this thesis and that they have not used any sources other than those listed in the bibliography and identified as references. They further declare that they have not submitted this thesis at any other institution to obtain a degree.

Contact Information:
Author(s):
Areon Lundqvist
E-mail: arba20@student.bth.se

University advisor:
Handledare Marie Grahn
Department of Computer Science

Faculty of Computing
Blekinge Institute of Technology
SE–371 79 Karlskrona, Sweden

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

Abstract

As developers of applications or some sort of data handling, we are obligated to contract some kind of data. Filtering techniques have a lot of power when it comes to reducing the unnecessary data back to the client. With these techniques the developer can implement a better performance of the filtering and end up speeding up the process of loading into the sites. And ultimately increasing the user's experience of the software, which increases the quality of the application.

A set of limitations were formed to provide the developing environment. Then we had to adjust and create what files the techniques needed to perform against. Some of the selected technologies had gotten deeper research behind them which led to a conclusion on what performed better within the technique.

The results gave a clear measurement that then got concluded with the other techniques. With smaller sized projects the technique did not matter too much as the difference was ~0.19s between the lowest and highest results. While going to a mid sized project it gave a clearer sign that AWK technique started to lose quality. For larger projects it showed a potential overlap for JavaScript to pass both AWK and MySQL in further studies.

Conclusion is that the JavaScript filtering technique overall is better used for the set environment. For smaller and larger size of projects used.

Keywords: filter, performance, techniques, performance, quality

Contents

Contents **4**

1 Introduction **6**
1.1 Background 6
1.1.1 MySQL 6
1.1.2 JavaScript 7
1.1.3 AWK 7
1.2 Focus and scope 7
1.3 Value and goals 8

2 Focus areas **9**

3 Metod **11**
3.1 Overview 11
3.2 Limitation 11
3.3 System and isolation 11
3.4 Iterating 12
3.5 The data and what to filter 12
3.6 RQ1 13
3.7 RQ2 och RQ3 14
3.8 Litteratur method 14

4 Literature review **15**
4.1 Literature search 15
4.2 Research 16
4.2.1 MySQL 16
4.2.2 AWK 18

5 Results **20**
5.1 MySQL 20
5.1.1 Technic performance 20
5.2 JavaScript filter 21
5.2.1 Technic performance 21
5.3 AWK 21
5.3.1 Technic performance 21
5.4 Comparison 22

6 Analysis and discussion **24**
6.1 RQ1 24
6.2 Isolation 24
6.3 RQ2 and RQ3 25
6.4 Indexing 25
6.5 AWK 26
6.6 Literature search 26

Conclusions 27

Validity threats 28

Future work 29

References 30

Appendix 32

1 Introduction

1.1 Background

As developers of applications or some sort of data handling, we are obligated to contract some kind of data. Usually there is an Application Program Interface, API [33], that takes a request from the client, then retrieves some data and sends it back. If we would just contract the data and send it back to the client we could potentially send too much unnecessary data that later gets sorted out to its right place on the webpage.
This is where filtering techniques come in. But it is not as easy as just picking the first thing that comes to mind. The techniques have different usable areas and are not always applicable on every environment.

If we just pick out the amount of data from a smaller perspective. Sure it does not hurt when counting a couple of rows of a database to test. Then it is quite hurtless on resources and sending over all the data. But when we are starting to get a bit real and coming up to a couple of thousands of rows of a log file, then we have to take into account different ways to handle the amount of data we actually need to send over. In even larger states, over millions of rows, we do not only have to handle the resources. We also have to look into the amount of time it takes to handle the loading of the data and see if there are any differences between the techniques.

There is not a lot of research about specific filtering techniques for API or how to handle the data most efficiently. If the developer does not have a specification or documentation of sorts then the filtering technique can become an overlooked concept. The implementation can then become a personal favorite or the first best found after a google search.

For a developer to be able to perform a faster filtering speed also means it increases the time it takes for the software to show its relevant information. This can lead to loss of interest for the user experience and that can lead to that user to click away, which leads to potential loss in money spent on the site[14]. With increased performance we also reduce the need of resources to process these activities that a, e.g. API, will be needed to perform the filtering. The research also shows a decrease in quality from the user's experience when faced with, *what they thought*, a slower loading to a website.

The techniques that were selected came from the literature search section of this report. Where they were selected from earlier experience and also surveys found with large responses of developers favorite programming environments.

1.1.1 MySQL

MySQL[21] is an open-source relational database system. With MySQL you can store data within tables and structure the content with columns. Each column can be specified with a specific data type and have optional attributes. For our purpose we want to store the data within a table and then use MySQL queries to extract the data. With the query we can specify what specific we want to extract with optional tags such as “WHERE” and “LIKE”.

```
await db.promise().query("SELECT * FROM table_name WHERE column_name LIKE '%searchWord%'")
```

Figure-1: Example of a MySQL query. Asynchronous example of a query call within the npm mysql2 package.

In **Figure-1** we can see the example of a query that extracts all data rows that contain the search word in a specified column. We can extract all columns with "*" (asterisk) and if we add a WHERE option we can make a condition that checks. E.g, that "column_name" column contains a search word anywhere within the text.

1.1.2 JavaScript

The JavaScript built-in function[17] "*.filter()*" is quite simple. When called it takes the given array and loops through every item in the array. It provides a callback function once for each element in the array, then constructs a new array with the elements that passes the conditions given. The ones that do not pass are not passed into the new array. It does not change the original array.

1.1.3 AWK

AWK[18] is a standard filtering language to most Unix-like operating systems. It was used in the vLinux course at BTH where we filtered a log file and created a json file that would come to later use. We wanted to see if it can hold up to the environments created and see how well it would perform compared to the other techniques.

AWK uses a regex[29] pattern to match against a row of text. As it is mostly used as a command line tool it has to be either imported to the develop environment, unless the developer is able to handle files. Most uses of AWK would be in the commandline to filter out the data grabbed from a file.

1.2 Focus and scope

This study is focusing on testing different filtering techniques. This will include three techniques of different kinds of main methods. They will be evaluated first by their own brackets of methods to get the most value to our developing environment and set limitations. With these evaluations within the techniques; there will be research on different methods to bring forward the techniques potentials. But also figure out how they work in the limited environment we give it.

To evaluate the best performance of these techniques we will add the techniques into a NodeJs[34] environment. NodeJs is a development platform for executing JavaScript code server-sided. With this we can set up a filtering test against a sampled sized data and measure the time it took to complete the filtering task.

We limit these techniques to NodeJs and JavaScript as they are the most favorite programming languages as of 2021[16] by this article. This will narrow the amount of possibilities, while also giving an out reach of many developers to be able to recreate the tests.

We also need to isolate the technique from variables such as CPU, software interfaces and internet, we will perform the test within docker containers[Appendix B]. A docker container is a way to seperate software infrastructure and isolate the program's execution to test.

1.3 Value and goals

After this study the goal is that a developer should be able to implement its own limitations with the same performance tests. Then further develop and take the best performance for the project. Hopefully increasing the speed as well as reduction overall of the resources needed for the actions.

The developer should also feel comfortable with testing out their own set of filtering techniques. Either taking pieces from this study and setting up their own test environment for the different techniques. The evaluation process is a healthy way to process the information given from the measurements to visually see the different performance of each technique. Not just do one test and compare it to the other values from the other techniques and judge from that. But to fully go through with isolation and setting up a more “real” environment that can be measured for a more realistic testing.

2 Focus areas

This study is going to research what is a more fitting technique for developers when it comes to a given environment. What tools can be used to efficiently receive data that is filtered to satisfy a given requirement. This study will take different techniques and test them within the given environment and then evaluate its performance. The performance that will be measured in this report is time based.

Focus areas:

- What tools do developers use to filter
- Can selection of technique increase efficiency
- How can the developer evaluate the techniques

RQ1 What filtering technique is appropriate to use for the developer? (i.e. small and large project differences)

The purpose of RQ1 is to get a view of what filtering technique is most relevant to the developers needs in a set data management. It also checks different environments and how a filtering technique fits into that given environment. Here we also try to get an overview to see if a technique is more suitable for bigger sizes data compared to the lower size and differentiate the ones fitting to better suited size. This report takes a look at five sizes of data that ranges from a smaller and larger size selection which is the “project” sizes. This can be valued differently from project to project, but is a generalization for this report to show a difference in each technique's performance.

RQ2 How does the technique affect the performance when filtering the data?

The purpose of RQ2 is to get measurements on how effective the techniques themselves are. We can tell how efficient the technique alone is with the given data. With this we can measure the performance of the given techniques and evaluate it for a better understanding on what technique is a clearer choice when picking a technique to work with. Focusing more on the given technique rather than the whole picture.

RQ3 How does the function affect the performance that is filtering the data?

The purpose of RQ3 is to get measurements on how effective the function of the technique is. Separating the technique itself and measuring the whole function needed to set up to be able to use the technique, that is this reports definition of the function and its performance.

It is a lot like RQ2, but we differentiate it with the performance of the function and not just the technique. We want to measure if the function is dependent on other things rather than just

the technique itself. Are there any factors that are in play that make the functions performance fluctuate.

The goal is to get a bigger picture into what filtering tool is most efficient overall. But also see if there is one more suitable for a smaller- and/or bigger project. With these evaluations the developer should be able to put up a few techniques of their own if not choose one between the one displayed in this study. And then evaluate on their own to conclude a filtering technique that fits its current requirements and the environment.

Possible future evaluation and analysations of other techniques for future work will be easier to implement and evaluate with this study in mind and to use as comparison. But to dive deeper into specified reasons and how different techniques can work together to further improve the efficiency in performance will not be evaluated in this study. With that said, we are still hoping to get a deeper understanding and a brother knowledge of the different filtering techniques.

3 Metod

3.1 Overview

Testing methods to get the techniques performance was performed on five different sizes of row counts of data. A selection of three different kinds of techniques were tested and evaluated which one had better performance efficiency. The selected techniques of filtering were conducted during the literature search and also evaluated between different versions of these techniques.

3.2 Limitation

With RQ1 we had to make some limitations to our research. With the research from the literature a conduction of JavaScript and its environments were selected. With this we have limited the study to not spiral out of hand. First we had to limit the amount of stuff that can fluctuate the measurements, such as sending requests between client and API. We have simulated it by having a function call that awaits a response from another function. It is not fully integrated, but it works as a reliable way to reduce errors in the measurements.

Second, we limited the environment and how flexible it can be. The filtering should not be manually executed from a command line interface, eg, AWK commands in Linux CLI. We want to be as flexible as possible but will act as if we have a client side sending a request that then receives the data.

The flexible part is also important, usually we can set up different kinds of methods and request endpoints that retract specific information that is already set. Like a column in a table of a database. But we want to be able to search for the whole word in this study, mostly because of the flexibility of the technique.

3.3 System and isolation

To make the measurements tests isolated and be able to be reconstructed by future studies we need an isolation of the environment. That is where a Virtual Machine (VM) comes in. This study uses the program called “VirtualBox”[1] with Debian ISO-image[2]. The machine contains settings of: *OS: Linux 2.6 / 3.x / 4.x (64-bit), Base Memory: 4096 och Processor: 6*.

To further isolate the program's execution to potential variables that can be affected by the VM, the choice of using Docker[3] containers were selected to further isolate the executions. With docker we can seperate the applications infrastructure[4] and execute the function with easier executions. Instead of setting up a fully working API that needs to send the data between components. With the docker containers we limit the amount of CPU[5] the single container uses, with this we make sure all of the containers use the same amount of cpu.

3.4 Iterating

The measurements of time performance for each filtering were collected and saved to an excel sheet[Appendix A] and then evaluated in the analysis and discussion chapter of this study. We chose to print out the time into the console and then copy paste it into the excel ark. We have made the programs asynchronous[6]. It awaits for the function to complete before continuing with the next execution of the function. Each execution of the test uses a loop of 100 iterations, this is run 3 times per filter technique to collect a bigger amount of data to evaluate. The functions use a JavaScript built in function “performance.now()”[7] to measure the time. It returns the time in milliseconds which we convert into seconds using the below formula.

(T0/ms - T1/ms / 1000 = Tres/s)

3.5 The data and what to filter

The data that we have used to measure the selected techniques against is a log file found on loghub’s github repo[8]. The one we selected is called Zookeeper[9], there was no specific intent behind it other than it has a small row count. We then scrolled through the log file and evaluated a search word that could potentially lead to an interesting way to test the different techniques on. We landed on the word “Received” as it occurs at a low frequency but spread across the whole log file.

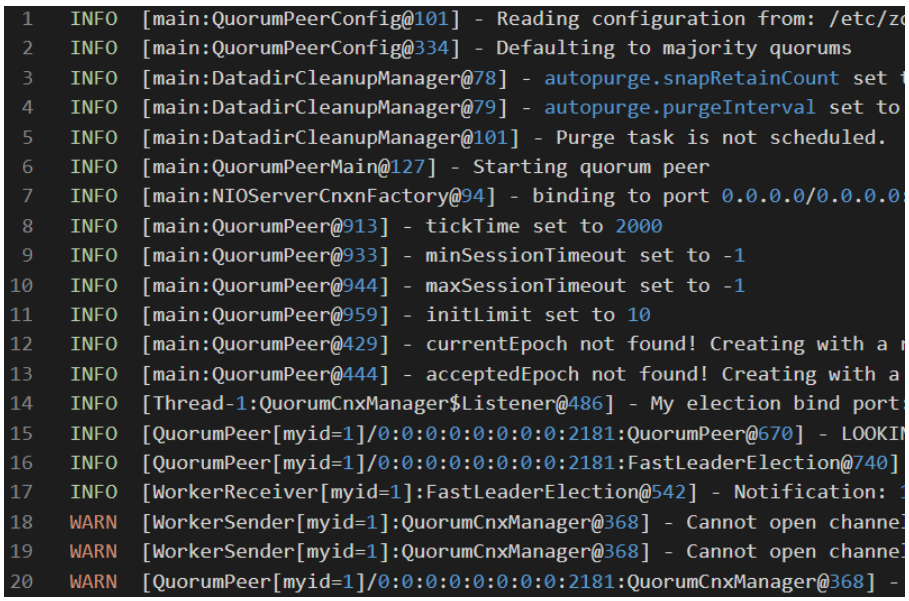

Figure-2: Before filtering example. The picture shows unfiltered 20 first rows of Zookeeper log file.

To make the file more adaptable and easier to evaluate on row count we adjusted the amount down to fifty thousand rows of the original file, cutting off anything after the fifty thousand mark. This made the file have 6977 occasions where the word “Received” matched which is roughly ~14% of the log file.

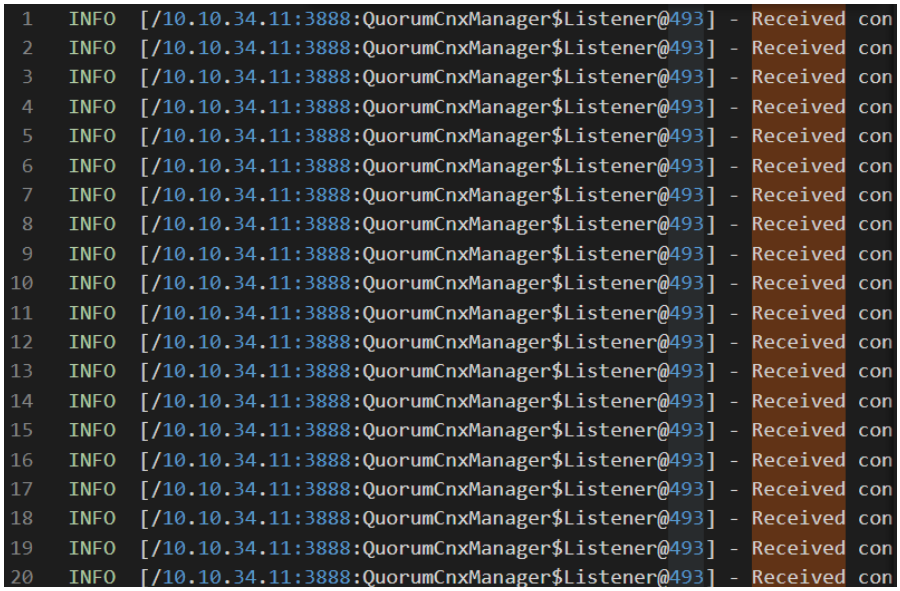

Figure-3: After filtering example. The picture shows the filtered 20 first rows of the word “Received” highlighted in orange in the Zookeeper log file.

That fifty thousand log file is now the default log to be used to create the other sizes. We then created a python script that we could create the sizes more efficiently and adjust the size of how big the file should be. The import method of the data is a bit different in MySQL. A second set of the data needs to be created. Csv file that has a table name and the rest wrapped with double quotations so it could be identified by the MySQL import system. Then thirdly a multi column csv file for the literature research.

Figure-2 and **Figure-3** shows a representative on what can be sorted and how we can expect what the user wants to see when we use the filters. The highlight in orange on **Figure-3** is just a visual but it shows where the matching search word is found within the text.

3.6 RQ1

To find an appropriate technique that is suitable for a developer there have been a couple of ground points. With this report there has been a generalization made and a conclusion on what is being used as of the research being done in the literature research section. With that there was a conclusion on an environment to use JavaScript/NodeJS. The environment where then tested with the different techniques on different sizes ranging between a smaller and larger sizes (see 3.5). When measuring the results there will be a diagram showing how the different technique performs with each size. Then there were conclusions around these diagrams, to see if they had potential to work with larger sizes or if they performed better with smaller ones.

3.7 RQ2 och RQ3

To measure the difference between the technique itself and the client, we set up timers around the specified area.
For RQ2 we measure the technique itself and leave other variables outside of its measuring grounds. Such as what will happen with the data that is filtered is not brought up with RQ2 making the technique isolated from other dependencies. A suitable approach would wrap the technique execution block, an example would be:

1. Call function and gather data if possible
2. Start time before filter applied
3. Apply filter
4. End time after filter is finished

Here we make sure we need asynchronous action to the filtering process. Making sure the filtering process is complete. One way is to check the length of the array and match it to how much the search word would reproduce to match.

For RQ3 there will be additional steps where the measurement starts timing before executing the function and ends when the simulation of sending data back to the request has finished.

1. Start time before function called
2. Call function and gather data
3. Apply filter
4. Return filtered data to function call
5. End time

This will check if there are more variables than just the technique at play such as data reading, like fs[10] (file system) for JavaScript filters or other variables that could affect the whole process.

3.8 Litteratur method

Literature study decided on what kind of approach of the filtering techniques that were chosen to go for. A search strategy was used that Prof. Pete Carr describes in his video “How to Read a Paper Efficiently”[11]. He explained that it is important that you do not read a report from beginning to the end, rather select the most relevant first and then judge if it is an important part for the study. Then continue to read the report more thoroughly.

In the literature method there were also testing and evaluation of different kinds of techniques within two of the three different selected techniques that were brought up. They are evaluated separately from the results but will be tested with the same data sizes and environments. This is to get an understanding of the different sub techniques or any thoughts about a technique that can be evaluated against other techniques of the same kind. Then take the best performing technique of that selection and use it in the result part of this study.

4 Literature review

4.1 Literature search

The literature search that have been brought up have been taken from Google Scholar[12] and BTH's library[13] with search words like: "testing api response", "database filtering techniques", "timed testing", "website load customers".

With these searches we have found many relatable literature and reports to select from. We had to then look through the reports and judge from the contents if they had value that could be connected to relevancy within my study. As the method stated we used the method from Prof. Pete Carr's video to evaluate the reports.

Weinberg, B. D. (2000)'s[14] did a study about how users experienced loading time could affect how the users decide if they want to stay on the web page or leave the page completely. The test contained a website that would take 7.5s to load. Here he had two categorized groups, one that had a sign saying "Please wait 5 seconds" and the other group where the sign said "Please wait 10 seconds". The group with 5 seconds experienced a higher quality of the website compared to the group that got the 10 second sign, even if both had the *same* load time (7.5 seconds).

Here we can conclude that with the faster a user gets on the webpage, they will experience the page as higher quality. Even if we do not write a sign that they need to wait 10- or 5 seconds, the principle of letting the customer wait longer will lower the quality experience.

2011 Jae Kyu Lee and Jong Yeol Lee[15] did a study to improve Android technologies. They conducted tests between two different programming languages, Java and Native C. They concluded a summary that "*Native C shows the best performance in the memory access operation*" but they also came to the conclusion that Java was more effective when designing an Android app that includes a lot of String processors. Because the processor does not need a type conversion of the strings. Their tests contained five different algorithms that contained different calculations of data. This is something that could be reflected with the different filtering technologies and how they handle the different data types. In such ways that changing the way the data is structured or what datatype the original data is might change the behavior of the technique.

In the study they used Java and Native C as their programming languages. They are object oriented languages and are adapted to run as fast as possible with as little dependencies of other modules. This was the year 2011 and new programming languages have developed science then. The most popular programming language as of 2021/08 is JavaScript[16]. According to the survey, with multioption, there were 83,505 people that answered with ~65% JavaScript as their most used and Node.js on ~33%.

The decision to go with the JavaScript/Nodejs environment was made very clear, to reach the most used area. As well as to get in some techniques that could be further replaced with different techniques other than what we are choosing by future works. Another outstanding programming environment in 4th place from the survey is SQL. Mostly aimed towards databases, which has a way of extracting data with queries that can be filtered if structured correctly. Searching on some surveys and articles and comparing to personal use we have narrowed the selection to MySQL as the SQL for this report. (First[35]- second[36]- and second[37] place).

With a few fitting choices to go for and also personal use that were used in earlier projects. It landed on three techniques where it could be implemented within a JavaScript environment: JavaScript built-in filter[17], AWK[18] and MySQL Queries[19] for databases.

The study of fingerprints by J. Feng and A. K. Jain[20] conducted a research of their, Because it took a long time to filter out one at a time, the need for a different technique was required. "*Therefore, a multi-stage filtering system is proposed, which utilizes pattern type, singular points and orientation field.*". Here they created a lesser- but with more specific filtering on the fingerprints to get an increase in results. There was an increase of performance by three fold in speed and precision increased by 3%.

With these studies there will be precautions that will be taken where there could be variants of ways to perform the different tests. Choosing different data types can affect the filtering techniques and therefore test different data types to choose a more optimized variant for the selected data selection.

While the JavaScript filter function does not need any change to implement to the given environment; MySQL and AWK do need to be looked at more thoroughly as they are not a part of NodeJS like JavaScript filters are. They need to be looked at how and what is the more effective technique in the given environment.

4.2 Research

Finding the most relevant information can be found with a simple google search and getting the simple explanation found through forums such as StackOverflow. But most, if not all, of the research is compiled from the manual, such as MySQL's[21], if it exists to help structure how to build the technique.

4.2.1 MySQL

MySQL queries can be powerful if used correctly. We can speed up the queries performed for MySQL filtering by using Indexing[22]. Indexing in MySQL works like a quicker search functionality. Without indexing, MySQL searches from top to bottom to check for matching rows. With Indexing they use an algorithm called B-Tree[23]. It starts at the middle and determines if the byte is bigger or smaller that needs to be searched, a lot like a Binary

Search[24]. Indexing works with the LIKE attribute of WHERE query, which we can use to filter out rows, though it does not work when searching with double % (wildcard[25]) of a word. Eg, 'search%' works while '%search%' does not work.[26] If we only care about checking for a column that we have created an index for and we either have a specified value or if it starts with the value we want to search for within the column, we should use indexes, otherwise we should search the whole row. We cannot create an index and do searches on every column with the OR option, we have to specify that we want to search every index given with AND signs[27].

While indexing can make the performance become highly necessary it has side effects. Such as updating, adding or deleting from a table that uses indexing will need more processing power to perform such tasks[28].

If we think around RQ1 then it is good to see the difference between indexing and just inserting the whole log for just a column. We can also take precaution to data types and structure different columns to handle the indexing. As a developer we get to evaluate how the file is structured and then proceed to create the database table with right columns. However in our limitations we want to search for the full text so we will need to use the double wildcards to match all text within the column.

So these two tests are created to find out which one is the most effective.

- One column
- Multiple columns

These are the two main ways I am going to test what is most efficient and then use this method for it in the result section. For this to work I had to modify the values of the log file to match the values slightly, which can cause a slight advantage of the multiple columns technique as it is slightly smaller in size. As we want to filter the whole text we will check every column for the search query as we want to be able to search everything to see if it matches our search word.

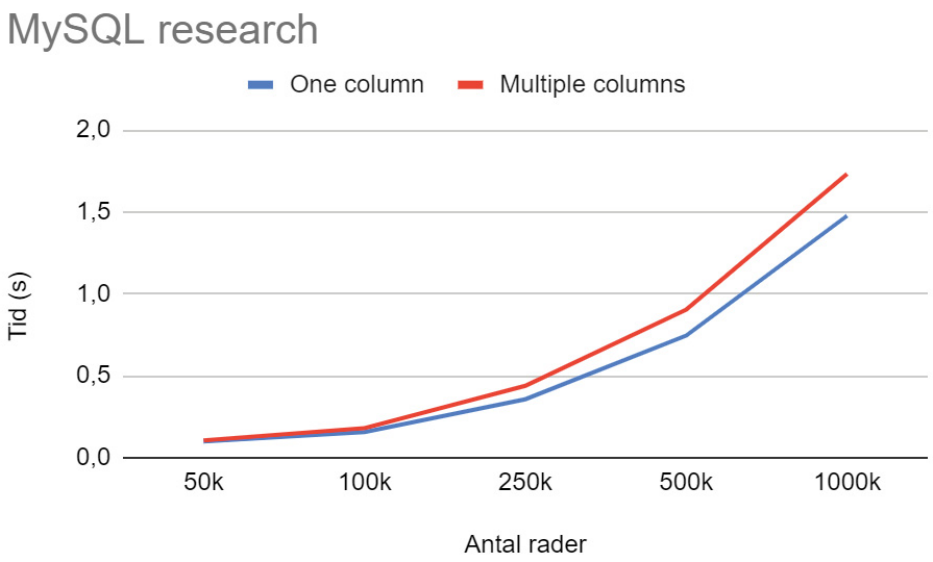

Figure-4: Effective comparison with different data sizes. The X-axis shows the amount of lines of a file. The Y-axis shows the average time to filter. Blue represents One column and Red represents Multiple columns

The results from **Figure-4** shows that the singular column takes the lead when it comes to bigger projects. Most likely because the multiple columns need to search every row five times compared to the one time for the singular. But for lower projects it would work to manipulate the data and create it into rows for better handling and possibly restricting the column searches. But as for this example we are still needing to search the whole string, which leads the One column test to be more efficient.

4.2.2 AWK

AWK is a standard filtering language to most Unix-like operating systems. It is used with a regex[29] pattern to check if the context is matching. As it is mostly used as a command line tool it has to be either imported to the develop environment, unless the developer is able to handle files. There are multiple options to this setup and to come up with the best one another test case needs to be performed.

There are two npm packets that should be used as AWK for JavaScript. The first one that were looked at only printed out everything into the console and did not save the results into a variable that we could potentially evaluate or send back to the client. The other npm packet for AWK that works like intended[30]. We will use the working npm packet and a JavaScript built-in function to execute the command in a JavaScript file called “*exec*”[31]. This we need to evaluate to see which one is better in performance and see how much difference they make for the linux based AWK tool.

- Linux AWK
- JS AWK
- Npm AWK

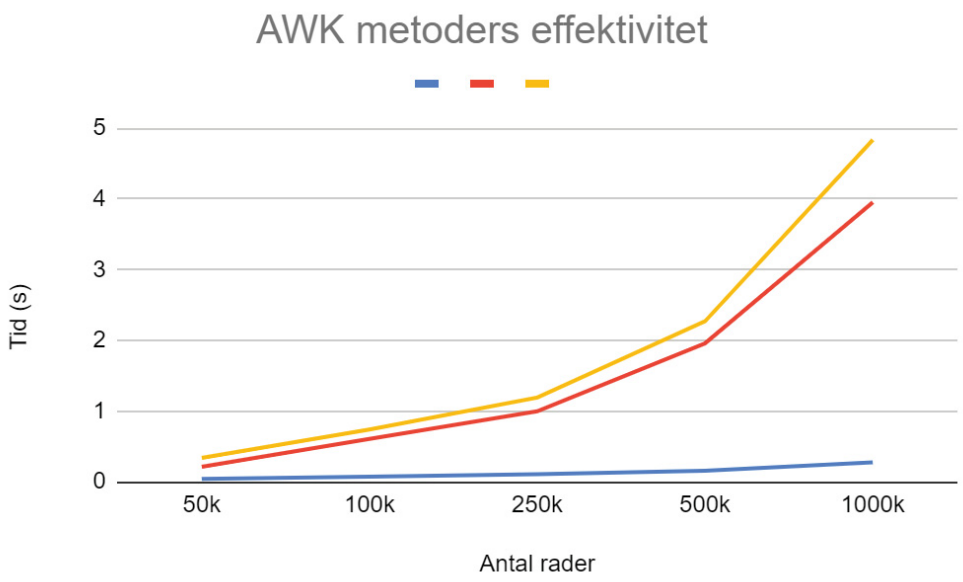

Figure-5: Effective comparison with different data sizes. The X-axis shows the amount of lines of a file. The Y-axis shows the average time to filter. Blue represents Linux, Red represents JavaScript and Yellow represents NPM Package.

The results from **Figure-5** show that the built in console performs really well at a bigger amount of data compared to the other two. It is also better at smaller sizes 50k to 100k, but not by a lot. NPM and JS keep a steady pace between each other, but JS is slightly better at all sizes.

Because we are figuring out what tool is most effective in a case by case, we aren't focusing on RQ3. But that said we still have to make sure we are able to use it. As the built in AWK tool for Linux is used primarily with a console, so as the results might look brightest for Linux based AWK. It might not work as great dynamically thinking from a client view, because it does not have a console to use from.

5 Results

Filtering techniques were executed in docker containers and the measurements were saved onto an excel sheet of their performance. Files that were used to measure the results is a log file called Zookeeper from Loghubs' github repo. The files have been modified to fulfill the requirements to be imported into the MySQL database. The first five thousand rows is the base for the tests log file and at larger sizes it is increased by another five thousand. The execution times were measured with the built in time tools for each technique as well as scoped for just the tools and the whole execution of the docker container.

The tables shown in each performance category will have the first column as a description of what the rows contain. The row with **Rows** has the amount of rows a file has in that filtering measurement. **Filter** row contains the time it took to filter out the associated amount of rows with that specific technique. **Execution** row contains the time it took to filter out the associated amount of rows with that specific technique.
The time is measured out to seconds and rounded up to its closest 5th decimal point to fit into the table cell.

5.1 MySQL

The database was set up with a One column with full text from the research in the page (insert page number). The query searches through the whole text string and finds the matching text back.

5.1.1 Technic performance

Table-1: MySQL filter measurements average time

JS filter	Technique time				
Rows	50k	100k	250k	500k	1000k
Filter	0,09882s	0,15591s	0,35652s	0,74516s	1,47654s
Execution	0,10161s	0,16126s	0,3579s	0,74575s	1,4634s

Note: Showing in second and rounded to its closest 5 decimal point.

Table-1 shows that the MySQL filtering technique has a steady increase by time as the amount of data increases. The execution time does not differentiate a lot from the filtering technique.

5.2 JavaScript filter

We set up the data by reading the log file with JavaScripts built in fs function.

One flaw of the docker component for JavaScripts filter is the memory size of the docker container. When implementing a size larger than two million rows crashed the docker container as it could not handle the amount of memory for that size. So the maximum number of rows for the data size came to one million for the docker containers base memory capacity.

5.2.1 Technic performance

Table-2: JavaScript filter measurements average time

MySQL filter	Technique time				
Rows	50k	100k	250k	500k	1000k
Filter	0,00415s	0,02015s	0,03471s	0,04084s	0,10505s
Execution	0,02083s	0,06708s	0,09523s	0,20293s	0,8057s

Note: Showing in second and rounded to its closest 5 decimal point.

Table-2 shows that the JavaScript filter technique performs really fast for small and big techniques. The execution time has a higher time than the technique and dramatically increases when performed with the largest set of data. This has to do with how the loading of the data is implemented when executing the function.

5.3 AWK

AWK uses the JavaScript function “exec” with a command line that executes AWK command as of the literature research results.

5.3.1 Technic performance

Table-3: AWK filter measurements average time

AWK filter	Technique time				
Rows	50k	100k	250k	500k	1000k
Filter	0,21038s	0,59282s	0,97729s	1,92854s	3,89074s
Execution	0,21332s	0,60325s	0,98891s	1,94280s	3,88871s

Note: Showing in second and rounded to its closest 5 decimal point.

Table-3 shows that the AWK filtering technique has a steady increase by time as the amount of data increases. The execution time does not differentiate a lot from the filtering technique.

5.4 Comparison

Table-4: Percentage between filter and execution time of each technique

	% increase execute time				
Rows	50k	100k	250k	500k	1000k
MySQL	102,83%	103,43%	100,39%	100,08%	99,11%
JavaScript	502,17%	332,93%	274,36%	497,05%	766,96%
AWK	101,39%	101,76%	101,19%	100,74%	99,95%

Note: Taken average filter time and respective execution time.

Table-4 shows that we can see an even percentage of filter- and execution time between MySQL and AWK. MySQL has +- of ~4% rounded up difference and AWK has +- of ~2% rounded up difference over all sizes of data. JavaScript has a bigger difference ranging between the sizes of data, going from ~503% at small data size and ~767% at bigger size. Meanwhile the middle size has the lowest increase between the other data sizes, at ~275% rounded up.

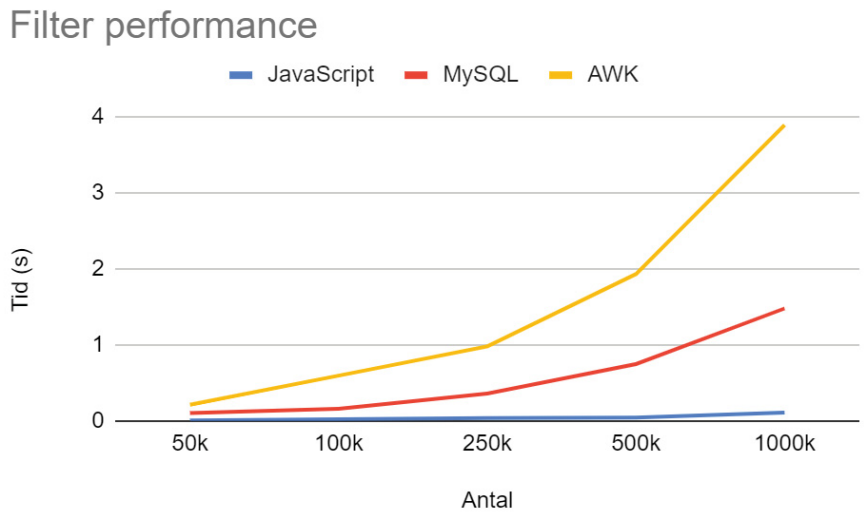

Figure-6: Filter performance with different data sizes. The X-axis shows the amount of lines of a file. The Y-axis shows the average time to filter.

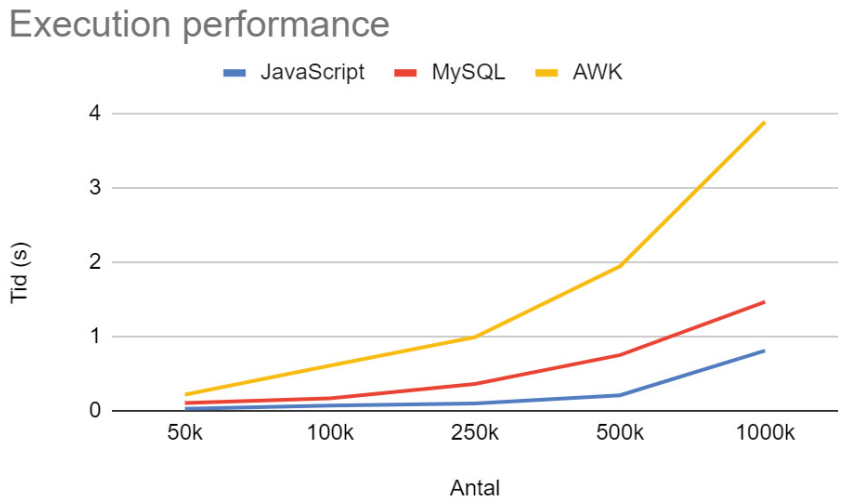

Figure-7: Execution performance with different data sizes. The X-axis shows the amount of lines of a file. The Y-axis shows the average time to filter.

As of **Figure-7** we can see that JavaScript has a higher exponential growth of time compared to **Figure-6**. Especially when compared between 500k and 1000k X-axis values.

6 Analysis and discussion

6.1 RQ1

Answering the question of RQ1 we have to work with the fact that the developer is having a setup of JavaScript where they can implement these filtering techniques. With the results we got from the literature search we would like to work with the linux technique from the AWK method. As for the implementation of the way to execute the AWK, it was through JavaScript's own command line execution function. It uses a utility method that promises the execution to become asynchronous, meaning waiting on the result before continuing. However it had no way near as fast as the linux built in AWK function.
Likewise to linux we researched MySQL methods to query a table if we should use one or multiple columns. Where the discussion of indexes came clear. Indexing is a common thing to use in MySQL to speed up the process of finding specific rows, especially when filtering out specific rows. On one hand it speeds up the process of almost all MySQL queries, especially the larger the file. But on the other hand it shows it cannot be performed under our preferences as we set up the environment as it uses the double wildcards, which renders the index method useless.

At the smallest size the amount of time it took to filter through the different techniques were below 0.25s on each one of them. On one hand it does not matter that much when the difference between the longest- and shortest time is *only* ~0,19s for the execution time, on the other hand it still saves time and will cause few resources to be used, as well as making the webpage a little better in quality.

The three techniques are very efficient to smaller sizes to medium with all of them being under one second of filtering. However JavaScript performed with a higher performance than the other, keeping itself under one second throughout the measuring phase.

6.2 Isolation

The isolation section has a big importance to testing the technique without having dependencies that fluctuate the results. With this there was not an environment with an API set-up and tested upon, but rather a small function that can simulate an API request.
It could also be argued that it is not the full experience of the API. However separating a function from the softwares infrastructure is a necessary thing to do as it makes sure you can implement the function and not depend on other methods to work before implementing them, especially at larger applications.

Docker containers worked with me alot here. They could set up the different techniques with their specific requirements and rely more on the fact that the docker container is isolated from the rest of the VM. This meant that the measurements were more reliable and would be able to get the same results over different periods of time.

6.3 RQ2 and RQ3

From **Table-1** and **Table-3** we can see that they are not differentiating a lot from the filter technique and when the data is sent back. Nearly no difference like we can see at **Table-4**. This is mostly because when we are executing MySQL queries- and AWK reads the file at the same time as it executes the filter function. So we can not separate them by filtering and reading data.

As of **Table-2** and **Table-4** JavaScript has a bigger change when needing to read the specified file and the executed filter time. As of RQ3 we can assure this is a client notice when reaching higher levels of filtering. We can see with the **Figure-5** that the JavaScript filtering execution is making an exponential curve, which will increase the time the bigger the project becomes.

Acknowledging the existence of latency within the function call of JavaScript filter technique could be interpreted as a faulty implementation of the technique. "*Just move the loaded state outside of function and call it a day*" is what someone could argue. However as this test of method we want to be able to access the logs with flexibility not hard code a path and select it. As AWK and MySQL both handle it with low differentiation between end-to-end and the techniques speeds it shows that it is a stable way to access the different tables or files without getting an overwhelming difference between the performances.

How exponential the JavaScript curve is could also be argued that the time will be too long for files with more than one million rows as it could surpass the other filtering techniques. For future work that is something to look into, a way to handle even larger files and compare it to MySQL and AWK. As the JavaScript filter will more than likely pass both MySQL and AWK in time needed to finish its execution time. However, the time the JavaScript filter takes to just filter anything, one could set up an API (or in our case, a docker container) with pre-loaded data, and therefore the JavaScript filter would compete for a larger size pool.

6.4 Indexing

One thing that should be looked at more is indexing within MySQL technique. Indexing would probably make a greater competitor to JavaScripts filter technique. As it showed with the limitations we have, we could not search the full text within each column. But if a developer instead had other requirements and was more limited to just searching for a specified string. Then the idea of indexing would have likely boosted the performance of filtering the data.

6.5 AWK

The AWK method felt like a very special case in this study. Yes it is a fast way to get a specified filtered file and then structure it to a different styled file. Which was introduced within the vLinux course within BTH[38], where a given assignment was to create a Json file out of a given log file. With the vLinux course it did not take long to structure the log file that where assigned to a json file that was used later to search through on a command line script and web application. But to use it as an extension of JavaScripts' built-in execution function made it feel less powerful as it had to take many steps to work asynchronously with the program.

6.6 Literature search

When conducting the research of fast loading time and how it affected the user's experience of a website, Weinberg, B. D. (2000)’s study gave a lot of insights. This was the general focus for RQ3, making sure that the end-to-end experience is also met with high quality and not just the technique itself.

With this in mind the focus of function testing and making sure the client side does not get forgotten with the testing methods. This also evolved into the isolations of the testing phase. Making sure that the techniques get the same amount of CPU with the help of docker containers and also not rely on the whole system to work. Like sending over the data over an internet connection, which would make the measurements fluctuate.

Getting a grasp on what programming languages are popular at this time is necessary. We believe to get a good majority of people under the same development area. Looking at how important it was for the Android technologies research that Jae K. L. and Jong Y. L. did make a broader scope on what technologies people could be using. Though AWK is not on the survey, it kind of includes the bashs/shell[32] techniques. But that being said it was selected mostly because of its own uses and wanted to see if it had any run for it compared to the other techniques.

Fingerprints study by J. Feng and A. K. Jain on how to increase performance might mean different but also altering methods of the selected technique. This led to wanting to test MySQL in a different direction than just implementing it to the database and then selecting a query. The test differentiate between multi columns or a singular. But with the given limitation we need to check every column of the multi column method. Because of that we searched the whole table five times compared to the one time the singular column test did. The test would be in favor of the multi column method if we knew what row we would need to search, which can be a part of future study.

Conclusions

This study had two main questions opposed. What filtering technique is most suitable for small and bigger projects. As Well as which technique is more effective within each technique and its execution time.

Given the limitation of an environment of NodeJs with its strict uses. We were still able to implement three different techniques and measure their performance. The study showed that JavaScript's filtering had a much clearer effectiveness when it comes to speed. Both within the filtering technique itself and execution time of the function. The literature research gave an overview on what could be expected from the techniques. As well as an understanding on what could be changed to improve the techniques if possible.

At small sizes all of them have really high performance. Though JavaScript's filter is still performing fastest, MySQL and AWK have potential of being used at the same sizes. The small to medium size is also really fast in performance. Here AWK is just barely at the one second mark which I would deem noticeable for the client side. Which from the study that Weinberg, B. D. (2000) did, lowers the quality of the overall site. As the study had worse quality when it displayed that the site would take longer to load, compared to displaying shorter time. Same goes with having to wait on the loading of a webpage, as it will decrease the quality of the webpage if it takes a second to load compared to other faster alternatives.

With bigger sizes of data the JavaScript filter implementation gave an unexpected outcome that led to a break point of sized data. But the technique alone still performed outstandingly at large amounts of data compared to the other two techniques. Looking at the outcome of the function performance at **Figure-7** the exponential curve in the positive direction of JavaScript filter function seems to increase drastically. For further research an even bigger data of size should be implemented to see how JavaScript's filter handles the data given. As it might cross over MySQL and AWK.

Validity threats

One of the biggest validity threats could be, at larger projects, I do not have larger samples. The docker container I built had a max capacity of memory and I could not find a way to increase it. The memory option for building the docker compose did not work so it stopped the process of expanding and therefore maybe JavaScript filters have a bigger impact on the performance.

My knowledge of the programming languages can be invalid as I do not know everything about MySQL or JavaScript, as there could be better ways to optimize the program. When outputting the test, a list of all the time should have been saved before printing out, might affect the cpu by printing out therefore not optimizing to a 100%.

The way I have implemented the different techniques might be another reason for the validity threat. I try to stay as close to the source and follow manuals and the given guides on how to implement the techniques. But that does not mean it is a bullet proof way or some ways have better optimisation.

The data collected is not tested with any particular statistical formula to see if any are an anomaly to one other. A Student T-Test could be set up with a zerohypothesy that checks if there is or is not an increase in difference between the different 100 loop sets. This would make it more reliable that the tests are valid to be evaluated.

Future work

This paper has alot of limitations, it started out with a bigger scope and alot more freedom. As I got more into setting up and creating a structure for the methods and how the data should be handled a lot more limitations were built. One of these limitations was the use of an API to send requests to then get the filtered data back. When we think about it, it makes sense to just limit the measurements to the function call and how long time it takes to get the data back from the function. Then we make it more solid with how long it takes and do not have unstable variables of how long each file took to reach the client.

A future work that has a broad exploration potential is MySQL indexing and JavaScript's memory. As the discussion of limitations and how indexing works on a narrower scope on this issue could be further tested upon. Figuring out ways MySQL's indexing can work together on an even playing field and testing different queries of indexing that could lead to a method being used that is acceptable with limitations but work fast. The scope of JavaScript's memory is also a challenge, if the study can be reproduced but with a better solution to the memory issue, the results will have an even bigger impact and larger sizes of data can be tested opon.

The docker CPU part helps out alot when collecting the data, because if we would have given it full power the measurements would become really fast and not as limiting, but also quite small to look at. We are milliseconds, but what would happen if I used my real computer at full capacity? It would be alot smaller, then we'd have to count on the timer function to be working just perfectly.

This leads to an interesting thought of, how would it work with a live api? Will it be able to be measured with marginal errors that affect the sending of data. Developing some API endpoint that sends back the filtered data and measures the time it took from button click to receiving the filtered out data.

References

[1] VirtualBox. https://www.virtualbox.org/ (Visited 2022-05-06)
[2] Debian. https://www.debian.org/distrib/ (Visited 2022-05-06)
[3] Docker. https://www.docker.com/resources/what-container/ (Visited 2022-05-06)
[4] Docker “Get started”. https://docs.docker.com/get-started/overview/ (Visited 2022-05-06)
[5] Docker “CPU”. https://docs.docker.com/config/containers/resource_constraints/#cpu (Visited 2022-05-06)
[6] Mozilla “Asynchronous”. https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Asynchronous/Introducing (Visited 2022-05-06)
[7] Mozilla “Performance now” https://developer.mozilla.org/en-US/docs/Web/API/Performance/now (Visited 2022-05-11)
[8] Shilin He, Jieming Zhu, Pinjia He, Michael R. Lyu. Loghub: A Large Collection of System Log Datasets towards Automated Log Analytics. Arxiv, 2020.
[9] Github “Zookeeper”. https://github.com/logpai/loghub/tree/master/Zookeeper (Visited 2022-05-11)
[10] NodeJs “File system”. https://nodejs.org/api/fs.html#file-system (Visited 2022-05-09)
[11] How to Read a Paper Efficiently (By Prof. Pete Carr). https://youtu.be/leaD0ZaUJ3Y (Visited 2022-04-23)
[12] Google scholar. https://scholar.google.com/ (Visited 2022-04-23)
[13] Bth’s bibliotek. https://www.bth.se/bibliotek/ (Visited 2022-04-23)
[14] Weinberg, B. D. (2000). Don’t keep your Internet customers waiting too long at the (virtual) front door. *Journal of Interactive Marketing*, *14*(1), 30–39. https://doi.org/10.1002/(SICI)1520-6653(200024)14:1<30::AID-DIR3>3.0.CO;2-M
[15] J. K. Lee and J. Y. Lee, "Android programming techniques for improving performance," 2011 3rd International Conference on Awareness Science and Technology (iCAST), 2011, pp. 386-389, doi: 10.1109/ICAwST.2011.6163105.
[16] Statista “Most used programming languages”. https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/ (Visited 2022-04-24)
[17] JavaScript Filter. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/filter (Visited 2022-04-24)
[18] Wikipedia “AWK”. https://en.wikipedia.org/wiki/AWK (Visited 2022-04-23)
[19] MySQL ”Examples of Common Queries”. https://dev.mysql.com/doc/mysql-tutorial-excerpt/8.0/en/examples.html (Visited 2022-04-24)
[20] J. Feng and A. K. Jain, "Filtering large fingerprint database for latent matching," 2008 19th International Conference on Pattern Recognition, 2008, pp. 1-4, doi: 10.1109/ICPR.2008.4761851.
[21] MySQL “Documentation”. https://dev.mysql.com/doc/ (Visited 2022-05-07)
[22] MySQL “Indexes”. https://dev.mysql.com/doc/refman/8.0/en/mysql-indexes.html (Visited 2022-05-06)
[23] MySQL “Comparison of B-Tree and Hash Indexes”. https://dev.mysql.com/doc/refman/8.0/en/index-btree-hash.html (Visited 2022-05-07)

[24] Wikipedia “Binary search algorithm”.
https://en.wikipedia.org/wiki/Binary_search_algorithm (Visited 2022-05-07)
[25] MySQL “Pattern Matching”.
https://dev.mysql.com/doc/refman/8.0/en/pattern-matching.html (Visited 2022-05-07)
[26] MySQL “Table scan avoidance”.
https://dev.mysql.com/doc/refman/8.0/en/table-scan-avoidance.html (Visited 2022-05-07)
[27] MySQL “Multiple column indexes”.
https://dev.mysql.com/doc/refman/8.0/en/multiple-column-indexes.html (Visited 2022-05-07)
[28] MySQL “Insert optimization”.
https://dev.mysql.com/doc/refman/5.7/en/insert-optimization.html (Visited 2022-05-07)
[29] Wikipedia “Regex expression”. https://en.wikipedia.org/wiki/Regular_expression (Visited 2022-05-11)
[30] NPM “Awkjs”. https://www.npmjs.com/package/awkjs (Visited 2022-05-02)
[31] Mozilla “Exec”.
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/RegExp/exec (Visited 2022-05-09)
[32] Wikipedia “Bash (Unix shell)”. https://en.wikipedia.org/wiki/Bash_(Unix_shell) (Visited 2022-05-11)
[33] Wikipedia “API”. https://en.wikipedia.org/wiki/API (Visited 2022-04-24)
[34] NodeJs https://nodejs.org/en/about/ (Visited 2022-05-11)
[35] TDS “Top 10 Databases to Use in 2022”. (Visited 2022-06-06)
https://towardsdatascience.com/top-10-databases-to-use-in-2021-d7e6a85402ba
[36] Appinventiv “Top 15 Databases to Use in 2022”. (Visited 2022-06-06)
https://appinventiv.com/blog/top-web-app-database-list/
[37] DB-Engines “Ranking”. https://db-engines.com/en/ranking (Visited 2022-06-06)
[38] BTH “vLinux”. https://www.bth.se/utbildning/program-och-kurser/kurser/20232/BQF3L/ (Visited 2022-06-12)

Appendix

A:

Google excel sheet with all the measurements included in different tabs
https://docs.google.com/spreadsheets/d/18cieCJj0WWWXPSyF_gCAPDyC4Y8dWdcfgT_uTWC4YPo/edit?usp=sharing

B:

Github which contains the code done to complete this study
https://github.com/AreonL/Exam