OULUN YLIOPISTO
UNIVERSITY of OULU

Designing, implementing and evaluating a database for a software testing team

University of Oulu
Department of Information Processing Science
Master's Thesis
Ilari Matinolli
23.11.2016

Abstract

Databases have existed since 1960s. Relational databases came out in 1970s and they have been the most popular database technology ever since. However, non-relational databases came out in 2000s and have gathered popularity since then. This thesis studied different open-source database management systems including relational and non-relational systems. Prompt looks into data analytics and distributed databases were also taken.

The main purpose of this study was to design, implement and evaluate a new database for the needs of a software testing team. In order to do that it was needed to find out a database management system that met the requirements best given by the case company. It was also needed to find out which database design principles and techniques should be followed to achieve a well-performing and easily maintainable database.

After studying different database management system, MySQL was chosen for the database management system and a new database was designed, implemented and evaluated. It turned out that MySQL is still a relevant and successful database management system to meet the needs of a modern software testing team when optimization techniques such as indexing and normalization are applied.

In the future it would be beneficial to study how a non-relational database management system could be combined with a relational database management system to meet the needs of different software testing teams.

Keywords
SQL, NoSQL, databases, relational databases, non-relational databases, software testing

Supervisor
Professor Mika Mäntylä

Foreword

This is it, probably the last piece of work I did to finish my university studies that began a little over 5 years ago in fall 2011. Back then the thought of finishing my Master's thesis and getting my Master's degree felt like a distant dream – now it is coming true.

I would like to thank the case company and the representatives of the case company for this opportunity to work as a thesis worker. I would also like to thank Professor Mika Mäntylä for supervising and guiding me during the process of making this thesis.

The process of making this thesis has taught me to do academic research and at the same time I got a good preview of what it is like to work in IT industry in a global environment which I am very thankful for.

Ilari Matinolli

Oulu, November 10, 2016

Contents

1. Introduction

The purpose of this Master's thesis is to design and implement a database for a software testing team. The motivation for this thesis comes from the need for a new database that the software testing team in the case company has. It is important to study different database management systems for two reasons. Firstly, there are nowadays hundreds of database management systems to choose from and the number is growing as the technology is evolving. In the ranking by DB-Engines (2016a) it could be seen that there existed 284 different database management systems at the time of this study. Secondly, different database management systems are designed to match different needs. For example, non-relational databases are used to store unstructured data (Leavitt, 2010). In the end, after the evaluation, it will become apparent if the previous research and the literature can be utilized in a real life case as the decision about the choice of technology will be based on previous research and literature.

The structure of the thesis is as follows. The rest of this section covers the research method and the research questions. The second section is an introduction to databases. The requirements for the database given by the case company are presented in the third section. The fourth section is about relational databases and the fifth section is about non-relational databases. A brief look into data analytics is taken in the sixth section and a look into distributed databases is taken in the seventh section. The final decision of the database management system used in this study is made in the eighth section. The design, implementation and evaluation process of the new database is presented in the ninth section. Discussion and implications are presented in the tenth section and conclusions are presented in the eleventh section. The list of references is found on the last pages.

1.1 Research method

In this sub-section the research method, design science research, conducted in this study is explained.

Along with behavioral science, design science is foundational to the information systems discipline. The fundaments of design-science paradigm is in problem-solving. The design-science paradigm seeks to create new and innovative artifacts to extend the boundaries of human and organizational capabilities. Design science connects people, organizations, and technology. In design science an artifact is designed, built and applied to understand the problem domain and its solution and to gain knowledge about them. (Hevner, March, Salvatore, Park & Ram, 2004.)

Organizations want new information systems to be implemented to improve effectiveness and efficiency within the organization. The final artifact is meant to extend the boundaries of human problem solving and organizational capabilities. This is done by providing tools that are computational and intellectual. (Hevner et al., 2004.)

The design process consists of activities which lead to an innovative product as the outcome. In other words, design science is a problem-solving process as previously mentioned. (Hevner et al., 2004.) Design can be seen as a set of activities which is a process and as a product which is an artifact (Walls, Widmeyer & El Sawy, 1992). The artifact is evaluated to get feedback and a better understanding of the problem area. This way the product and the design process can then be improved. The researcher must be aware of improving the design process and the artifact during the possible several iterative loops that consist of building and evaluating. (Hevner et al., 2004.)

1.1.1 Design science research guidelines

The objective of this study is to design and implement a database for a software testing team. The previously mentioned design science approach by Hevner et al. (2004) is applied in this study. The database is the design artifact. There are seven guidelines for a successful design science research by Hevner et al. (2004). I explain how the guidelines are followed in this research. However, it should be noted that Hevner et al. (2004) mentioned that the guidelines are not mandatory to follow. Creative skills and judgement should be used to decide when, where and how to apply each of the guidelines in research projects. The guidelines are the following:

1. **Design as an artifact:** Design-science research must produce a viable artifact in the form of a construct, a model, a method, or an instantiation. The result of DSR is a purposeful IT artifact that is created to address an organizational problem. The IT artifact must be effectively described that it can be implemented and applied in an appropriate domain. (Hevner et al., 2004.) The artifact in this research is the designed, implemented and evaluated database. It addresses the case company's problem which is the lack of a proper database for a software testing team's internal use.

2. **Problem relevance:** The objective of design-science research is to develop technology-based solutions to important and relevant business problems. Design science approaches this problem by constructing innovative artifacts aimed at occurring changing phenomena. A problem is sometimes defined as the differences between the current state and the goal state. Designing organizational information systems plays a big role in enabling effective business processes to achieve goals such as increasing revenue and decreasing cost. (Hevner et al., 2004.) The software testing team in the case company needs a better database; the new database must have better read-performance and better maintainability than the old database. I construct the artifact in this research to improve daily work processes. The new database intends to enable more effective processes inside the organizational unit.

The team currently has an internal database in use. However, the database has been implemented in a hurry without paying much attention to the design. MySQL was chosen for the technology without doing research or comparison to other database technologies. MySQL was chosen because there is an open-source version available and it is relatively easy to find guides and instructions of MySQL. Those actions have led to a situation where the current database is disorganized. As there has not been attention paid to the design, for example no indexing has been made, the current structure of the database has led to problems. The current sizes of the tables are becoming too large and the query times have become long which, of course, affects the usability of the database system as well. A query time of over 20 seconds for an action that is needed to execute often is not optimal. New data is being stored around the clock, so the situation is just getting worse all the time. The database is difficult to expand and update; when something new is needed to be added into the database, a new table is just created. It has not been thought how, for example, the current tables could be updated in a way that the outcome would be the same but more efficient. The maintainability of the database is weak, in other words. No normalization, that is very common within relational databases, has been made to minimize data redundancy.

3. **Design evaluation:** Evaluation is an important part of the research process. The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation methods. The requirements, that the evaluation of the artifact is based on, come from the business environment. IT artifacts can be evaluated, for example, in

terms of the following factors; functionality, completeness, consistency, accuracy, performance, reliability, usability, and fit with the organization. Mathematical evaluations can be also done when analytical metrics are appropriate. (Hevner et al., 2004.)

The design artifact is complete when it satisfies the given requirements. As design is an iterative activity, the evaluation provides feedback for the construction phase which is essential in order to get a complete artifact. The design evaluation methods can be observational, analytical, experimental, testing, or descriptive. (Hevner et al., 2004.) In this research the requirements for the artifact come from the case company. For example the increased performance of the database system compared to the old database is evaluated experimentally by simulation. A more detailed explanation of this can be found in section 1.1.2 Design science research framework.

4. **Research contributions:** Effective design-science research must provide clear and verifiable contributions in the areas of the design artifact, design foundations, and/or design methodologies. The artifact itself is the most common way to contribute in design science research. The artifact offers solutions to the previously unsolved problems. The artifact may extend the knowledge base or it can apply the existing knowledge innovatively. The creative development of well-evaluated constructs, models, methods, or instantiations are important contributions as well. Those contributions should extend or improve the existing foundations. The creative development and evaluation methods also provide contributions to the design science research area. (Hevner et al., 2004.) The database is constructed to solve a problem within the case company which is the main contribution. After the evaluation I will be able to see whether the knowledge from the previous literature is beneficial and utilizable in a real life case. Another contribution is intended for people who are about to implement a new database. They can find up-to-date information about databases in this research. Nowadays there is a big boom about big data and non-relational databases. People interested in databases can find out whether there really is a need for a non-relational database in their case or if a relational database does the job as well.

5. **Research rigor:** Design-science research relies upon the application of rigorous methods in both the construction and evaluation of the design artifact. In design science rigor is derived from the effective use of theoretical foundations and research methodologies. The success is defined from how well the researcher selects techniques to develop the artifact and how well the researcher selects the means to evaluate the artifact. It is also important to remember to exercise the artifact in appropriate environment. (Hevner et al., 2004.) The methods for constructing the artifact come from the existing literature. Thus, it is important to choose literature that is authentic. The methods for evaluating the artifact depend on what the case company requires. The performance can be evaluated, for example, by comparing the query times of the old and the new database. The database is also tested in its real environment with real data.

6. **Design as a search process:** The search for an effective artifact requires utilizing available means to reach desired ends while satisfying laws in the problem environment. Design process usually consists of two parts that make an iterative cycle. The parts are generating design alternatives and testing alternatives against requirements and constraints. (Hevner et al., 2004.) The design phase is based on the existing research and literature about databases that can be found from the 1970s until today. The requirements come from the case company. Those two processes make an iterative cycle that is repeated until a satisfied outcome is reached. The design and test cycle is repeated until the requirements are met.

7. **Communication of research:** Design-science research must be presented effectively both to technology-oriented as well as management-oriented audiences (Hevner et al., 2004). This study communicates with the technology-oriented audience by explaining the design and implementation processes and, for example, why particular technologies were chosen to conduct the work. The study communicates with management-oriented audiences by explaining how the artifact improves the daily working within the case company's software testing team in this case.

1.1.2 Design science research framework

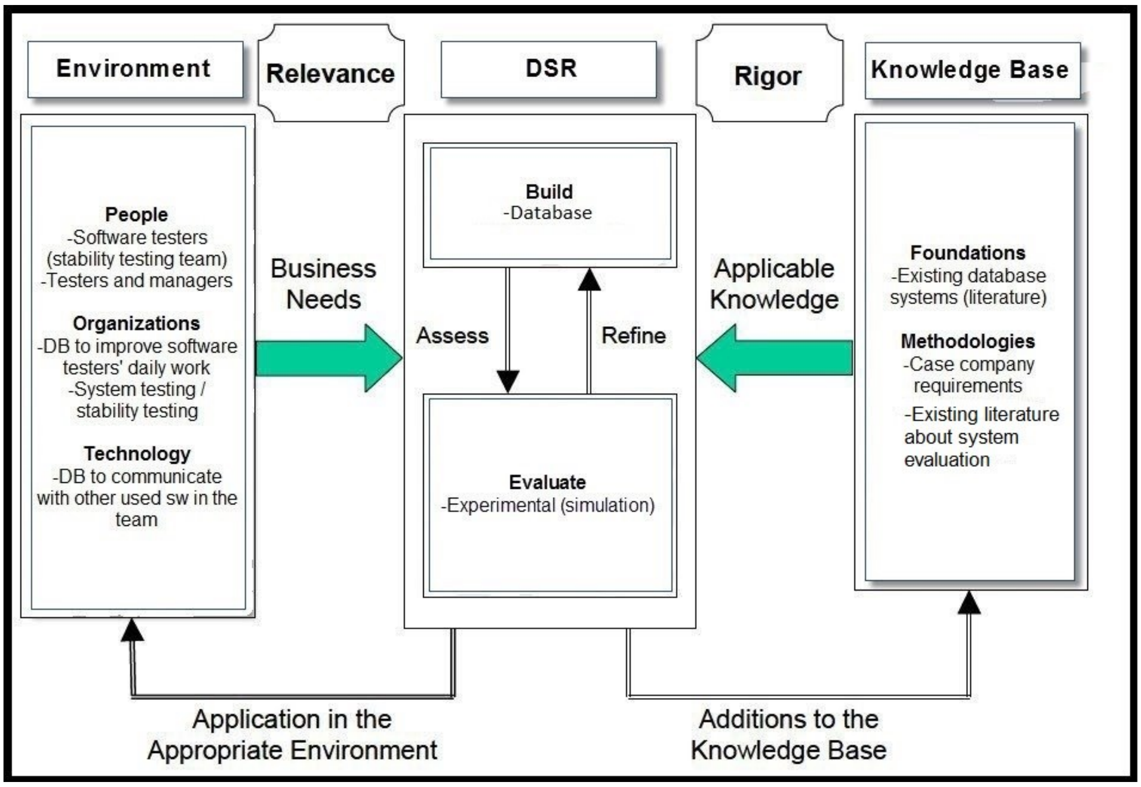

Environment
Relevance
DSR
Rigor
Knowledge Base
People
-Software testers (stability testing team)
-Testers and managers
Organizations
-DB to improve software testers' daily work
-System testing / stability testing
Technology
-DB to communicate with other used sw in the team
Business Needs
Build
-Database
Assess
Refine
Evaluate
-Experimental (simulation)
Applicable Knowledge
Foundations
-Existing database systems (literature)
Methodologies
-Case company requirements
-Existing literature about system evaluation
Application in the Appropriate Environment
Additions to the Knowledge Base

Figure 1. Design science research framework applied in this study inspired by Hevner et al. (2004).

Hevner et al. (2004) have created an information systems research framework. It was made for understanding, executing and evaluating information system research when behavioural science and design science are combined. Figure 1, inspired by Hevner et al. (2004), presents how design science approach is utilized in this study.

Environment: The actual users of the database are mainly the software testers of a stability testing team. The team runs automated stability software tests for specific software that are run on specific devices under test (DUT). Stability testing is part of system testing. People from management level might find the information offered by the database useful as well.

The main reason for implementing a new database is to improve the daily work of the software testers. A lot of data is generated around the clock as software tests are being run continuously. This data needs to be stored in a way that it is efficient, easy and fast to access, and extendable in the future when needed. The current solution can be seen as inefficient and disorganized as explained in the Problem relevance –section. The stability software testing team is just one small part of a much larger organization and the database is implemented for the team's internal use.

The database is used to store data from the test runs. DUTs' statuses, tested DUTs, tested software versions, found faults, key performance indicators (KPI), uptimes of the DUTs and CPU loads are stored, for example. This data is presented in a way that it is fast accessible and easy to read for the users. An example situation of use would be the following. A software tester finds out that a new fault is occurring with the recent software version tested. To decide how to proceed further with the fault, for example whether to report the fault or not, he wants to find out if the same fault has been visible with the previous software versions tested or with other DUTs that are under testing. The tester could use the database system to view the previously mentioned matters. The explained situation is possible to go through with the current database system, but as explained in the Problem relevance –section, it has its current weaknesses, such as slow queries, which are addressed to be solved with the new database.

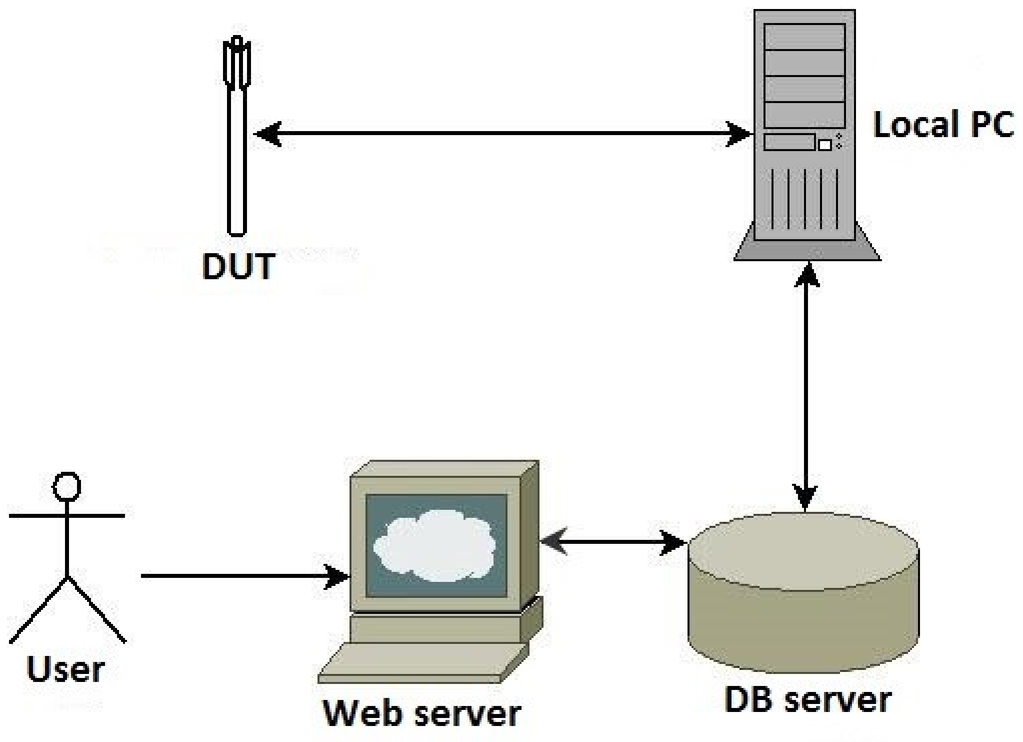


Figure 2. Technology of the environment.

The technology of the environment is presented in Figure 2. Each DUT is connected to a local PC. The local PCs are used to control DUTs via software and the local PCs are used by the software testers in the testing laboratory. An automation framework software is used to run test cases for the DUT software. The automation framework also automatically collects data about tested software versions: faults found during the test runs, data about memory usage and uptimes of the DUTs, for example. This data is sent to the database server that all local PCs are connected to via web server.

The users, who are mainly the software testers of the stability testing team, are able to access the database via a web server. The web server and the database server are connected, and the web server gathers data from the database server. I construct the database that is located at the DB server.

Design Science Research: The artefact that is built is the database. The artefact is evaluated experimentally by simulation. The database is used with data that is similar to real-life data stored into the database. As the data stored in the database system can easily be copied into another database system, it is even possible to use the same data in the environment. By doing this I can see whether the performance, for example, query times decrease compared to the old database. In order to be able to do this kind of evaluation some tests must be done with the old database system to get data that the results of the new database can be compared to. After testing it can be seen whether I have been able to improve the database by using the information that I have gained from the knowledge base. Building and evaluating make a cycle that is repeated until the artefact satisfies the pre-set requirements given by the case company.

Knowledge Base: The foundations of the knowledge base for development come from the existing database systems in general and the literature related to the subject. Existing literature about system must be studied in order to choose efficient and sufficient evaluation methods. The methodologies used are dependant from the case company's requirements. Sufficient methodologies are chosen from the literature that can be used also to meet the case company's requirements. The knowledge base is applied in the actual building and evaluation cycles.

1.2 Research questions

The research questions consists of one main questions and five sub-research questions. The questions are formulated from the requirements given by case company and parts of the design science research framework that are applied in this thesis. I will study previous database related research and literature to find out answers to the research questions. Most of the studied literature is from 2000s which can be considered up-to-date. However, some of the material used in this thesis is from 1990s, 1980s and 1970s. In the end I also design, implement and evaluate a new database. The results from this process are also used to answer the main research question.

The main research question of my thesis is the following:

RQ1: What database design principles and techniques should be followed to make a database well-performing and easily maintainable?

This research question must be answered because it is requested by the case company that the new database must be fast in terms of read-performance and easily maintained. Performance aspect must be studied also because the evaluation, which is a part of DSR, is done by measuring the performance of the new database. The query times of the old and the new database will be compared to achieve a proper application of DSR.

The following sub-questions will be answered as well:

RQ2: What database management systems do currently exist?

This sub-research question must be answered because I have previous knowledge that there are several tens if not hundreds of database management systems nowadays to choose from. It is also assumed by me that different database management systems are designed to match different needs. It is important to find out what database management systems currently exist so a few can be chosen for deeper studying. It is obligatory to gather knowledge base that can be applied in the build/evaluate –cycle in design science research.

RQ3: What are the differences between relational and non-relational database management systems in terms of usage and performance?

This sub-research question must be answered for two reasons. Firstly, it is known that there are differences between relational and non-relational database management systems. Secondly, it is requested by the case company that new database must be fast in terms of read-performance and easily maintained. It is essential to find out whether database management systems that are relational or non-relational fit better to meet those requirements.

RQ4: What are the differences between different relational database management systems?

This sub-research question must be answered because it is important to find out the differences between different relational database management systems to find out which one fits best to meet the requirements given by the case company.

RQ5: What are the differences between different non-relational database management systems?

This sub-research question must be answered because it is important to find out the differences between different non-relational (NoSQL) database management systems to find out which one fits best to meet the requirements given by the case company.

RQ6: How do different database management systems match the needs of a software testing team when the stored data is heavily filtered?

This sub-research question must be answered because only one database management system will be chosen to design and implement the new database with. It was agreed with the representative of the case company that the final design, implementation and evaluation phases will be done using only one database management systems because of limited resources. The DUTs are generating huge loads of log data but only a small part of the data needs to be stored; in other words it could be said that the generated log data is heavily filtered to store only the essential data.

2. Introduction to databases

In this section some basic database related knowledge and terminology that are essential to understand are gone through in order to gain a deeper understanding of databases in general. In the beginning the following matters are worth to be mentioned. The database research and the activities within the research guide the development of database technology (Feuerlicht, 2010). Database research also plays a big role in solving research problems thus database management system technology can be recognized as one of the most successful efforts in computer science (Silberschatz, Stonebraker & Ullman, 1996).

Nowadays databases are essential in every business (Garcia-Molina, Ullman & Widom, 2009). Minsky (1974) defined database as a model of some real world system that satisfies the four following properties. 1. The model consists of a large amount of coded information. 2. The database has a long life time. 3. The model can be examined and interrogated any time. 4. The model changes when operations are applied to it from the outside. Elmasri and Navathe (2009) defined a database to have the three following properties. 1. A database represents some aspect of the real world. The changes in the real world are reflected in the database. 2. A database consists of collection of data that is logically coherent. A random choice of data can't be referred to be a database. 3. A database is designed, built, and populated with data that is used for specific purpose by intended user groups.

Data is defined as facts that can have meaning and can be recorded (Elmasri & Navathe, 2009). Mahajan (2012) defined data as the main component of a database and as a collection of facts about something. A database is a collection of data (Elmasri & Navathe 2009; Ramakrishnan & Gehrke 2000; Garcia-Molina et al., 2009). Databases and database systems are an important part of everyday life in in today's society. People daily encounter situations that databases are part of. Databases can vary in size and complexity, and the information stored in traditional database applications is usually numeric or textual. Multimedia databases can store data such as pictures, videos and sound messages. (Elmasri & Navathe 2009.)

Database management system (DBMS) consists of several programs that the users can use to create and maintain a database (Elmasri & Navathe 2009). The power of databases lies in the DBMSs and database systems. The technology of those has been developed for decades. (Garcia-Molina et al., 2009.) A database and a DBMS together form a database system. The DBMS is a system that eases the process of defining, constructing, manipulating, and sharing databases among several users and applications. (Elmasri & Navathe 2009.) Ramakrishnan and Gehrke (2000) defined DBMS as a software that is designed to assist to maintain and utilize large collections of data. Garcia-Molina et al. (2009) defined DBMS in the same way as the previous definitions but also added that a DBMS allows data to persist for long periods of time.

Elmasri and Navathe (2009) defined DBMSs to facilitate the following six processes. **Defining** means that the data types, structures, and constraints for the stored data are specified. **Constructing** is the process of storing data on some storage controlled by the DBMS. **Manipulating** consists of functions for querying the database to retrieve wanted data, updating the database, and generating reports from the data. **Sharing** means that several users and programs can access the database simultaneously. **Protecting** consists of system protection and security protection. **Maintaining** means that the DBMS must be able to maintain a database system as the systems evolve and requirements change.

Garcia-Molina et al. (2009) defined that a DBMS is expected to do the following five matters. 1. It allows the users to create new databases and specify their schemas (also known as logical structure of the data). This is done by using a data-definition language. 2. It allows the users to query and modify the data. This is done by using a data-manipulation language (also known as a query language). 3. It allows the users to store large amounts of data for a long periods of time. The data access should be efficient. 4. It enables durability which means that the database should recover in case of, for example, failures and errors. 5. It allows several users to access the data at once without unexpected behavior among the users.

In the early days the database management systems were expensive software that were run on large computers. Large computers were needed because at the time the technology had not evolved to a level where hundreds of gigabytes could fit on a personal computer's disk like nowadays. However, the advantage of relation model usage today in database systems has allowed DBMSs to be part of almost every computer. These days corporate databases are used to store terabytes or even petabytes of data. This is because, for example, nowadays corporations are storing different kind of data such as pictures and videos which require more much space compared to text data. (Garcia-Molina et al., 2009.)

The first database management systems for commercial use took place in the late 1960s. They evolved from file systems. The filesystems lacked many functions that the DBMSs offer; for example, data could be lost and multi modifying for files wasn't supported. The first applications of database management systems were used, for example, in banking systems and in airline reservation systems. With those systems it was highly important that system failures didn't cause any money to disappear in the banking systems and large volume of actions had to be supported in the airline reservation systems. With the early days' DBMSs the programmer was in charge of visualizing the data. There were different data models used to describe the database structure. The most common ones were the hierarchical tree-based model and graph-based network model. Those early models didn't support any high-level query languages. (Garcia-Molina et al., 2009.)

It is important to understand what transaction processing means as database applications usually deal with it. Transaction consists of one or several operations which make up a single task. (Lake & Crowther, 2013.) Data-modelling language actions and queries are transactions. They are executed automatically and in isolation. The transactions must also be executed durably. (Garcia-Molina et al., 2009.) There are four operation categories which are the following; Create, Read, Update and Delete. An abbreviation, which is sometimes used, for the four categories is CRUD. Transaction processing has a critical role especially in multiuser systems relying on a single database. (Lake & Crowther, 2013.)

ACID is short for atomicity, consistency, isolation and durability (Lake & Crowther, 2013; Leavitt, 2010; Garcia-Molina et al., 2009). ACID is crucial for database transaction processing (Lake & Crowther, 2013). Garcia-Molina et al. (2009) stated that transactions which are properly implemented should meet the so-called "ACID test". A closer look into ACID is taken next.

Atomicity means that if any part of a transaction fails, the whole transaction fails. After that the database is returned to its original state where it was before the transaction started.

If a transaction completes without failing, the database is updated for good. **Consistency** means that the data must be valid according to defined rules when it is written. **Isolation** means that even though transactions are executed in parallel the outcome will be as if transactions were executed serially. **Durability** means that when a transaction is committed it will stay committed. (Lake & Crowther, 2013.)

The concept of data model is fundamental to understand. Data models are used to describe data or information. A data model is usually described in the following three parts: structure of the data, operations on the data, and constraints on the data. In terms of database systems, the data structures, that are used to implement data, are sometimes called the physical data model. This is somewhat misleading as they are far from the actual physical level of data. A reference to a conceptual model is sometimes used when discussing of databases to point out the level difference. In the world of databases, the operations on the data consists of queries and modifications. The limited set of operations allows for describing the operations at a very high level and implementing the DBMS operations efficiently. The data models of databases also set limits to the data. For example, a person can be limited to have only one surname to be saved into the database. (Garcia-Molina et al., 2009.)

3. Requirements of the database

In this section the requirements of the database given by the case company are explained. The requirements are essential to know and understand in order to choose an appropriate technology for the artefact in this thesis. Later in the thesis the requirements are compared to the abilities of different technologies to see which technology or technologies fit best for the case company's needs. After the comparison I can study the chosen technology deeper and start designing and implementing the database simultaneously.

The case company is not willing to spend money on the chosen technology which makes the commercial products out of choice. As I am not able to choose a technology that requires license purchasing, the database technology must be open-source. This requirement obviously reduces the amount of technologies to choose from. Hippel and Krogh (2003) defined open-source the following way. Open-source software is software that is available to everyone for free. Open-source software are developed by software developers voluntarily who feel that there is a personal or organizational need for the software. Open source projects are nowadays economic and social phenomena; there are thousands of open-source projects with thousands of developers.

The main purpose of the database is to store structured data thus relational databases can be used. According to Li, Ooi, Feng, Wang and Zhou (2008) unstructured data consists of text documents and semi-structured data consists of XML documents, for example. Relational databases are structured data. As it is not required by the case company that items that can be considered as semi-structured or unstructured data should be stored, a technology that is used to store structured data can be chosen.

It was estimated by a representative of the case company that the software stability testing team is generating new data at a rate of from 2500 to 5000 rows every hour which is reasonable as the software tests are being run around the clock on several DUTs simultaneously and the current size of the database is around 3 gigabytes. However, the case company values read-performance more compared to write-performance. Getting results fast from queries makes the usability better. The write-performance isn't valued to the same level as it does not really matter if the data storing takes a minute or three minutes, for example. It is more important that the data is faster to access. It was requested by the case company to pay attention to mechanisms that make the read-performance better. These mechanisms consists of, for example, indexing (Giacomo, 2005). According to Giacomo (2005) indexing is the most important tool to speed up queries in relational databases. In short, indexes are used to match the wanted rows faster.

The number of database users simultaneously isn't very high; approximately 20 at the most which means that the database server doesn't get much workload through web server from the users at once. However, as previously mentioned, new data is being stored into the database around the clock. The database must be able to handle the previously mentioned matters well. All the data stored is equally valuable, but it would be a plus if the data of the current software being tested was faster to access than the data from previous test runs. The data from releases that the testers are no longer interested in must be archived in a way that it can be accessed if one needs to do so later. This data is, for example, data from software release runs that are no longer in use in the field nor being tested anywhere.

Maintainability must be well-planned. The technologies that are under testing are evolving continually. The evolution brings new features and attributes that might have to be stored into the database. This probably leads to a situation where changes must be made in the database. It is impossible to know what the future brings along, so the database must be designed in a way that maintaining can be easily and efficiently done. The database must be easily maintained so that the upcoming new requirements can be met. The workload might be increasing as the area of testing is getting broader which means that scalability must be thought of as well. The database must be able to handle the increased workload.

The actual database and the availability to access data is considered more important than the analysis of the data. The stored data is structured which doesn't require complicated analytics. Essential functionalities can be hard-coded into the source code when needed. However, the support for data-analytics can be considered as an advantage but it is not a necessity and shouldn't affect the decision of the database.

Maintainability plays a big role in the database. It is very likely that some sort of changes and modification will be made in the future. However, because my contract as a thesis worker will end by the time this thesis is finished, the responsibility for maintaining the database will be laid to the person who has created the current database. This person has experience from MySQL RDBMS and naturally from the query language SQL. This leads to two requests. It would be very beneficial if the chosen technology is based on SQL or a language that is similar to SQL. Also, as the chosen technology will be open-source, the more free online support is available the better it will be. Those two matters will make the maintainability of the database easier.

The database system should store data permanently. This means that no in-memory databases should be used in this thesis. In-Memory Database (IMDB) is defined as a system that stores data on main memory. They are known for being fast. (Techopedia, 2016.) This would be beneficial in the case of this thesis but as the data must be stored permanently, IMDBs should not be considered.

To sum the previously mentioned requirements up it can be said that the two most important features of the database are a good performance and a good support for maintenance.

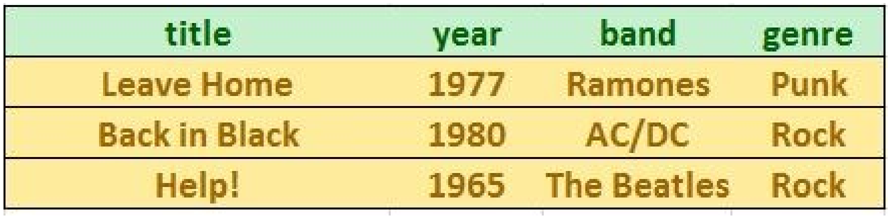
4. Relational databases – RQ1, RQ2, RQ4 & RQ6

In this section the previous knowledge found about relational databases in research and literature is gone through. This section of the thesis will answer fully or partially the following research questions: "*RQ1: What database design principles and techniques should be followed to make a database well-performing and easily maintainable?*", "*RQ2: What database management systems do currently exist?*", "*RQ4: What are the differences between different relational database management systems?*" and "*RQ6: How do different database management systems match the needs of a software testing team when the stored data is heavily filtered?*".

Edgar F. Codd is known as the father of relational database model (Leavitt, 2010). Edgar F. Codd was a mathematician who worked for IBM. The history of relational model began when Codd (1970) published his first article about it.

The uniformity of the relational model is considered as one of its major advantages. The data is stored into tables, and the tables have rows with the same format in relational databases. (Maier, 1983.) Relational databases are designed to work best with structured data (Leavitt, 2010). A database is a collection of interrelated tables and a table is a collection of interrelated records. The records are a collection of fields and the fields are the set of value that is defined by the single domain. (Sharma & Bhardwaj, 2014.)

An example of a relation inspired by Garcia-Molina et al. (2009 pp. 22) can be seen in Figure 3. The basic terminology of relations will be explained in the following sub-sections with the help of relation Albums of Figure 3. The name of the relation begins with a capital letter and the names of the attributes begin with a lower-case letter as instructed in the paper by Garcia-Molina et al. (2009 pp. 23).



title	year	band	genre
Leave Home	1977	Ramones	Punk
Back in Black	1980	AC/DC	Rock
Help!	1965	The Beatles	Rock

Figure 3. An example of a relation named Albums

4.1 Basic terminology of the relational model

Attributes are the columns of a relation. Attributes are usually presented at the top. In the Figure 3, the attributes are presented in green text in green background. The attribute tells the viewer about the entries that are seen below the attributes. When looking at the Figure 3, for example, the column with year-attribute stores the year of publication for each album. (Garcia-Molina et al., 2009.) In other words, attributes are column headers whose information is presented below (Elmasri & Navathe, 2009). In the Figure 3 year-attribute is the subject and the information, publication years of the albums, are presented below.

The attributes in the relation schema form a set (Garcia-Molina et al., 2009). Once a relation schema is introduced, the order of attributes becomes a standard, thus it should not be changed (Garcia-Molina et al., 2009; Elmasri & Navathe, 2009). Schema consists of the name of the relation and its set of attributes. The schema of the relation Albums in

Figure 3 would be “Albums(title, year, band, genre)” without the inverted commas. (Garcia-Molina et al., 2009.)

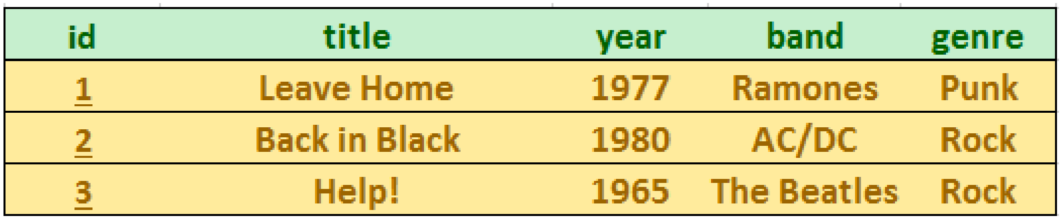
Tuples are all the rows in the relation excluding the header row. A tuple consists of one component for each attribute found in the relation. An example of a tuple from Figure 3 would be presented in the following way “(Leave Home, 1997, Ramones, Punk)” without the inverted commas. When tuples are presented without any other reference, all the attributes should be presented in order to avoid confusion. For example the previously mentioned tuple would not be ideal to present in the following way “(Leave Home, Ramones, Punk)” missing the year-attribute. (Garcia-Molina et al., 2009.)

Each component in the relational model must be atomic which means that a value should not be, for example, a set or an array, which could be divided into several components. (Garcia-Molina et al., 2009). It is important to define the data type for the values (Elmasri & Navathe, 2009). An example of a domain from Figure 3 would be presented in the following way “Albums(title:string, year:integer, band:string, genre:string)” without the inverted commas (Garcia-Molina et al., 2009).

The order of the tuples doesn’t matter because relations are sets of tuples. It would make a difference if relations were lists of tuples. Even if the order of the tuples changed, the relation would still remain the same. For example, the original order “Albums(title, year, band, genre)” could be changed to “Albums(genre, band, title, year)” without really affecting the relation, the relation is just presented in a different way. (Garcia-Molina et al., 2009.)

Relations are not unchangeable; it is normal to, for example, add new tuples or modify the existing ones. Changes within tuples are easy to make. However, changing schemas, for example, adding or a new attribute, is not so simple. These kind of matters must be taken into account when one is planning to make changes into a database. A change in the schemas might require hundreds or thousands of tuples to be re-added, depending on the size of the database. (Garcia-Molina et al., 2009.)

Key constraints are important to understand before going deeper into the relational databases. A key is formed from a set of attributes. We assume that the key is unique. If we think about the relation in Figure 3, we could decide that attributes “title” and “band” form a key. In other words this means that there can’t exist, for example, another album where the band is being “Ramones” and the album title is being “Leave Home”. If we want to express the key values, it could be done by, for example, underlining the keys the following way “Albums(title, year, band, genre)” without the inverted commas. However, it is impossible to know for sure that there doesn’t exist another album with the same name by the same band. That is why, often in relational databases, unique ID-attributes are used. For example, our example in Figure 3 could be done the following way; “Albums(id ,title, year, band, genre)” which would lead to “(1, Leave Home, 1997, Ramones, Punk)”, (2, Back in Black, 1980, AC/DC, ROCK), et cetera as seen in Figure 4. (Garcia-Molina et al., 2009.)



id	title	year	band	genre
1	Leave Home	1977	Ramones	Punk
2	Back in Black	1980	AC/DC	Rock
3	Help!	1965	The Beatles	Rock

Figure 4. An example of a relation named Albums with id-attribute.

4.2 Data types

Every attribute must have a data type. Data types are declared when a table is created. (Garcia-Molina et al., 2009.) The general data types of SQL are explained next.

There are two types of character strings; fixed length and varying length. CHAR(n) is a type for a fixed length string where *n* is the number of characters. VARCHAR(n) is a type for a varying length where *n* is the maximum number of characters. There are two types of bit strings; fixed length and varying length. The values are strings that consist of bits. BIT(n) is a type for a fixed length where *n* is the number of bits. BIT VARYING(n) is a type for a varying length where *n* is the maximum number of bits. BOOLEAN is a type that is used for attributes whose values are logical. Values can be TRUE, FALSE, and UNKNOWN. (Garcia-Molina et al., 2009.)

Types INT and INTEGER are the same. They are used for integer values. There is also a type SHORT INT which is used for integers with less bits. Types FLOAT and REAL can be used for floating-point numbers. DOUBLE PRECISION is a type that can be used for more precise values. Type DECIMAL(n,d) is used for values with decimal digits. *n* is the number of digits and *d* tells the spot of the decimal point counted from the right. Type NUMERIC can be used the same way as DECIMAL. Types DATE and TIME represent values of character strings that are in special forms. The following examples are in the standard SQL form; DATE ‘1990-01-24’ and TIME ’13:00:02.05’. (Garcia-Molina et al., 2009.)

4.3 Table declaration

The simplest way to declare a relation schema can be done the following way. Statement CREATE TABLE with the name of the relation and a parenthesized list of the attributes and the data types separated by commas afterwards. (Garcia-Molina et al., 2009.) The table in the Figure 3 could be declared the following way, for example.

```
CREATE TABLE Albums (
  title  VARCHAR(30),
  year   INT,
  band   VARCHAR(30),
  genre  VARCHAR(30)
);
```

The year attribute is an integer as not an exact date is needed. The rest of the attributes are VARCHAR as the length of the titles, bands and genres might vary.

4.4 Modifying relations

An entire table can be deleted using DROP statement (Garcia-Molina et al., 2009). The table in the Figure 3 could be deleted with the following statement.

```
DROP TABLE Albums;
```

Once this is done, the relation is not anymore in the database schema which means that it can no longer be accessed. Modifying a relation is done by statement ALTER TABLE with the name of the relation afterwards. It is possible, for example, to add and drop attributes. (Garcia-Molina et al., 2009.)

```
ALTER TABLE Albums ADD singer varchar(30);
```

The previous statements adds a new attribute singer into the relation Albums seen in the Figure 3.

```
ALTER TABLE Albums DROP year;
```

The previous statement deletes the existing attribute year from the relation Albums seen in the Figure 3.

4.5 Declaring keys

Key declaration is executed within CREATE TABLE statements. A key can be an attribute or a set of attributes. It is possible to declare an attribute to be a key when the relation schema is set or an additional declaration can be added to the schema that tells which attribute or a set of attributes forms the key. The latter method must be used if a set of attributes forms the key. Both methods can be used if the key is only one attribute. (Garcia-Molina et al., 2009.)

A key can be declared in two ways: PRIMARY KEY and UNIQUE. When PRIMARY KEYS are used, the attributes set can't be NULL. When UNIQUE is used, an attribute can be NULL. (Garcia-Molina et al., 2009.) The previously mentioned knowledge could be applied the following way continued on the example presented in section 4.3 Table declaration. If I assume that there are no movies with the same name published in the same year, I can declare attributes title and year to be a PRIMARY KEY.

```
CREATE TABLE Albums (
  title         VARCHAR(30),
  year          INT,
  band          VARCHAR(30),
  genre         VARCHAR(30),
  PRIMARY KEY   (title, year)
);
```

4.6 Database queries

In this sub-section the previous research and literature found about database queries are gone through. Attention should be paid to database queries as they are the way to operate with the databases.

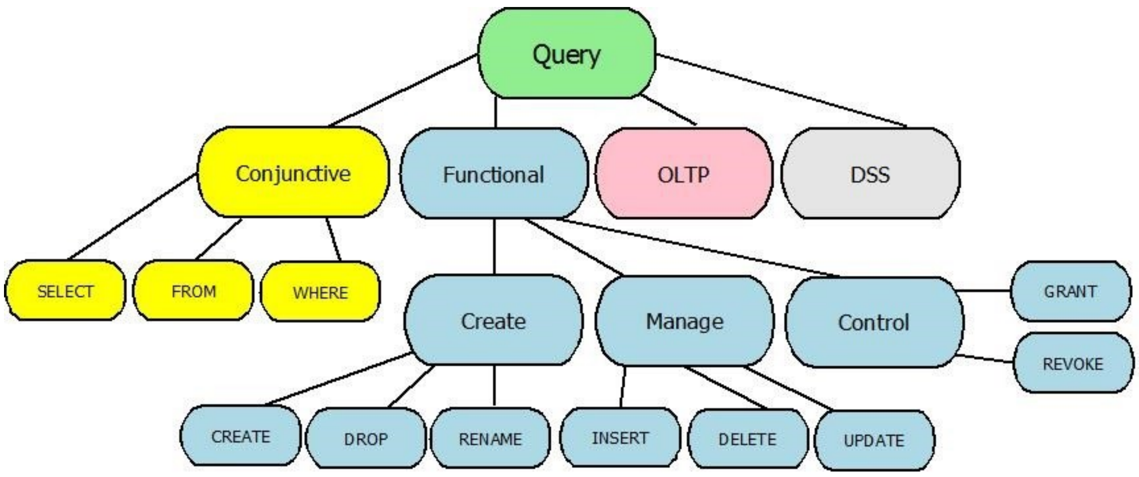


Figure 5. Types of database queries based on Sharma and Bhardwaj (2014) and Wisnesky (2014).

The easier it is to retrieve data from a database the more valuable it is to the users (Ramakrishnan & Gehrke, 2000). All data operations within a database can be done with queries. They are used in different settings. However, the most used application is requested by the user who wants to retrieve data from the database. (Sharma & Bhardwaj, 2014.) A query language, which is used for submitting queries, comes along with a DBMS (Ramakrishnan & Gehrke, 2000). Relation models are known for supporting query languages. Relational calculus and relational algebra are formal query languages. (Ramakrishnan & Gehrke, 2000; Vardi, 1982.) Relational calculus is a logical non-procedural language. When relational calculus is applied in a database, a set of tuples is returned to answer the query. Relational algebra is an algebraic procedural language. It consists of basic algebraic operations such as a join-operation. (Vardi, 1982.) Relation algebra is not nowadays widely used as such, but it is included, for example, in the well-known query language SQL (Garcia-Molina et al., 2009).

A data manipulation language is used in a DBMS to, for example, create and query data. It should be noticed that a query language makes only one part of the data manipulation language. (Ramakrishnan & Gehrke, 2000.) Queries can be categorized in four different categories. Functional, online transaction processing system (OLTP) queries, and decision support system (DSS) queries. Functional queries consist of create, manage and control queries. Create query consists of create, drop and rename statements. Manage query consists of insert, delete and update statements. Control query consists of grand and revoke statements. (Sharma & Bhardwaj, 2014.) Conjunctive queries consist of select, from, and where (Wisnesky, 2014). Different types of queries are presented in Figure 5. The figure is modified from figure by Sharma and Bhardwaj (2014) and I added conjunctive queries presented by Wisnesky (2014) into it.

Database structures have been redesigned at the same time as business requirements have become more complex. An OLTP system is one outcome of that process. OLTP is a system that is based on relational theory. Tuples are arranged in rows and the rows are stored in blocks in OLTP systems. The blocks are stored in disk, and the database server serves as a platform for caching the blocks in main memory. Refined indexing makes it fast to access single tuples in OLTP systems. It should be noted OLTP systems are not eternally fast; the higher the number of requested tuples is the slower the access becomes. (Plattner, 2009.) OLTP systems are optimized for managing changing data. Live databases where several users are performing several transactions simultaneously, i.e. real time changes are made, are a good use for OLTP systems. (Sharma & Bhardwaj, 2014.)

Decision support is a multibillion dollar industry nowadays and it will grow even more in the future. Organizations are interested, for example, in analyzing and comparing previous data and current data to help making big decisions within the company. The vendors of relational DMBS have realized the need of new features and now the new features are added into their products. (Ramakrishnan & Gehrke, 2000.) The tables in DSS databases are greatly indexed, and in order to support different kind of queries the data in a DSS database is preliminary processed. DSS queries are naturally distributed as they are used to retrieve data from several sources. DSSs are used for combined information. When compared to OLTP, DSS' queries take more execution time and they require more system resources. The execution times of queries are also longer and harder to predict than the queries in OLTP systems. (Sharma & Bhardwaj, 2014.) However, view queries can be run faster if they are precomputed (Ramakrishnan & Gehrke, 2000).

Different kind of needs such as OLTP and DSS queries might be needed alternately which must be taken into account when designing a database system. The following good example of a bank's database is explained to understand this better in the paper by Elnaffar, Martin and Horman (2002). A bank's database must be able to handle OLTP queries daily for almost a month as regular transactions are made by several users. However, in the end of the month, the queries become more DSS-like as, for example, financial reports and summaries are queried from the system.

4.7 Indexes

An index is a helping data structure that is meant to help find records of desirable search conditions. In other words, indexes are used to make queries faster. (Elmasri & Navathe, 2009; Ramakrishnan & Gehrke, 2000.) Entry is the name used for the records stored in an index file. Entries are used to find data records with a given search value. All indexes have a search key. It consists of one or several fields of the file. Indexes should be built on the attributes that the wanted speeded up queries are related to. (Ramakrishnan & Gehrke, 2000.)

4.8 Database normalization

Normalization of databases is considered as important because it helps to build a logical structure that is easy to be maintained. The formal reason for normalization is the ease of retrieving data from the database. Normalized databases have also become an industry standard; it is easier for other people to work with one's databases if they are normalized. More time and effort, complex procedural code for example, are needed to get information from a database that is not normalized. (Dorsey & Rosenblum 2006.) Normalization is used to see whether the design of the database need to be restructured by adding, for example, new tables into the database (Churcher, 2007). The rules of normalization are taken into use basically by eliminating redundancy and inconsistent dependency in the table designs (Wise, 2000). The first three rules of normalization are basic that everyone working with databases should be familiar with (Dorsey & Rosenblum 2006).

If a normal form is applied in a relation schema it can be said with certainty that particular problems will not rise. There are several normal forms that relations can be in. It should be noted that if a relation is in the second normal form the same relation is also in the first normal form, and if a relation is in the third normal form the same relation is also in the second formal norm, for example. (Ramakrishnan & Gehrke, 2000.) A closer look into how to apply the forms of normalization is taken next. The following examples of normalization are slightly modified from Wise (2000).

4.8.1 Zero form

Figure 6 presents a table named "users" that has been created to store the following attributes; name, company, company address, and some urls for each user. The table is in zero form as no normalization rule has been applied yet to it. (Wise, 2000.) The next sub-section, 4.8.2. First normal form, solves the problem of adding a third url into the table.

users				
name	**company**	**company_address**	**url1**	**url2**
Joe	ABC	1 Work Lane	abc.com	xyz.com
Jill	XYZ	1 Job Street	abc.com	xyz.com

Figure 6. Table *users* in zero modified from Wise (2000).

4.8.2 First normal form

The first normal form consists of the following three rules. 1. "Eliminate repeating groups in individual tables." This is solved by making only one url-attribute. 2. Create a separate table for each set of related data." 3. "Identify each set of related data with a primary key." This is done by adding a new unique attribute "userId". The outcome of applying the first normal form is presented in Figure 7 below. A few problems have been solved so far but there is still more normalization to do. Every time a new record is stored into the table, company and user name data must be duplicated. The database grows bigger than intended and needed. (Wise, 2000.) This is solved in the next sub section 4.8.3 Second normal form.

users				
userId	**name**	**company**	**company_address**	**url**
1	Joe	ABC	1 Work Lane	abc.com
1	Joe	ABC	1 Work Lane	xyz.com
2	Jill	XYZ	1 Job Street	abc.com
2	Jill	XYZ	1 Job Street	xyz.com

Figure 7. Table *users* in the first normal form modified from Wise (2000).

4.8.3 Second normal form

The second normal form consists of the following two rules. 1. "Create separate tables for sets of values that apply to multiple records." 2. Relate these tables with a foreign key." The duplication problem presented in the section 4.8.2 is solved by creating a separate table for the URLs named "urls". Primary key values are used to relate to urls fields. The solution after applying the second normal form is presented in Figure 8 and Figure 9. The next problem occurs when one wants to add more employees of company ABC. Company names and company addresses get duplicated if the tables remain the same. (Wise, 2000.) This is solved in the next sub section 4.8.4 Third normal form.

users			
userId	name	company	company_address
1	Joe	ABC	1 Work Lane
1	Joe	ABC	1 Work Lane
2	Jill	XYZ	1 Job Street
2	Jill	XYZ	1 Job Street

Figure 8. Table *users* after applying the second normal form modified from Wise (2000).

urls		
urlId	relUserId	url
1	1	abc.com
2	1	xyz.com
3	2	abc.com
4	2	xyz.com

Figure 9. Table *urls* after applying the second normal form modified from Wise (2000).

4.8.4 Third normal form

The third formal norm tells to eliminate fields that are not dependent on the key. In this case this means that the attributes company and company_address will have an own table "companies" with an id. Now it is possible to add new users without duplicating the company name. (Wise, 2000.)

users		
userId	name	relcompId
1	Joe	1
2	Jill	2

Figure 10. Table *users* after applying the third normal form modified from Wise (2000).

companies		
compId	company	company_address
1	ABC	1 Work Lane
2	XYZ	1 Job Street

Figure 11. Table *companies* after applying the third normal form modified from Wise (2000).

urls		
urlId	relUserId	url
1	1	abc.com
2	1	xyz.com
3	2	abc.com
4	2	xyz.com

Figure 12. Table *urls* after applying the third normal form modified from Wise (2000).

4.9 Different relational databases

MySQL, PostgreSQL, Microsoft SQL server, Oracle, and DB2 were the five most popular relational database management systems at the time of this study according to DB-Engines (2016a). Because a commercial database system can't be chosen for this thesis, only MySQL and PostgreSQL are studied as they are open-source products. Many studies about different relational databases include commercial products as well but they will not be taken into account. They might be mentioned in the comparison to get a better understanding of the results of the studies but they are not considered as an option for this study's artefact.

4.9.1 MySQL

MySQL is the most popular open source SQL DBMS. It is nowadays developed, distributed, and supported by Oracle Corporation. Being open source means that MySQL can be used and modified by anyone. It is also possible to buy a commercially licensed version of MySQL if one doesn't agree what is told in the General Public License that is used by MySQL. (MySQL, 2016a.)

MySQL is advertised as very fast, reliable, scalable, and easy to use. The database software of the MySQL is a client/server system. It comes with support for different backends, different client programs and libraries, admin tools, and several APIs. MySQL is written in C and C++. It works on different platforms such as different Microsoft Windows, Apple OS X, and Linux platforms. Transactional and non-transactional storage engines are provided in MySQL. The high speed comes from B-tree disk tables with index compression, thread-based memory allocation system, optimized nested-loop joins, in-memory temporary tables, and highly optimized class library. (MySQL, 2016a.)

Many data types are supported in MySQL along with SQL statements and functions. MySQL includes a password system that enables host-based verification. MySQL is known for supporting large databases; even databases with 50 million records. Index support is up to 64 indexes per table. APIs for many languages such as C, C++, PHP and Python are available. There are several client and utility programs in MySQL both command-line and graphical programs, for example mysqldump, mysqladmin and MySQL Workbench. (MySQL, 2016a.)

4.9.2 PostgreSQL

PostgreSQL is a client/server relational database with its history beginning in the late 1970s. A lot of new features have been added into PostgreSQL until today and nowadays PostgreSQL is a free open source software that is developed by an international group called PostgreSQL Global Development group. PostgreSQL is recognized as an advanced database server. (Douglas & Douglas, 2003.)

PostgreSQL comes with the following features, for example. PostgreSQL is **object-relational** which means that every table defines a class, there is inheritance between tables and classes, and functions and operators are polymorphic. PostgreSQL is **standard compliant** which means that most of the SQL92 along with many features of SQL99 are implemented in PostgreSQL syntax. **Transaction processing** in PostgreSQL is handled in a way that it protects data and multiple users are coordinated through full transaction processing simultaneously. **Referential integrity** is included which means that PostgreSQL foreign and primary key relationships along with triggers are supported. **Multiple procedural languages** are supported which means that different procedural languages can be used to write triggers, for example. Server side can be coded in PL/pgSQL, TCL, Perl and bash. PostgreSQL supports **multiple-client APIs** which means that client applications can be developed in many languages such as C, C++, PHP and Python. PostgreSQL provides different data types along with **unique data types** such as geometric types. PostgreSQL is **extendable** which means that needed features, such as new data types, can be added by oneself. (Douglas & Douglas, 2003.)

4.10 Choosing the right SQL database

All databases are built for the same purpose which is data storing. However, when looking at, for example, performance and reliability of different DBMSs they differ from each other. (Malaysian Public Sector, 2009.) When looking at the descriptions of different relational DBMSs it is hard to see big differences between them, especially between the open source products MySQL and PostgreSQL. However, it is important to understand the differences and features of each in order to make a choice that fits best for the requirements given by the customer. At the time of the study, in 2016, it was difficult to find up-to-date studies, comparisons and benchmarks of open-source relational database management systems, especially MySQL and PostgreSQL. It seems that nowadays the database related studies are more focused on non-relational databases. Because of the previously mentioned problematic situation, the information was gathered from older studies and articles.

Comparisons between different DBMSs have been made by different researchers. Conrad (2004) studied the differences between MySQL, PostgreSQL and commercial databases such as Oracle, DB2, and Microsoft SQL server. The following statements are based on his article. **Licenses:** MySQL has two licenses to choose from, a general public license and a commercial license. GPL means that the source code of one's own must be shared and the project can't be distributed. Commercial license must be purchased if one intends to make the database into commercial use. PostgreSQL's licensing scheme is released under the Berkeley License which means that the code can be used or released as a commercial product without sharing the source code. **Data storage:** There are several mechanisms to store data in MySQL but it mainly uses InnoDB. PostgreSQL uses only one mechanism to store data called Postgres storage system. **Data integrity:** MySQL and PostgreSQL are both ACID compliant. (Conrad, 2004.) Isolation stands for the "I" in ACID. There are four isolation levels: Repeatable Read, Read Committed, Read Uncommitted and Serializable. MySQL supports all four levels. (MySQL, 2016c.) PostgreSQL supports three levels: Read Committed, Repeatable Read and Serializable (PostgreSQL, 2016). Both of them also have a support for partial rollbacks of transactions and both of them can handle deadlocks. MySQL uses row-level locking and PostgreSQL uses Multi Version Concurrency Control. **Stored procedures and triggers:** Both PostgreSQL and MySQL have support for triggers and stored procedures. **Indexes:** Both have support for the following indexes: single column, multi-column, unique and primary keys. **Data types:** Both have support for ordinary data types. Users can define their own data types in PostgreSQL but this is not possible in MySQL. **Replication:** Both support single-master and master-slave replication in the open-source releases. **Platform support:** Both have support for Windows, MacOSX and Linux. **Database interface methods:** Along with native methods for accessing the database, both have support for ODBC and JDBC. Both also have support for Java, Perl, Python, PHP and C and C++. **Database authentication:** MySQL stores authentication information into a table and the data from the table is compared to the credentials. PostgreSQL has support for same kind of system along with others such as a local authentication system that is based on a UNIX password. **Backups:** Both have possibilities for hot database backups and backing up a database without performing a shutdown. Both DBMSs can recover from soft crashes and power failures. Overall, the backup technologies are not in the same level as in commercial products. **GUI tools:** There are open-source and commercial tools available for both DBMS that can be used to manage databases. **Data migration:** Both have open-source and commercial support to migrate data from commercial DBMSs. (Conrad, 2004.) The previously written comparison is presented in Table 1.

Starcu-Mara and Baumann (2008) conducted a comparative benchmark of the leading commercial and open-source databases that could handle Binary Large Objects (BLOB). The open-source databases were PostgreSQL and MySQL. The commercial ones stayed anonymous. Readings and writings were measured in the study. PostgreSQL version was 8.2.3 and MySQL version was 5.0.45 at the time. In the study it turned out that MySQL had a better write-performance than PostgreSQL. MySQL was also more stable during writing compared to PostgreSQL. MySQL had a better read-performance than PostgreSQL as well. Both, MySQL and PostgreSQL, had excellent scalability. MySQL was declared as the winner in the study. (Starcu-Mara & Baumann, 2008.)

Giacomo (2005) studied different databases to fit the following requirements. Several users simultaneously must be supported, easy access from APIs written in different languages must be supported, support for ACID was needed, search capabilities such as joins and sub-selects must be supported, online backups must be created while the database is in read- or write-state, support for large amounts of data was needed while a the performance is high, and high availability was needed. It turned out that MySQL was a better choice than PostgreSQL. However the differences between the two were very slight. MySQL seemed to have a better scalability and embedded replication than PostgreSQL. MySQL was considered as a fast database in general in the paper. (Giacomo, 2005.)

According to DB-Engines (2016b) the system properties of MySQL and PostgreSQL were very similar. However, a few differences that were listed should be mentioned. MySQL was more popular than PostgreSQL. MySQL also supported more programming languages and MySQL included partitioning methods that were not included in PostgreSQL. A recap of the comparison studies by Starcu-Mara and Baumann (2008), Giacomo (2005) and Malaysian Public Sector (2009) are found in Table 4.

According to a study conducted by Malaysian Public Sector (2009) MySQL has been said to be faster than PostgreSQL for a long time. However, PostgreSQL was said to be a more featured open-source version of Oracle. MySQL was known for its easier usability as well. In the same study it was said that this information was outdated which is why there was a new comparison made between MySQL and PostgreSQL. Features and the speed of queries were compared between the two. Features such as raw sweep, concurrency, ACID support, data types, triggers, licensing, community support and database administration tools were compared. It turned out that there weren't much differences between the features of the two DBMSs. A benchmark was made on Drupal to see which database was faster. Similar test configurations were set up for both MySQL and PostgreSQL. Data was added into the databases and the results were collected. It turned out that MySQL was faster than PostgreSQL. However, PostgreSQL became faster when there was more concurrency.

A closer look into indexes is taken in Table 2. Indexes are a way to make queries faster which is one of the requirements for this thesis. The comparison is based on information found in WikiVS (2016).

Table 1. Differences between MySQL and PostgreSQL by Conrad (2004).

	MySQL	**PostgreSQL**
License	OS and commercial	OS
Platforms	Windows, MacOSX, Linux	Windows, MacOSX, Linux
Model	Client/server	Client/server
Data storage	Mainly InnoDB	Postgres Storage System
ACID	Yes	Yes
Data type support	General	General & possibility for own definitions
Replication	Single-master & Master-slave	Single-master & Master-slave
ODBC & JDCB support	Yes	Yes
Stored procedures	Yes	Yes
Triggers	Yes	Yes
Data migration tools	OS and commercial	OS and commercial
GUI tools	OS and commercial	OS and commercial
Back ups	Ability to recover from soft crashes and power failures	Ability to recover from soft crashes and power failures

Table 2. Index comparison of MySQL and PostgreSQL based on WikiVS (2016).

	MySQL	**PostgreSQL**
Hash indexes	Yes	Yes
Multiple indexes	Yes	Yes
Change buffering	Yes	No
Full-text indexes	Yes	Yes
Partial indexes	No	Yes
Prefix indexes	Yes	Yes, part of expression indexes
Multi-column indexes	Yes, up to 16 columns per index	Yes, up to 32 columns per index
Bitmap indexes	No, but can be achieved with index_merge feature	Yes
Expression indexes	No, but can be emulated	Yes
Non-blocking CREATE INDEX	Yes, depends on storage engine	Yes
Covering indexes	Yes	Yes

Table 3. Popularity and free online support for MySQL and PostgreSQL

	MySQL	**PostgreSQL**
Page loads of official websites	65 150 000	21 100 000
Wikipedia page loads	603 486	350 900
Stack Overflow	391 774	52 562

Table 4. MySQL vs. PostgreSQL. A recap of the comparison studies by Starcu-Mara and Baumann (2008), Giacomo (2005) and Malaysian Public Sector (2009).

	MySQL	**PostgreSQL**	**Source**
Better read-performance	X		(Starcu-Mara & Baumann, 2008)
Better write-performance	X	Better when more concurrency.	(Malaysian Public Sector, 2009)
Better stability	X		(Starcu-Mara & Baumann, 2008)
Better scalability	X		(Giacomo, 2005)
Better API support	X		(Giacomo, 2005)

The popularity and online support found for two SQL databases, MySQL and PostgreSQL, are presented in Table 3. The data for the attribute “Page loads of official websites” was gathered using a tool named SimiliarWeb found in https://www.similarweb.com/. The number of total visits for the official websites of the NoSQL technologies was gathered from October 2015 – March 2016 period. The final result is rounded. In this case the terms MySQL and PostgreSQL are not easily to confuse as there are no multiple meanings for them which means that people probably don’t accidentally visit those Wikipedia pages.

The data for the attribute “Wikipedia page loads” was gathered using Pageviews Analysis tool from https://tools.wmflabs.org/pageviews/. The number of page loads for the Wikipedia pages of the NoSQL technologies in English was gathered from August 1st 2015 to April 18th 2016.

The data for the attribute “Stack Overflow” was gathered from the official website of Stack Overflow from http://stackoverflow.com/. Both previously mentioned SQL databases were used as an entry to see how many questions were found related to each technology. The amount of questions found indicates how much free support is easily available online.

From the results presented in the Table 3 it can be deduced that MySQL is more popular than PostgreSQL, and MySQL has more free online support available. However, it should be mentioned that the popularity and free online support for PostgreSQL isn’t low.

4.11 Meeting the requirements

In this section it is explained how the previously mentioned relational databases, MySQL and PostgreSQL, meet the requirements given by the case company. The requirements and requests were explained in section 3. Requirements of the database. In this section the requirements are shortly mentioned in **bolded** text and how the studied technologies fit into them is explained. A more detailed selection process of the database chosen for this thesis is explained in section 8. Choosing the database.

The chosen technology must not cost any money. Both relational databases, MySQL and PostgreSQL, are open-source that are available to everyone for free which means that those both technologies meet this requirement equally. **The database must be able to store structured data.** MySQL and PostgreSQL are relational databases which are designed to store structured data which means that those both technologies meet this requirement equally as well.

Read-performance is valued by the case company. As seen in Table 4, MySQL has a better read-performance than PostgreSQL which makes MySQL to meet the requirement better. It is also important to look at index support because according to Giacomo (2005), as previously mentioned, indexes are the most important tool to make queries faster in relational databases. When looking at Table 2, it can be seen that both technologies have advanced indexing system. 11 different index mechanisms were compared and there is one mechanism that is supported in PostgreSQL but not in MySQL. If read-performance is evaluated from index support point of view, then PostgreSQL slightly beats MySQL. **Write-performance** isn't considered as important as read-performance. However, thousands of rows of new data is generated around the clock which is stored into the database so it wouldn't harm to have a database system that performs well in write executions. It is seen in Table 4 that MySQL has a better write-performance when there is not much concurrency. There will be not much concurrency in the environment where the database lies which means that MySQL is a better fit from write-performance point of view.

Scalability is valued because it is known that the workload of the database will increase in the future as the technologies that are under testing evolve continually which will probably increase the workload. From Table 4 it can be seen that MySQL has a better scalability than PostgreSQL. **Archiving** is required function. From Table 1 it can be seen that both, MySQL and PostgreSQL, support replication which can be used for archiving data. Both databases meet this requirement equally well.

The better **maintainability** the technology offers, the better fit it is. The person who will maintain the database system in the future is familiar with MySQL, so support for SQL and naturally MySQL can be considered as pluses. Both technologies are SQL based which means that both of them meet the requirement equally well. The more free online support there is available the easier the maintainability is. From Table 1 it can be seen that both have GUI based tools available which can be considered as helpful tools when considering maintainability. From Table 3 it can be seen that MySQL is more popular and offers more free online support which makes MySQL a better choice from this requirement point of view.

The database system should be able to **store data permanently**. In other words it shouldn't be an in-memory database. A term RAM-based database seems to be sometimes used for an in-memory database as well, for example, in the study by Lourenço, Cabral, Carreiro, Vieira and Bernardino (2015) Neither of the database systems is an in-memory database which means that both databases meet this requirement equally.

From the previous comparison it can be deduced that MySQL meets the requirements of the database system given by the case company better than PostgreSQL. However, it should be mentioned that PostgreSQL can't be considered as a significantly worse fit for the database system than MySQL. The differences are only slight.

A similar comparison of NoSQL databases is made in section 5.4 Meeting the requirements. After that suitable candidates are chosen and the final decision of the database will be made in section 8. Choosing the database system. MySQL will be compared to the non-relational databases that met the requirements.

5. Non-relational databases – RQ2, RQ3, RQ5, RQ6

In this section the previous knowledge found about non-relational databases in research and literature is gone through. This section of the thesis will answer fully or partially the following research questions: "*RQ2: What database management systems do currently exist?*", "*RQ3: What are the differences between relational and non-relational database management systems in terms of usage and performance?*", "*RQ5: What are the differences between different non-relational database management systems?*" and "*RQ6: How do different database management systems match the needs of a software testing team when the stored data is heavily filtered?*".

SQL relational databases are still the dominant database technology today (Feuerlicht, 2010). However, during the last few years, companies and science have started to question whether there are other solutions for data storing than relational databases (Strauch, 2011; Feuerlicht, 2010). NoSQL is a term that is used to describe the new data store movement. NoSQL as a term was first used for a relation database that had left out the use of SQL in 1998. Over ten years later, in 2009, the term was used again in conferences where people advocated non-relation databases as an alternative to relational databases for situations where relational databases were not a good fit to solve certain problems. (Strauch, 2011.) NoSQL is not intended to replace SQL but it is intended to provide something that SQL is lacking so they can co-exist (Nayak, Poriya & Poojary, 2013). Non-relational databases are usually called NoSQL-databases. Organizations that collect a lot of unstructured data are starting to use more non-relational databases. NoSQL databases enable better performance which is beneficial when there are huge amounts of data in the databases. (Leavitt, 2010.)

Different NoSQL databases exist that have different approaches. However, what they have in common is that none of them is relational. (Leavitt, 2010.) It has been estimated that structured data is only 5% of total data generated. The rest of the data is either unstructured or semi-structured. It is more difficult to manage data that is not structured. Database research faces new challenges because the amount of data is rising very quickly. (Feuerlicht, 2010.) The main advantage of NoSQL databases is that they can handle unstructured data which relational databases don't handle. Unstructured data consists of multimedia, emails, and word-processing files, for example. Many organizations and companies have developed NoSQL databases. Dynamo and Big Table NoSQL databases were developed by Amazon and Google as Web 2.0 rose the amount of data was growing as well. Those databases have inspired many of today's other NoSQL databases. (Leavitt, 2010.)

In the article written by Lai (2009) it can be seen that there are NoSQL advocates that resist traditional relation database systems. They believe that relational databases are slow and expensive for today's needs; they claim that NoSQL databases offer more efficient and cheaper ways to manage data. Oracle and MySQL have been popular to use among start-up companies but nowadays, especially, Web 2.0 start-up companies are building data stores inspired by NoSQL technologies. NoSQL seems to be a better fit for today's huge needs of data that are used in cloud computing, for example. (Lai, 2009.)

NoSQL databases are mostly open-source. It is common that disruptive software related trends do often better in open-source environment as users do the technical evaluations which is very low-costly. The three most popular types of NoSQL databases are key-value stores, column-oriented databases, and document-based stores. (Leavitt, 2010.) I will go through them next in more detail.

Key-value store systems can store structured and unstructured data. The stored values are indexed for retrieval keys. (Leavitt, 2010.) Key-value stores are said to be simplistic but efficient and powerful. The stored data is usually an object or a data type of a programming language. The key-value pair consists of data in two parts. The first part is a string that represents the key and the second part is the actual data that can be referred as value. Hash tables that use keys are indexes that are similar to these stores. The data model is simple which makes it faster than relational database management systems. A map or a dictionary is used to get the key specified values the user requests. High scalability is preferred over consistency which means that querying and join operations, for example, are not used. The advantages of key-value store databases are high concurrency, fast look-ups and options for mass storage. One main weakness is the lack of schema so customized views of the data are hard to obtain. Key-value data stores are used, for example, in online shopping websites where one wants to store user's shopping carts to get more detailed data. (Nayak et al., 2013.) An example of key value pairs is presented in Figure 13. From the figure it can be seen that one key can have one or several values. Key DD11 only has value "John", and key DD22 has two values "Ferrari" and "Red".

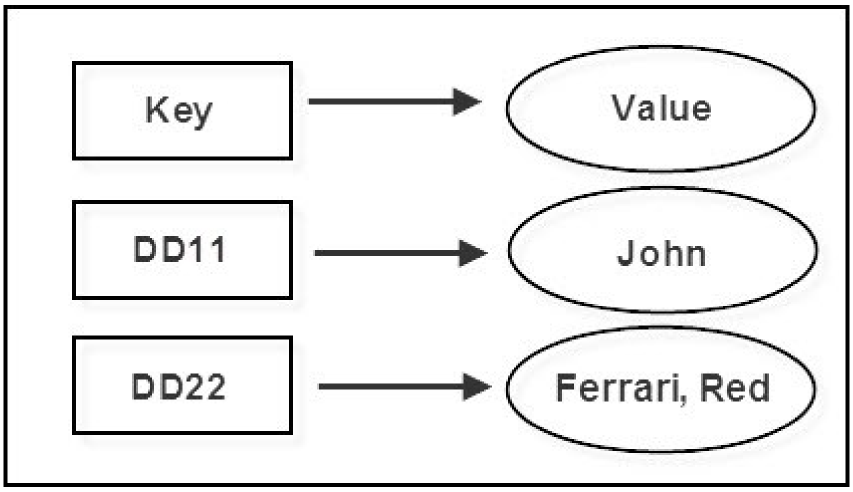
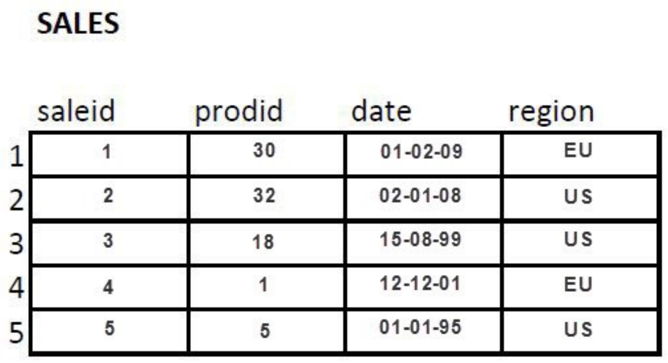


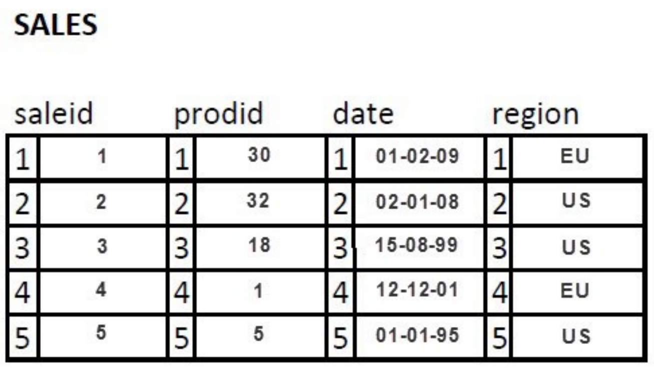
Figure 13. An example of key value pairs.

A column-oriented database is vertically partitioned into a set of individual columns. The columns are stored in the column-store system separately. This allows for not having to read entire rows when queries are executed; queries are enabled to read just the needed attributes instead. (Abadi, Boncz, Harizopoulos, Idreos, Madden, 2013.) The differences of physical layouts of column-oriented and row-oriented databases can be seen in the Figures 14, 15 and 16 that are modified from Abadi et al. (2013 pp. 199). Figures 14 and 15 show how all columns are stored independently as separate data-objects. Data is read from storage and it is written into storage in blocks. Each block that is holding data for the sales tables is holding data only for one of the columns. In a row store, seen in Figure 16, a single data object contains all the data for the sales tables. This means that it is not possible to read just the needed attributes because queries go through the surrounded attributes as well. (Adabi et al., 2013.) The main advantage of the column-oriented databases is the high scalability (Nayak et al., 2013). A column-store approach is more efficient compared to row-store approach when queries are executed because less data must be gone through in order to find the needed attributes. Column-oriented databases are mostly used in data mining and in analytic applications. (Nayak et al., 2013.)


SALES

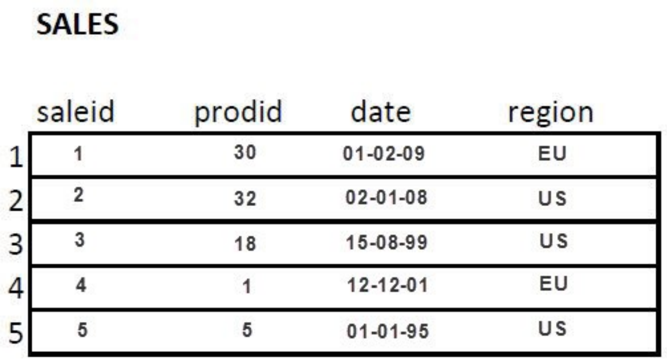
	saleid	prodid	date	region
1	1	30	01-02-09	EU
2	2	32	02-01-08	US
3	3	18	15-08-99	US
4	4	1	12-12-01	EU
5	5	5	01-01-95	US

Figure 14. Column Store with virtual Ids. Modified from Abadi et al. (2013 pp. 199).


SALES

saleid		prodid		date		region	
1	1	1	30	1	01-02-09	1	EU
2	2	2	32	2	02-01-08	2	US
3	3	3	18	3	15-08-99	3	US
4	4	4	1	4	12-12-01	4	EU
5	5	5	5	5	01-01-95	5	US

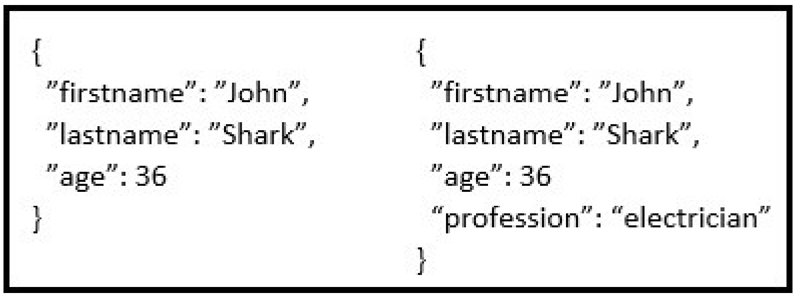
Figure 15. Column Store with explicit Ids. Modified from Abadi et al. (2013 pp. 199).


SALES

	saleid	prodid	date	region
1	1	30	01-02-09	EU
2	2	32	02-01-08	US
3	3	18	15-08-99	US
4	4	1	12-12-01	EU
5	5	5	01-01-95	US

Figure 16. Row Store. Modified from Abadi et al. (2013 pp. 199).

Data as collections of documents are stored and organized in **document-based stores**. The users can add fields of any number or length into a document, unlike with structured tables where the fields are pre-defined for their sizes, for example. (Leavitt, 2010.) This is possible because document-oriented databases are schemaless; new key-value pairs can be added into the documents to represent the new fields (Couchbase, 2016). An example of this can be seen in Figure 17. There are two documents and the latter one has an attribute that the first one doesn't have even though they belong to the same collection. This makes the documents much more flexible. Document stores also allow great performance alongside with great horizontal scalability options. The database stores documents which are addressed using a unique key representing the document. The key can be, for example, a simple string referring to URI. Document store databases are used in applications where special characteristics in documents are needed. Document stores should not be used if relations and normalization are needed. Instead, they should be used when the domain model can be split into several documents. They are used, for example, in blog software. (Nayak et al., 2013.)



```
{                                   {
  "firstname": "John",                "firstname": "John",
  "lastname": "Shark",                "lastname": "Shark",
  "age": 36                           "age": 36
}                                     "profession": "electrician"
                                    }
```

Figure 17. Two documents with different attributes in the same collection.

Alongside with the previous three NoSQL databases, there are also graph and object oriented NoSQL databases. In **graph databases** the data is stored in the form of a graph. Nodes and edges constitute the graph. Nodes are the objects and edges are the relationship between the objects. There are also properties related to nodes in the graph. A technique called index free adjacency is used which means that every node has a direct pointer that points to the adjacent node. This technique allows to traverse through huge amount of records. Graph databases are ACID compliant and they are mostly used to store semi-structured data in applications such as social networking applications. (Nayak et al., 2013.) An example of a graph database with objects and relationships between them can be seen in Figure 18 that is modified from Neo4j (2016). There are three objects; users John, Sarah and William. The follow arrows, edges, show the relationships between each object.

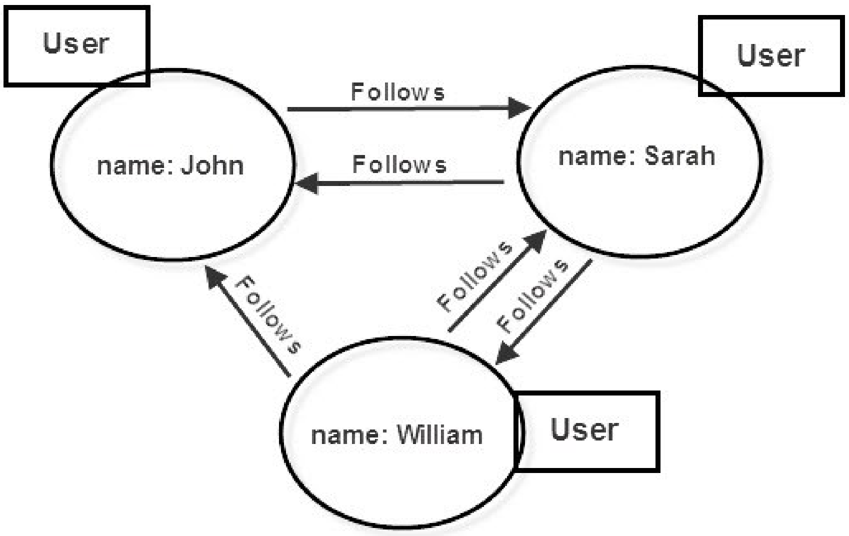


Figure 18. An example of a graph database with objects and relationships between them modified from Neo4j (2016).

Object oriented databases are a combination of object oriented programming and database principles. The data or information is stored as an object. Familiar features from object oriented programming such as data encapsulation, polymorphism and inheritance are offered. A relational database has tables, tuples and columns that are comparable to OOD's class, objects and class attributes. Every object has a unique object identifier. Access to data is fast in object oriented databases. OODs are used in scientific research and telecommunication, for example. If the application includes complex object relationships or changing object structures, OOD is a good choice. If the relationships and data are simple, OOD should not be used. OODs are also tied to a specific programming language which can be seen as a downside. (Nayak et al., 2013.) A simple example of object-oriented data model modified from Hauer (2015) can be seen in Figure 19.

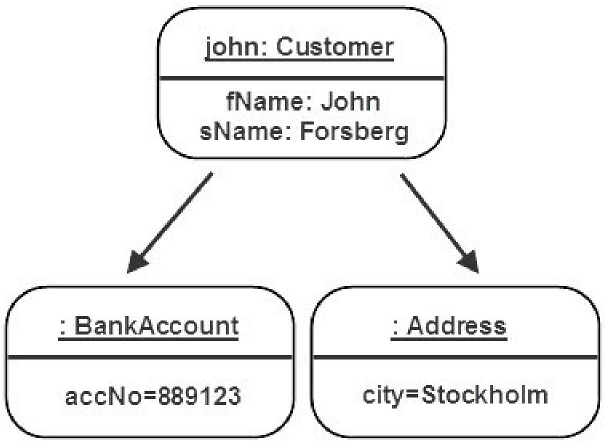


Figure 19. A simple example of object-oriented data model modified from Hauer (2015).

5.1 Advantages and concerns of NoSQL databases

NoSQL offers the following four main aspects as advantages. 1. The data can be read and written quickly, 2. NoSQL supports mass storage, 3. NoSQL is easy to expand, 4. NoSQL is low cost. (Han, Haihong, Le & Du, 2011.) NoSQL databases are usually faster than relation databases when talking about data processing speed. NoSQL databases are not usually ACID supported. This allows the increased performance as the restraints don't have to be performed on every bit of data as in relational databases. However, as the performance increases, the precision might decrease at the same time. (Leavitt, 2010; Nayak et al., 2013.) Precise transactions are often critical in business related databases which is why traditional databases are more used in that area. It is also common that the data models of NoSQL databases are simpler which make them faster as well. (Leavitt, 2010.) NoSQL databases provide several different data models to choose from. NoSQL databases are also evolving fast, and some of them include some features that the relational databases don't such as potentiality to handle hardware failures. (Nayak et al., 2013.)

There are some known concerns about NoSQL databases. Manual query programming is needed with NoSQL databases as they don't work with SQL. Complex query programming can be time-consuming and difficult. Additional programming is needed if one wants to apply ACID into NoSQL databases. ACID is not a must but it provides reliability which is natively supported in relational databases. Consistency is also compromised because of the lack of ACID. Better performance and scalability are enabled but the lack of consistency might cause problems in certain application areas such as banking businesses. (Leavitt, 2010.) NoSQL databases are nowadays still seen as immature that are difficult to be maintained (Nayaka et al., 2013).

NoSQL databases are something that many organizations are not familiar with which leads to a situation that organizations don't feel comfortable to even considering choosing a NoSQL database technology over traditional database technologies. Open source NoSQL applications have also a lack of customer support and management tools. NoSQL databases will not probably replace relation databases. Instead, they will be used for types of projects that require scalability with unstructured data. (Leavitt, 2010.) The previously mentioned matters are collected into a comparative table in Table 5. However, it should be mentioned that not all attributes apply in every SQL and No-SQL databases; the comparison is done in more general level.

Table 5. SQL vs. NoSQL (Han et al., 2011; Leavitt, 2010; Nayak et al., 2013.)

	SQL	**No-SQL**
Faster data reading and writing		X
Support for mass storage		X
Easy to expand		X
Lower costs		X
ACID support	X	
More reliable	X	
Better performance		X
Better precision	X	
Better consistency	X	
Better scalability		X
Simpler data models		X
Ability to handle HW failures		X
No need for manual query programming	X	
Mature - easier to maintain	X	
Better customer support	X	
Better management tools	X	
Familiarity among organizations	X	

5.2 Different NoSQL databases

There are many NoSQL databases nowadays. Before going deeper into the databases, it is important to understand a couple of terms that are common when talking about NoSQL databases. **The CAP theorem** is irrelevant to relational databases; it is more about distributed systems (Loukides, 2012). According to Bryant (2014) many non-relational databases, for example CouchDB and HBase, are distributed so it is more meaningful to go through CAP theorem when talking about non-relational databases. The CAP theorem, presented in Figure 20, means that there is not any distributed system that permits consistency, availability and partition-tolerance. One system can guarantee only two of the previous three attributes at once. (Lourenço et al., 2015; Han et al., 2011; Stonebraker, 2010.) **Availability** means that the system should work even though a failure occurs, this can be done by taking advantage of a replica, for example. **Partition-tolerance** means that even if a network failure occurs that makes the nodes unable to communicate with each other, the processing could continue in subgroups normally. The idea of **consistency** is to make multisite transactions support the all-or-nothing feature. In other words it means that the states of replicas are equivalent. (Stonebraker, 2010.) **Shared-nothing** means that every node in the system has its own memory and disk storage that are not shared with nodes of the system (DeWitt, Naughton & Schneider, 1991).

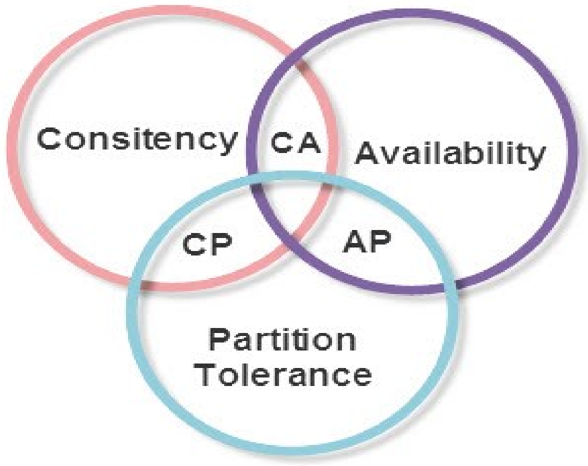


Figure 20. Cap theorem.

In the next sub-sections I will go through the best-known ones found in the literature: Aerospike, Cassandra, CouchDB, Couchbase, HBase, MongoDB, and Voltemort.

5.2.1 Aerospike

Aerospike is a shared-nothing open-source key-value database. It supports availability and partition-tolerance (AP). However, it has been said by the developers that Aerospike can provide high consistency by exchanging other attributes such as availability. In short, Aerospike is an in-memory database that comes, for example, with disk persistence, automatic data partitioning and synchronous replication. (Lourenco et al., 2015.) The official website of Aerospike promotes itself as a distributed NoSQL database that fits for web-scale applications of today’s needs thus providing strong consistency with no downtime (Aerospike, 2016).

5.2.2 Cassandra

Cassandra is a shared-nothing open-source column-store database. It supports availability and partition-tolerance (AP) and consistency and partition-tolerance (CP). Cassandra supports a language called CQL that is similar to SQL. Indexes, secondary indexes, and atomicity is supported in Cassandra. Consistency and latency are tunable by the developers. (Lourenco et al., 2015.) The official website of Cassandra promotes itself as a database with scalability and high availability alongside with good performance. The support of multiple datacenters is said to be one of the best. (Cassandra, 2016.)

5.2.3 CouchDB

CouchDB is an open-source document-oriented RAM-based database. It is written in Erlang, and its documents are written in JSON. CouchDB supports AP. (Lourenco et al., 2015.) The documents are accessed and the indexes queried with web browser using HTTP, and the documents are modified with JavaScript. CouchDB is advertised to work well with modern web and mobile applications. CouchDB is a master-slave database. (CouchDB, 2016.)

5.2.4 Couchbase

Couchbase is a RAM-based database. It is a combination of CouchDB, that is a document-oriented database, and Membase, that is key-value system. Couchbase can be used like key-value databases but it is considered as a document-oriented database. It supports CP. (Lourenco et al., 2015.) Couchbase is a distributed database that is advertised for its performance, scalability, and availability. The power of SQL and the usage of JSON enables for easy and fast application building. Couchbase is a shared-nothing database. (Couchbase, 2016.)

5.2.5 HBase

HBase is an open-source column-store database which is written in Java and developed by the Apache Software Foundation. The inspiration for HBase came from Google’s Big Table. Distributed storage is possible because The Hadoop Distributed File System can be used. ACID is supported to some level. All the data is a byte array in HBase. HBase supports CP. (Lourenco et al., 2015.) The official website of HBase advertises itself, for example, for linear and modular scalability, strictly consistent reads and writes, and JAVA API that is easy to use. HBase is a shared-nothing database. (HBase, 2016.)

5.2.6 MongoDB

MongoDB is an open-source document-oriented database which is written in C++. MongoDB supports horizontal scalability. Writes are processed through master node and reads can be processed through both master and slave nodes. Document manipulation is regarded as one of the strengths of MongoDB. MongoDB supports CP. (Lourenco et al., 2015.) The official website of MongoDB advertises itself providing high performance, high availability, and automatic scaling. MongoDB is a master-slave database. (MongoDB, 2016.)

5.2.7 Voldemort

Voldemort is an open-source RAM-based key-value store database. It is written in Java and it supports AP. Voldemort is pretty simple but also limited; there are only three commands used for the operations. (Lourenco et al., 2015.) The official website of Voldemort advertises itself as a distributed system that, for example, allows the data to be automatically replicated over several servers, its server failures are handled transparently, and all of its nodes are independent (Project Voldemort, 2016).

5.3 Choosing the right NoSQL database

When choosing a NoSQL database there are many attributes to be thought of such as availability and consistency (Nayak et al., 2013). Han et al. (2011) wrote that properties such as query API, CAP support, and data persistence must be thought of as well. Even though NoSQL databases have been studied, it is still somewhat difficult to see which purposes each database fits well for. It is clear that there are a lot of differences between different NoSQL databases. More use-case studies must be done in order to understand the capabilities of NoSQL databases better. (Lourenco et al., 2015.)

The data presented Table 6 is gathered from a table by Lourenco et al. (2015). They gathered together a table where availability, consistency, durability, maintainability, read-performance, recovery time, reliability, robustness, scalability, stabilization time, and write-performance were evaluated for each previously presented NoSQL database. Explanations for the evaluations are the following: (2=great), (1=good), (0=average), (-1=mediocre), (-2=bad), and (?=unknown). The database model is presented as well the following way (KV=Key-value store), (COL=Column-oriented store) and (DOC=Document-based store). Some of the quality attributes are explained earlier in this thesis and some are explained below. The data presented in the table was gathered by Lourenco et al. (2015) by surveying the database literature that was available at the time. The table can offer hints for situations when one is thinking whether to choose a NoSQL database and which NoSQL database fits for one's needs well.

Table 6. Quality attributes of different NoSQL databases evaluated (Lourenco et al., 2015) and the database model presented.

	Aerospike	**Cassandra**	**Couchbase**	**CouchDB**	**HBase**	**MongoDB**	**Voldemort**
Availability	2	2	2	2	-1	-1	2
Consistency	2	2	1	1	0	2	1
Durability	-1	1	1	-1	1	1	1
Maintainability	1	0	1	1	-1	0	-1
Read-performance	1	-1	2	0	-1	2	1
Recovery time	2	-2	1	?	?	2	?
Reliability	-1	1	-1	1	1	2	?
Robustness	1	1	0	0	-2	0	?
Scalability	2	2	2	-1	2	-1	1
Stabilization time	-2	1	1	?	?	-2	?
Write-performance	1	2	1	-1	1	-1	2
Database model	KV	COL	DOC	DOC	COL	DOC	KV
License	Open source	Open source	Open source	Open source	Open source	Open source	Open source

Availability is a quality attribute that tells us how well the system is available. In other words it tells us how much of the total time the system is working as intended. **Consistency** is a quality attribute that ensures that the same data is seen by every node within the database system simultaneously. **Durability** is a quality attribute that ensures that the data will stay committed and valid after a transaction has been done.

Maintainability is a quality attribute that tells us how easily a database can be maintained. Maintaining includes actions such as upgrading, repairing, debugging, and meeting new requirements. **Write-performance** is a quality attribute that tells us how well a database acts when executing writes, also known as inserts**. Read-performance** is a quality attribute that tells us how well a database acts when executing reads. **Stabilization time** is a quality attribute that tells us how long it takes for the system to stabilize when nodes join the cluster again after failure. **Recovery time** is a quality attribute that tells us how long it takes for the system to recover after a failure. **Reliability** is a quality attribute that tells us how long a system can operate without facing failures. **Robustness** is a quality attribute that tells us how well a database is able to deal with errors during execution. **Scalability** is a quality attribute that tells us how well a system is able to deal with increasing workload. (Lourenco et al., 2015.)

Table 7. Popularity and free online support for different NoSQL databases.

	Aero spike	**Cassandra**	**Couchbase**	**CouchDB**	**HBase**	**MongoDB**	**Volde mort**
Page loads of official websites	510 000	1 130 000	3 170 000	450 000	1 140 000	21 550 000	115 000
Wikipedia page loads	40 864	259 897	50 716	70 084	90 243	472 637	17 077
Stack Overflow	698	23 960	4793	12 897	11 364	61 456	304

The popularity and online support found for different NoSQL databases are presented in Table 7. The data for the attribute "Page loads of official websites" was gathered using a tool named SimiliarWeb found in https://www.similarweb.com/. The number of total visits for the official websites of the NoSQL technologies was gathered from October 2015 – March 2016 period. The final result is rounded. It should be mentioned that people visit websites that they do not intend to visit. It is common among websites with names that have several meanings such as Voldemort and Cassandra in this case.

The data for the attribute "Wikipedia page loads" was gathered using Pageviews Analysis tool from https://tools.wmflabs.org/pageviews/. The number of page loads for the Wikipedia pages of the NoSQL technologies in English was gathered from August 1st 2015 to April 14th 2016.

The data for the attribute "Stack Overflow" was gathered from the official website of Stack Overflow from http://stackoverflow.com/. Each previously mentioned NoSQL database was used as an entry to see how many questions were found related to each technology. The amount of questions found indicates how much free support is easily available online.

When looking at the page loads of the official websites the ranking is the following starting from the website with the most loads. 1. MongoDB, 2. Couchbase, 3. HBase, 4. Cassandra, 5. Aerospike, 6. CouchDB, and 7. Voldemort. When looking at the Wikipedia page loads of the databases the ranking is the following starting from the websites with the most loads. 1. MongoDB, 2. Cassandra, 3. HBase, 4. CouchDB, 5. Couchbase, 6. Aerospike, and 7. Voldemort. When looking at the questions found in Stack Overflow the ranking is the following starting from the technology with the highest amount of questions. 1. MongoDB, 2. Cassandra, 3. CouchDB, 4. HBase, 5. Couchbase, 6.

Aerospike, and 7. Voldemort. The same results are presented in Table 8 to make the comparison more clear.

Table 8. Popularity and free online support for different NoSQL databases in order.

Page loads of official websites	Wikipedia page loads	Stack overflow
1. MongoDB	1. MongoDB	1. MongoDB
2. Couchbase	2. Cassandra	2. Cassandra
3. HBase	3. HBase	3. CouchDB
4. Cassandra	4. CouchDB	4. HBase
5. Aerospike	5. Couchbase	5. Couchbase
6. CouchDB	6. Aerospike	6. Aerospike
7. Voldemort	7. Voldemort	7. Voldemort

From the results it can be deduced that the most popular NoSQL technology with the best free online support is MongoDB and the second most popular is Cassandra. The least popular NoSQL technology with the least free online support is Voldemort and the second least popular is Aerospike.

5.4 Meeting the requirements

This section explains how the previously mentioned non-relational databases, Aerospike, Cassandra, CouchDB, Couchbase, HBase, MongoDB, and Voldemort meet the requirements given by the case company. It is also explained why or why not each database fit for a candidate in this thesis. The requirements and requests were explained in section 3. Requirements of the database system. A more detailed selection process is explained in section 8. Choosing the database system.

Aerospike. Even though Aerospike is a key-value database that is free and it can store structured data which is a requirement and it is known for being fast, it must be left out of considerations because it is an in-memory database. The database system in this case must store the data permanently. Popularity and free online support of Aerospike seen in Table 7 and Table 8 are kind of low as well which approve the previously mentioned decision as well. In short: Aerospike is not an eligible choice for a database technology in the case of this thesis.

Cassandra is a free column-store database which means that it can store the type of data that is required in this case. Cassandra is told to have a good support for indexing and high performance as told in section 5.2.2. Cassandra. However, in Table 6 it turns out that the performance is only good when considering write-performance. Read-performance, which is valued more by the case company, is not that good after all. Popularity and free online support for Cassandra can be considered as pretty high which can be seen in Table 7 and Table 8. In short: Cassandra is an eligible choice for a database technology in the case of this thesis.

CouchDB is a free document-oriented database that is ram-based. CouchDB is not a suitable choice for the following reasons. Document-oriented databases should not be used if relations and normalization are needed as written in section 5. Non-relational databases. In the case of this thesis relations and normalization will be needed as the data is relationally structured and queries need to be fast. Data also must be permanently stored which means that a ram-based is not on option. Also, when looking at Table 6 it can be seen that read-performance, write-performance and maintainability of CouchDB are pretty low. In short: CouchDB is not an eligible choice for a database technology in the case of this thesis.

Couchbase is a free document-oriented database that is ram-based. Couchbase is not a suitable choice for the same reasons as CouchDB; Document-oriented databases should not be used if relations and normalization are needed as written in section 5. Non-relational databases. In the case of this thesis relations and normalization will be needed as the data is relationally structured and queries need to be fast. Data also must be permanently stored which means that a ram-based is not on option. Even though Couchbase offers a good read-performance, write-performance and maintainability as seen in Table 6, the previously mentioned facts are more influence the decision more. In short: Couchbase is not an eligible choice for a database technology in the case of this thesis.

HBase is a free column-store database. HBase could be considered as a suitable choice for the following reasons. Column-store databases can be used as row-based databases but column-store databases are known for being faster in reading as mentioned in section 5. Non-relational databases. When looking at Table 6 it can be seen that HBase offers mediocre maintainability and read-performance, and write performance is good. These quality attributes can be considered as acceptable in the case of this thesis. In short: HBase is an eligible choice for a database technology in the case of this thesis.

MongoDB is a free document-oriented database. MongoDB offers a great read-performance and average maintainability as seen in Table 6. The popularity and free online support of MongoDB are also great as seen in Table 8. These attributes advocate the fact that MongoDB could be a great choice for a database technology in the case of this thesis. However, as mentioned earlier in this section and in section 5. Non-relational databases, document-oriented databases should not be used when relations and normalization are needed. These facts make the decision contradictory, but as the data in the case of this thesis can be considered as relational, MongoDB must be left out of being a candidate for a database technology in this thesis. In short: MongoDB is not an eligible choice for a database technology in the case of this thesis.

Voldemort is a free key-value store ram-based database. The fact that it is a key-value store it could be an eligible choice but because of the fact that it is a ram-based database, it must be left out of considerations. The database system in this case must store the data permanently. The read-performance is good and write-performance is great as seen in Table 6. These facts advocate that it would be a good choice in the case of this thesis. However, the fact of being a ram-based and having a very low free online support and popularity as seen in Table 8 mean that Voldemort must be left out of being a candidate. In short: Voldemort is not an eligible choice for a database technology in the case of this thesis.

After the analysis of meeting the requirements of the previous 7 non-relational databases, it turned out that only 2 of them, Cassandra and HBase, could be considered as candidates for the database technology to be chosen in this thesis. Cassandra, HBase, and MySQL, which were chosen in section 4.11 Meeting the requirements, will be compared among themselves and the database that meets the requirements the best will be found out in section 8. Choosing the database.

6. Databases and data analytics – RQ2 & RQ6

In this section knowledge found about databases and their relation to data analytics is gone through. This section of the thesis will answer fully or partially the following research questions: "*RQ2: What database management systems do currently exist?*" and "*RQ6: How do different database management systems match the needs of a software testing team when the stored data is heavily filtered?*". Terms *Small Data* and *Big Data* are essential to understand when choosing a database for analytics as different databases are more suitable for different kind of data. Those terms, databases and data analytics are discussed in the following text. In the end of this section I explain what kind of database fits best to meet the requirements of the database given by the case company when considering databases and data analytics.

NoSQL databases have started to tackle the problem of handling data collections which have grown so large that can't be well managed with relational database management systems. A term *Big Data* is used for this kind of data collection. NoSQL databases can analytically process large scale datasets, and NoSQL supports exploratory and predictive analytics. (Moniruzzaman & Hossain, 2013.) There is not just one definition for big data. However, data storage and data analysis are mainly connected to big data. (Ward & Barker, 2013.) Moniruzzaman and Hossain (2013) described big data as data that is growing faster than even before and it is in many different forms, structured, unstructured and sometimes hybrid as well. Chen, Mao and Liu (2014) described big data as data that consists of masses of unstructured data which require more real-time analysis. Ward and Barker (2013 pp. 2) summed up the definition of big data the following way: "Big data is a term describing the storage and analysis of large and or complex data sets using a series of techniques including, but not limited to: NoSQL, MapReduce and machine learning.".

Small data is data that is accessible, informative and actionable in its volume and format. The main idea of small data is that businesses can benefit from the small data without using complex systems that are needed to in big data analytics. (Banafa, 2014.) The following Table 9 is slightly modified from Banafa (2014). Table 10 is a table by Levin (2015) where he listed criteria of a database and whether relational or non-relational database fits better for that kind of analytical needs. The next table, Table 11, is a table by Levin (2015) as well that shows good database fits for an analytical database depending on the data size.

Table 9. Big data vs. small data (Banafa, 2014).

	Big Data	**Small Data**
Data Sources	Data generated outside the enterprise from nontraditional data sources such as social media and videos.	Traditional enterprise data such as web transactions and financial data.
Volume	Terabytes Petabytes Exabytes Zettabytes	Gigabytes Terabytes
Velocity	Often real-time Immediate response needed	Batch or near real-time Immediate response not always needed
Variety	Structured Unstructured Multi-structured	Structured Unstructured
Value	Complex, advanced and predictive	BI, analysis and reporting

Table 10. Choosing a database for analytics (Levin, 2015).

Criteria	**Relational**	**Non-relational**
Type of data	Structured	Unstructured
Would fit in massive	Excel sheet	Word document
The schema	Stays the same	Changes often
Works well with	User data, inventory	Emails, photos, videos
For analysis like	User paths, funnel analysis	Text mining, language processing
Can query with	SQL	MapReduce, Python

Table 11. Analytical database fits depending on the data size. (Levin, 2015).

Data size	**Good fit**
< 1TB	PostgreSQL, MySQL
2TB-64TB	Amazon Aurora
64TB-2PB	Amazon Redshift, Google BigQuery
"All of the data"	Hadoop

When looking at Table 9 it can be seen that the data stored into the database in this thesis will not be data that is considered big data, it could rather be considered small data because data sources, volume, velocity, variety and value match better to small data. When looking at the criteria from Table 10 it can be seen that a relational database is a better choice for a database for analytics in this case. There will be, for example, no need to store videos and the schema will not change often. When looking at Table 11, PostgreSQL and MySQL can be considered as good fits because the data size of the database will be less than one terabyte. However, as compared in section 4.11 Meeting the requirements, MySQL can be considered as a better fit than PostgreSQL in this study. The other databases found in Table 11 were not compared earlier or considered as candidates on this thesis because there weren’t comparisons and earlier research found about them. To sum this section up it can be said that MySQL can be considered as a good fit for this thesis when looking at data analytics point of view as well.

7. Distributed databases – RQ2 & RQ6

In this section previous knowledge found about distributed databases in research and literature is gone through. This section of the thesis will answer fully or partially the following research questions: "*RQ2: What database management systems do currently exist?*" and "*RQ6: How do different database management systems match the needs of a software testing team when the stored data is heavily filtered?*". In the end of this section it can deduced whether a distributed database is relevant in this thesis or not. This deduction is based on findings found from the literature.

A distributed database is defined as a collection of multiple databases that are logically interrelated and distributed over a computer network (Özsu & Valduriez, 2011; Mahajan, 2012). Sharma and Bhardwaj (2014) defined a distributed database as kind of virtual database that has component parts physically stored in several number of databases at different locations. However, Özsu and Valduriez (2011) mentioned that the locations of the databases don't necessarily have to be geographically far apart; they can even be in the same room. A distributed DBMS is defined as the software which is used to manage the distributed database and it also makes the distribution clear to the users. The term, distributed database system (DDBS), is sometimes used when talking about the combination of a distributed database and a distributed database management system. (Özsu & Valduriez, 2011.)

A DDBS technology is a combination of database system and computer network technologies. These two technologies are very different from each other but together they make a very powerful combination. A database is intended to provide centralized and controlled access to the desired data, and computers networks work without any focus on centralization. However, it should be noted that the main purpose of the database technology lies in integration. The goal of the distributed database technology is to attain integration but no centralization. (Özsu & Valduriez, 2011.)

The environment of a distributed database system is presented in Figure 21 modified from Özsu and Valduriez (2011). From the figure it can be seen that there are three sites in the networks and the data is distributed into two of them; into SITE 1 and SITE 3. If a database was centralized on a network, the database would be managed by one site only which means that everything would be routed through that site. In DDBS the data can be delivered from different sites depending on which site the query by the user is pointed to.

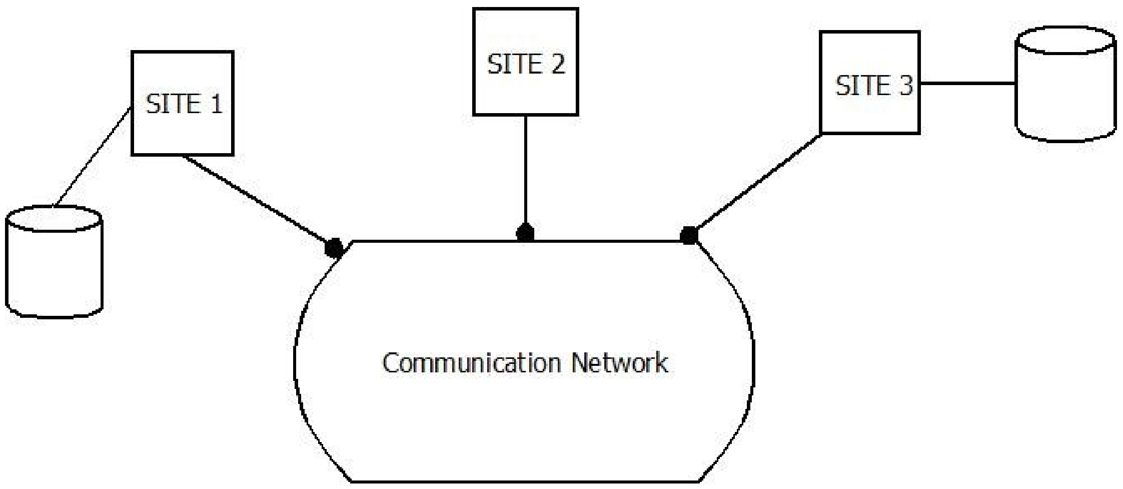


Figure 21. A distributed database system environment with three sites modified from Özsu and Valduriez (2011).

Özsu and Valduriez (2011) presented four fundamental advantages of the DDBS technology. They were 1. Transparent management of distributed and replicated data, 2. Reliable access to data through distributed transactions, 3. Improved performance, and 4. Easier systems expansion. A closer look to those promises is taken next.

A fully transparent system would be ideal as it provides high level of support for developing sophisticated applications. The implementation details are not seen to the users in transparent systems. A DDBS provides more reliability because of the replicated components. A failure on one site doesn't break the whole system; the users are still able to access the other sites of the DDBS. The use of data localization makes the performance of a DDBS better; the data can be stored close to its place of use which, for example, reduces the remote access delays that are common in wide area networks. A DDBS is also easier to expand as it can be done by increasing the processing and storage power of the network. The cost of expanding a DDBS is lower as it is not so costly to purchase several computers that make up equivalent power of one big main system. (Özsu & Valduriez, 2011.)

The problems in distributed database systems are similar to the ones there are in conventional database systems but they are a bit more complex as the nature of DDBS is more complex as well compared to conventional database systems. Some of the well-known problems are related to security, distribution management and increased costs because of replicated resources. Data items might be duplicated because the databases are located in more than one sites. The distributed system must be able to make sure that even though communication failures happen between some sites, the actions must take place as soon as possible so that every site gets updated. It should be also noted that it takes more effort to synchronize transactions in a distributed system because of the several possible sites. (Özsu & Valduriez, 2011.)

Mesmoudi and Hacid (2014) studied the performance of traditional database management systems compared to distributed database management systems. The traditional DBMSs were MySQL, PostgreSQL and DBMS-X. The DDBSs were HadoopDB and Hive. The goal of the study was to find out the abilities of the previously mentioned systems to support declarative queries. In the results it turned out that traditional DBMSs outperformed distributed systems when the performed queries consisted of a few tuples only.

Distribution is sometimes needed because it seems to fit into today’s world better; enterprises are distributed so databases need distribution as well. Distribution is also needed to deal with large-scale problems that lie in data management today. Distributed databases should be viewed as tools that make distributed processing better. Distributed databases have become a helping hand for developing software that is distributed as well. (Özsu & Valduriez, 2011.) In other words, in my opinion, it could be summed up that distributed databases have been created to face the challenges the evolving world and technology have brought along.

It seems that the database I will construct will not need any distribution because of the following matters. The database will not need to store data that is, for example, distributed among several organizations. Distribution would also cause extra costs as mentioned by Özsu and Vaduriez (2011) which is not allowed in this thesis. The results from previously mentioned study by Mesmoudi and Hacid (2014) also advocated that a traditional DBMS would be a better choice for this thesis as the queries will not probably need to consist of several tuples.

8. Choosing the database – RQ3 & RQ6

In this section the final selection process of the database is explained. This section of the thesis will answer fully or partially the following research questions: "*RQ3: What are the differences between relational and non-relational database management systems in terms of usage and performance?*" and "*RQ6: How do different database management systems match the needs of a software testing team when the stored data is heavily filtered?*". The databases that met the requirements in sections 4.11 and 5.4 are compared. The requirements of the case company for the database system were explained in section 2. Requirements of the database system. This final comparison is made between one relational database, MySQL, and two non-relational databases, Cassandra and HBase. Two of the previously mentioned databases are compared at once to come up with a more suitable choice. The comparison is heavily based on the maintainability and performance as they are the most important features requested by the case company. Once every database has been compared among themselves, the final decision can be made and the choice can be represented to the representative of the case company. The same selection process is also explained to the representative of the case company, and the DBMS will be finally chosen once the representative has agreed with my decision.

Table 12. Popularity and free online support for MySQL, Cassandra and HBase.

	MySQL	**Cassandra**	**HBase**
Page loads of official websites	65 150 000	1 130 000	1 140 000
Wikipedia page loads	603 486	259 897	90 243
Stack Overflow	391 774	23 960	11 364

8.1 MySQL vs. Cassandra

Maintainability. The data for Table 12 is gathered from Table 3 and Table 7. When considering maintainability it can be clearly seen from Table 12 that MySQL is more popular and offers more free online support than Cassandra. NoSQL databases are more immature as mentioned by Leavitt (2010) which makes the maintainability more difficult as well. One request from the case company was the support for SQL. Cassandra doesn't support SQL but it supports its own query language QCL that is similar to SQL (Lourenco et al., 2015). It can be summed up from the previous that when looking at maintainability point of view MySQL can be considered as a more suitable choice because of its better free online support and case company representative's previous experience with SQL.

Performance. NoSQL databases are known for offering a good performance (Han et al.; Leavitt, 2010; Nayak et al., 2013). However, SQL databases offer a good performance as well when the amount of data is not growing rapidly (McNulty, 2014) and when the table size hangs around in the size of hundreds of gigabytes (Dulin, 2015). When considering the amount of data, which is not very high for the subject of this thesis, SQL databases offer a good performance as well as NoSQL database. MySQL supports indexing (Conrad, 2004) and Cassandra supports indexing as well (Lourenco et al., 2015).

According to Giacomo (2005), as mentioned before, indexing is a way to speed up queries. It can be summed up from the previous that when looking at performance point of view MySQL and Cassandra perform evenly.

MySQL can be declared as the winner of this comparison because of its better maintainability. Cassandra would become a better option if the amount of data that needs be stored was a lot higher. However, in the case of this thesis it is not needed.

8.2 MySQL vs. HBase

Maintainability. When considering maintainability it can be clearly seen from Table 12 that MySQL is more popular and offers more free online support than HBase. HBase doesn't support SQL (IBM, 2006a) which was a request from the case company. HBase has its own shell with data manipulation language and data definition language which are used to communicate with the database (Tutorialspoint, 2016). The same NoSQL immaturity (Leavitt, 2010) Cassandra has applies to HBase as they both are NoSQL databases. It can be summed up from the previous that when looking at maintainability point of view MySQL can be considered as a more suitable choice because of its better free online support and case company representative's previous experience with SQL.

Performance. HBase supports indexing as well (IBM, 2016b). The same performance attributes that Cassandra has apply to HBase as well because of being a NoSQL database which means that HBase has good performance. As mentioned before, SQL databases offer a good performance as well when the amount of data is not growing rapidly (McNulty, 2014) and when the table size hangs around in the size of hundreds of gigabytes (Dulin, 2015). It can be summed up from the previous that when looking at performance point of view MySQL and HBase perform evenly in this case because the amount of data is not high.

MySQL can be declared as the winner of this comparison because of its better maintainability. HBase would become a better option if the amount of data that needs be stored was a lot higher. However, in the case of this thesis it is not needed.

8.3 Cassandra vs. HBase

Maintainability. When considering maintainability it can be clearly seen from Table 12 that Cassandra is more popular and offers more free online support than HBase. HBase has more visits of the official website but Cassandra has more Wikipedia page loads and Stack Overflow search results. Neither of the databases supports SQL but Cassandra supports a query language that is similar to SQL (Lourenco et al., 2015). It can be summed up from the previous that when looking at maintainability point of view Cassandra can be considered as a more suitable choice because of its better free online support and SQL-like query language.

Performance. Because so far I have based assumptions of the performance of Cassandra and HBase based on more general level, I felt that there was a need to find a more precise comparison between the two. End Point (2015) conducted a benchmark of top NoSQL databases Cassandra, Couchbase, HBase and MongoDB. In the same study, which consisted of testing read and write operations, it turned out that Cassandra outperformed all other NoSQL databases in the terms of throughput and latency. This means that from performance point of view Cassandra can be considered as a more suitable choice because of its better performance.

Cassandra can be declared as the winner of this comparison because of its better maintainability and performance.

8.4 Conclusion

From the previous sub-sections 8.1, 8.2, and 8.3 it can be deduced that MySQL is the best database technology in the case of this thesis. From NoSQL point of view Cassandra would be the best choice. Also my previous deductions from sections 6. Databases and data analytics and 7. Distributed databases advocate my choice of MySQL.

After presenting my study of different databases to the representative of the case company, I proposed MySQL to be the database technology used in this thesis. The representative agreed with the choice.

9. Designing, implementing and evaluating the database – RQ1 & RQ6

In this section the design, implementation and evaluation phases of the database are explained. This section of the thesis will answer fully or partially the following research questions: "*RQ1: What database design principles and techniques should be followed to make a database well-performing and easily maintainable?*" and "*RQ6: How do different database management systems match the needs of a software testing team when the stored data is heavily filtered?*". It should be noted that some of the data included in the database are confidential within the case company and me. For that reason some data is left out out from the presentation of the design and queries or the data is censored, for example, by naming data and columns differently from the original design. This might affect the readability and understandability on some level.

Analysis of the need for database is critical no matter the complexity or size of the database. One should determine the use of the database. One way to do is by constructing use cases or user stories. Use cases are made in free-format text and the main idea of them is to describe the actions executed by the end-user. At least a few use cases must be thought of in the design phase. (Churcher, 2007.)

A data model should also be created. A data model is used to connect the basic purpose of the database and the design of the relevant tables. Data modeling is used to understand the problem, which must be solved, better. A database can be considered as unsuccessful if no proper data model is made. Diagrams, such as UML notations, can be used to represent the data models. The idea of diagrams is to present understandable information without using unnecessary text. (Churcher, 2007.)

A data model that supports the use cases reveals how broad the problem area is along with the details it involves. This is the base for designing an application before making the implementation one's business. Moving from a data model to a design phase shouldn't require much effort. The implementation phase requires more effort, on the other hand. One should not underestimate what it takes to implement an operative database. A good data model ensures that the database will be good as well if it is implemented precisely. It is better to realize the problem within the design phase than after hours of work that has been spent on the implementation only to realize that the solution is not working as intended. It is also important to think about the needs of the future; short-term planning might lead to a satisfactory outcome for a while but it might lead to problems in the future. (Churcher, 2007.)

9.1 Initial problem statement with use cases

Using use cases with UML is one way to describe the problems. Use cases are used to represent how actors, the users, use the system. The tasks the actors do are most likely, for example, entering data or extracting data when dealing with a database system. (Churcher, 2007.)

The UML notation for use cases consists of a stick figure that represents the user and ovals that represent the tasks executed by the user. One should also write a little documentation about each of the use cases to get a more detailed description of each use case. (Churcher, 2007.)

Four use cases are presented in Figure 22. A little documentation about each of the uses cases is presented next as well.

Use case 1: Maintain software information for each DUT. The user wants to enter and retrieve information for each software used on each DUT. This includes data such as DUT name, SW, HW, uptime, active alarms, inactive alarms and start times.

Use case 2: Retrieve faults found on each DUT. The user wants to find out DUT specific faults found for each software. This includes data such as the software version, fault ID, fault name, number of faults, and the time/times the faults were found.

Use case 3: Retrieve the number of tests run on each SW. The user wants to find out the number of tests that have been run on each SW. This includes data such as the software version, the DUT name the tests have been run on, the number of tests run and the name of the test cases that have been run.

Use case 4: Create a new table to store new data. The user wants to create a new table into the database to store new data. The database needs maintenance as time goes by and new data must be stored.

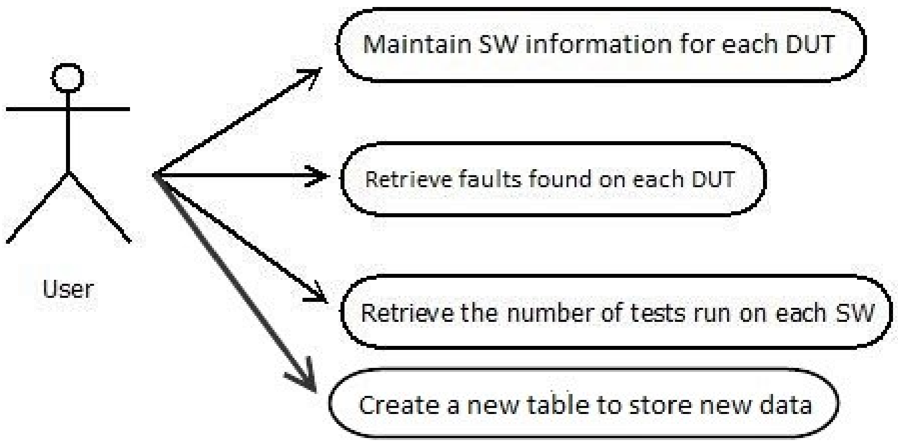

Maintain SW information for each DUT
Retrieve faults found on each DUT
User
Retrieve the number of tests run on each SW
Create a new table to store new data

Figure 22. Use cases.

9.2 Moving to analyzing and data modeling

It is time to move to modeling phase after understanding the problem area a little better. One useful way to do this is to make data models which should represent how the data interacts among themselves. There are class diagrams in UML that are a suitable choice for doing this. The class diagrams can be initially done with paper and pen but computer software can be used as well. (Churcher, 2007.) The following example of a class and its objects, which are presented in Figure 23, is inspired from Churcher (2007, pp. 16) but modified to match the case of this thesis. A class consists of names for each attribute and the objects of the class have values for the attributes (Churcher, 2007). In this case the class is named *fault* and attributes *fault_id*, *actCritical*, *inactCritical*, *software_id* and *configuration_id*. The class has three objects with values for the attributes.

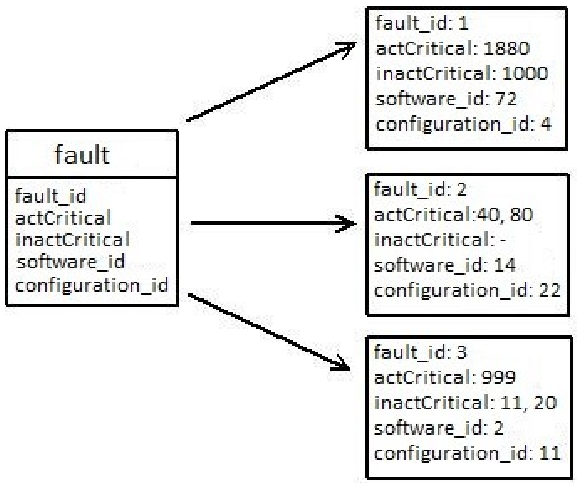


Figure 23. A class diagram with three objects.

There can be relationships between the objects of classes. When dealing with databases, every time a new class is created a new table is created as well. The relationships are presented in the design of the tables. Relationships should be read both ways: from left to right and from right to left. The following example of relationships, seen in Figure 24, presented by an UML class diagram is from Churcher (2007, pp. 18).

When reading from left to right the attention should be paid to "1..*". It means that an object of Class A is associated with at least one or several objects of Class B. When reading from right to left the attention should be paid to "0..1". It means that an object of Class B is associated with zero or maximum of one object of Class B. (Churcher, 2007.)

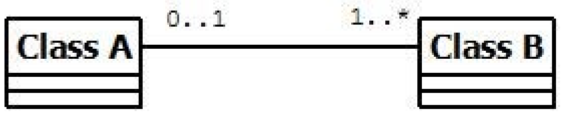


Figure 24. An example of relationships between classes by Churcher (2007, pp. 18).

For the sake of this study MySQL Workbench was used to design the database. MySQL Workbench is a design tool that provides possibilities for data modelling. The user is able to design, model, generate, and manage databases with MySQL Workbench. Data modelling is done with ER models. (MySQL, 2016b.) Both, UML class diagrams and UML ER models, can be used to represent a database and its data elements and relationships even though there are some differences between the two models (Al-Shamailh, 2015). The information presented in Figures 23 and 24 can be applied in the following ER models as well. It is possible to make models with ER in MySQL Workbench (Letkowski, 2015) so the data model will be represented using an ER model.

9.2.1 The design: tables, attributes and relations

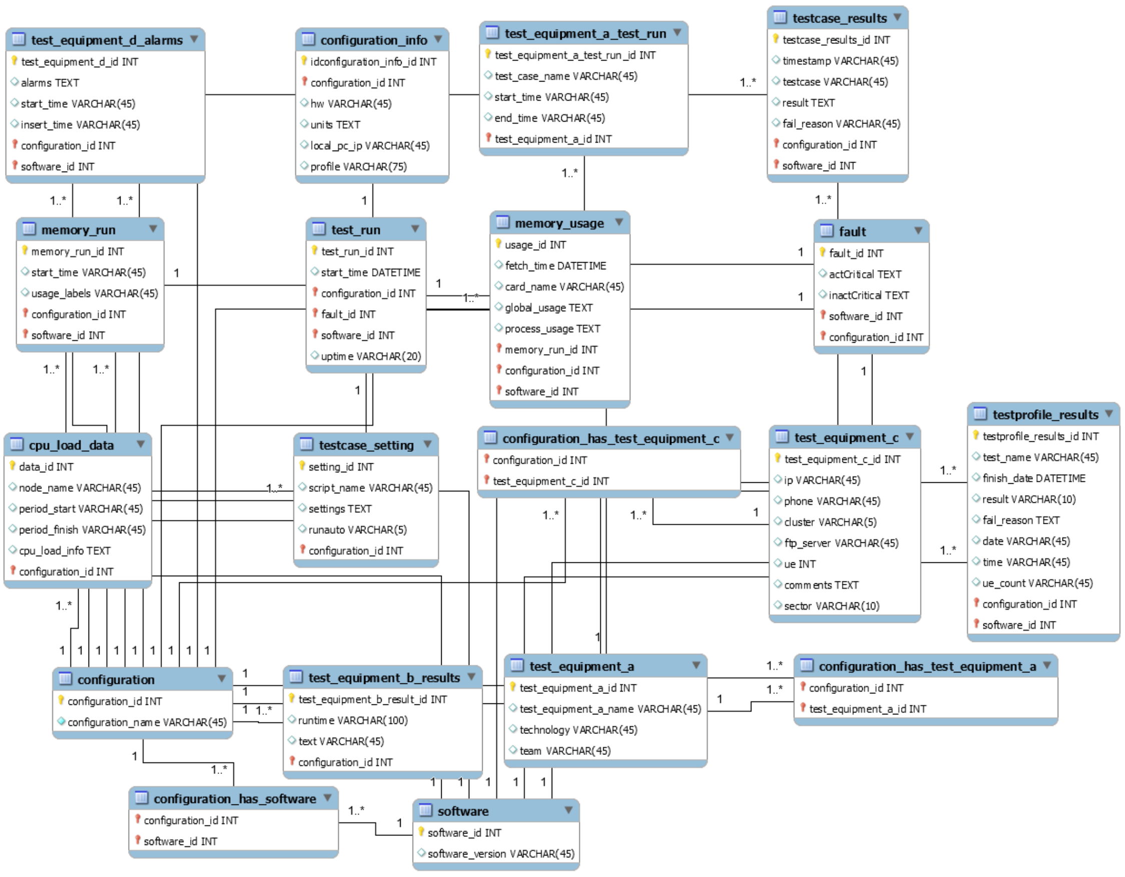


Figure 25. The final design of the new database.

The final design presented in Figure 25 of the database is rather straightforward. The main tables are *configuration* and *software*. All the other tables are related to either one of them or to both of them. Indexes were created on all foreign keys because the tables are related to each other via foreign keys, and the foreign keys are highly used in queries that are executed to insert and retrieve data from the database. The reason for doing that was because as mentioned in section 4.7 Indexes, one should set indexes on attributes that the wanted speeded up queries are related to (Ramakrishan & Gehrke, 2000).

It took three iteration cycles to end up with the final form of design. Each design was discussed with a representative from the case company and tested with fake data that was similar to real data. This build and evaluate –iteration phase in an obligatory part of design science research framework.

The main problem in the first version of the new design lied in the fact that the design didn’t allow tables to be related to both main tables *configuration* and *software* at the same time. This was solved by making multiple relations starting from one table. Now, for example, the table *memory_run* has two foreign keys that are related to tables *configuration* and *software*.

The main problem in the second version of the design showed up when testing the design with a real life scenario. It turned out that it was not possible to identify a difference between the test runs if the same software was tested on the same configuration more than once. Table *test_run* was added to the design to solve the problem.

During the final iteration cycle small changes were made. Some attribute names were modified, new attributes were added and data types were modified.

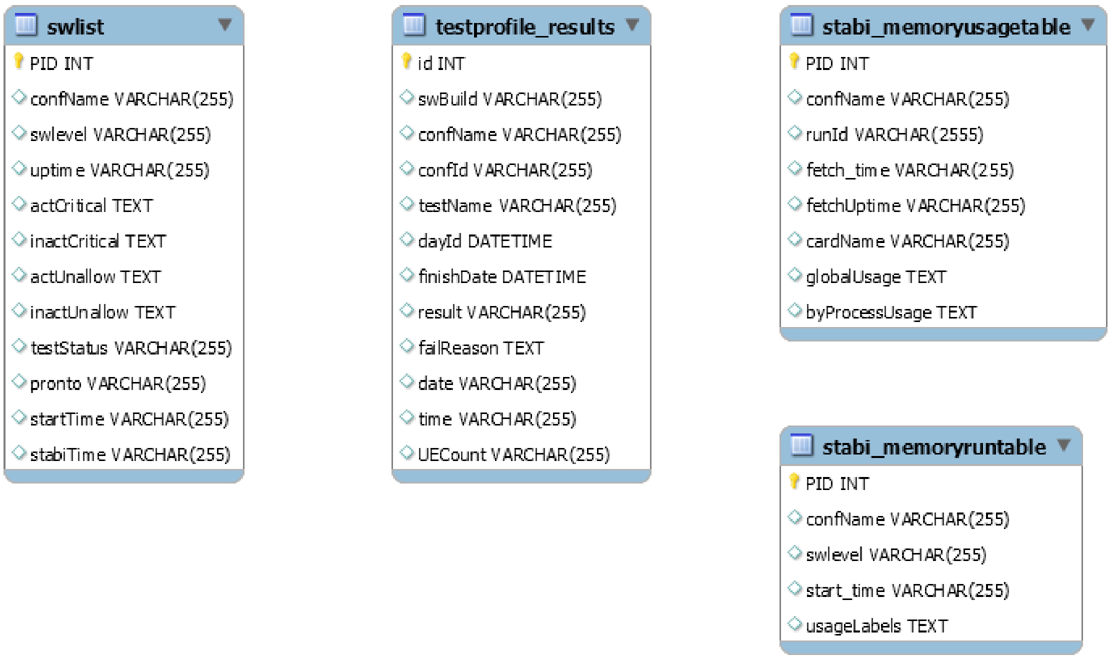


Figure 26. A part of the design of the old database.

One part of the design of the old database is presented in Figure 26. As it can be seen, there are just discrete tables and no relations had been created between them. This had led to a situation where duplicates were inevitable. All four tables in Figure 26 have attributes *confName* and three tables have attributes *swlevel* or *swBuild* which are used to store same data. With the new database, in Figure 25, this was solved by using relations and foreign keys. Foreign key IDs are pointed to their original source of data by using relations. In the old database there were no indexes were set either.

The data in the new database is separated into several tables because of normalization. For example, the data in the old database in Figure 26 in table *swlist* is distributed in tables *configuration*, *configuration_has_software*, *software*, *fault* and *test_run* in the new database seen in Figure 25. Those tables are related to each other via foreign keys. Indexes have also been set to the foreign keys.

9.2.2 Creating the schema of the database

MySQL Workbench has a feature called “Forward engineer” that allows the user to make a SQL script that creates the database that has been designed with the ER diagram (Letkowski, 2015). The generated SQL script is presented with an explanation for each generated table for the sake of readability. The commands themselves are quite self-explanatory and easily understandable with the information found earlier in this thesis in section 4. Relational databases. If there is abnormality along the commands, the aberrations are explained. The SQL scripts are presented in Appendix A.

9.3 Evaluating the database

A black box testing was conducted for the database. Black box testing is done to test the behavior of the system while focusing on the functional requirements (Pressman, 2010).

The testing was done to see whether the database can store the required data and whether the required data can retrieved.

The query times of the old and the new databases were compared in order to see the effectiveness of the new database because the fast query times were one of the two main requirements along with good maintainability given by the case company. Both databases stored the exact same data, both databases were on the same server and the same computer was used to execute the queries on both databases. The purpose of the queries that were executed was to fetch the same data from both databases. To sum it up; both databases, the old one and the new ones, were used in real-life situation with the same real data in order to see whether the new database performed better. The chosen queries were ones that are executed often daily. The results of the query times are presented in Table 13.

Table 13. Query times of the old and new databases.

	Old database	New database
Query #1 time	0.1808s	0.0023s
Query #2 time	19.6055s	0.0008s
Query #3 time	28.8947s	0.0016s
Query #4 time	0.1746s	0.0221s
Query #5 time	0.0024s	0.0005s

9.3.1 The queries

Query #1 explanation

The purpose of the query is to retrieve data from table *testprofile_results* for a specific configuration and software. The query was chosen for comparison because it is a query that is executed often to retrieve data, such as test names and finish times, about the tests that have been run on a specific configuration with specific software.

With the old database it took 0.1808 seconds to retrieve the data while with the new database it took 0.0023 seconds to retrieve the same data. The new query is as follows:

```
SELECT    test_name,    date,    time,    finish_date,    configuration_name,
software_version
FROM testprofile_results
INNER JOIN
Configuration                                                              ON
testprofile_results.configuration_id=configuration.configuration_id
INNER JOIN software ON
testprofile_results.software_id=software.software_id WHERE
configuration_name='X' AND software_version='X';
```

The old query in the old database to retrieve the same data was as follows:

```
SELECT test_name, date, time, finish_date, confName, swBuild
FROM testprofile_results
WHERE confName='X' and swBuild='X';
```

The old query was simpler than the new one but the old one turned out to be slower. The main differences between the new and old queries are the following. All the data was retrieved from one table with the old query; the size of the old table was greater than the new one which means that more rows of data had to be gone through to find the needed result. There were a lot of duplicates of *confName* and *swBuild* in table *testprofile_results* in the old database. The new database was normalized as instructed in section 4.8 Database normalization. The data is now separated in several tables instead of one. By doing this the sizes of the tables remain lower and no duplications of data is stored. Join-operations are used to combine data from several tables (Ramakrishnan & Gehrke, 2000). Attributes *configuration_name* and *software_version* can be retrieved from their own *configuration* and *software* tables using INNER JOINs. According to Ramakrishnan and Gehrke (2000) indexes should be built on the attributes that the wanted speeded up queries are related to. Indexes were set during the design phase on attributes that need to be looked up fast: *configuration_id* and *software_id* in this case. In conclusion, the new query performs better because of normalization, indexing and use of join-operations.

Query #2 explanation

The purpose of the query is to retrieve data from table *memory_usage* for a specific configuration, software and memory run. The query was chosen for comparison because it is a query that is executed often to retrieve memory usage data, such fetch times and memory usage values for cards, for a specific configuration and software. With the old database it took 19.6055 seconds to retrieve the data while with the new database it took 0.0008 seconds to retrieve the same data. The new query is as follows:

```
SET @memoryid=(SELECT memory_run_id FROM memory_run
INNER JOIN configuration
ON memory_run.configuration_id=configuration.configuration_id
INNER JOIN software ON memory_run.software_id = software.software_id
WHERE
configuration_name='X'
AND software_version='X');
SELECT * FROM memory_usage WHERE memory_run_id=@memoryid;
```

The old query in the old database to retrieve the same data was as follows:

```
SET @memoryid=(SELECT PID FROM stabi_memoryruntable
WHERE confName='X' AND swlevel='X');
SELECT * FROM stabi_memoryusagetable WHERE runId=@memoryid;
```

The old query was simpler than the new one but the old one turned out to be significantly slower. The main differences between the new and old queries are the following. All the data was retrieved from two tables with the old query; the sizes of the old table were greater than the new ones which means that more rows of data had to be gone through to find the needed results. There were a lot of duplicates of *confName* and *swBuild* in tables *stabi_memoryruntable* and *stabi_memoryusagetable* in the old database. Normalization, inner joins and indexes were applied the same way as explained in Query #1 explanation. Indexes were set during the design phase on attributes that need to be looked up fast: *configuration_id*, *software_id* and *memory_run_id* in this case. In conclusion, the new query performs better because of normalization, indexing and use of join-operations.

Query #3 explanation

The purpose of the query is to retrieve data from the table *memory_usage* about which cards were measured on a specific memory run. The query was chosen for comparison because it is a query that is executed often to retrieve the card data for a specific configuration and software and memory run. With the old database it took 28.8947 seconds to retrieve the data while with the new database it took 0.0016 seconds to retrieve the same data. The new query is as follows:

```
START TRANSACTION;
```

```
SET   @memoryid=(SELECT   memory_run_id   FROM   memory_run   INNER   JOIN
configuration ON
memory_run.configuration_id=configuration.configuration_id  INNER  JOIN
software    ON    memory_run.software_id    =    software.software_id    WHERE
configuration_name= 'X' AND software_version='X');
SELECT     distinct     card_name     FROM     memory_usage     WHERE
memory_run_id=@memoryid and card_name like '%0x10%';
COMMIT;
```

The old query in the old database to retrieve the same data was as follows:

```
START TRANSACTION;
SET @memoryid=(SELECT PID FROM stabi_memoryruntable WHERE confName= 'X'
AND swlevel='X');
SELECT    distinct    cardName    FROM    stabi_memoryusagetable    WHERE
runId=@memoryid and cardName like '%0x10%';
COMMIT;
```

The old query was simpler than the new one but the old one turned out to be significantly slower. The main differences between the new and old queries are the following. All the data was retrieved from two tables with the old query; the sizes of the old table were greater than the new ones which means that more rows of data had to be gone through to find the needed results. There were a lot of duplicates of *confName* and *swlevel* in tables *stabi_memoryruntable* and *stabi_memoryusagetable* in the old database. Normalization, inner joins and indexes were applied the same way as explained in Query #1 explanation. Indexes were set during the design phase on attributes that need to be looked up fast: *configuration_id*, *software_id* and *memory_run_id* in this case. In conclusion, the new query performs better because of normalization, indexing and use of join-operations.

Query #4 explanation

The purpose of this query is to get data from table *testprofile_results* for a specific configuration and time frame. The query was chosen for comparison because it is a query that is executed often to retrieve test data, such as test names, test results, start times and finish times, for a specific configuration. With the old database it took 0.1746 seconds to retrieve the data while with the new database it took 0.0221 seconds to retrieve the same data. The new query is as follows:

```
SELECT * FROM testprofile_results
INNER  JOIN  configuration  ON  testprofile_results.configuration_id  =
configuration.configuration_id
WHERE configuration_name = 'X'
AND date > '2016-09-01';
```

The old query in the old database to retrieve the same data was as follows:

```
SELECT * FROM testprofile_results
WHERE confName='X'
AND dayId > '2016-09-01 00:00:00';
```

The old query was simpler than the new one but the old one turned out to be slightly slower. The main differences between the new and old queries are the following. All the data was retrieved from one table with the old query; the size of the old table was greater than the new one which means that more rows of data had to be gone through to find the needed result. There were a lot of duplicates of *confName* in table *testprofile_results* in the old database. Normalization, inner joins and indexes were applied the same way as explained in Query #1 explanation. Index was set during the design phase on attribute that needs to be looked up fast; *configuration_id* in this case, In conclusion, the new query performs better because of normalization, indexing and use of join-operation.

Query #5 explanation

The purpose of this query is to get data from table *configuration_has_software* to get all tested software versions for a specific configuration. The query was chosen for comparison because it is a query that is executed often to retrieve a list of all tested software versions for a specific configuration. With the old database it took 0.0024 seconds to retrieve the data while with the new database it took 0.0005 seconds to retrieve the same data. The new query is as follows:

```
SELECT software_version
FROM software
INNER  JOIN  configuration_has_software  ON  software.software_id  =
configuration_has_software.software_id
INNER  JOIN  configuration  ON  configuration.configuration_id  =
configuration_has_software.configuration_id
WHERE configuration_name = 'X'
```

The old query in the old database to retrieve the same data was as follows:

```
SELECT swlevel FROM swlist WHERE confName='X';
```

The old query was simpler than the new one but the old one turned out to be slightly slower. The main differences between the new and old queries are the following. All the data was retrieved from one table with the old query; the size of the old table was greater than the new one which means that more rows of data had to be gone through to find the needed result. There were a lot of duplicates of *confName* and *swlevel* in table *swlist* in the old database. Normalization, inner joins and indexes were applied the same way as explained in Query #1 explanation. Indexes were set during the design phase on attributes that need to be looked up fast; *configuration_id* and *software_id* in this case, In conclusion, the new query performs better because of normalization, indexing and use of join-operation.

9.3.2 Summary

From the results of the queries it can be seen that the new database has faster read-performance in all tested queries. The tested queries were ones that are executed often and thus chosen for comparison. Some differences were much more noticeable than others. The difference with the fourth query is hardly noticeable but the second and third queries were usability and performance wise noticeable better with the new database.

The new database is noticeably performing better than the old database. The actual better read-performance can be seen in the daily usage and the presented query results advocate this claim as well. The maintainability of the database can be considered good. It is easy to add something new into the database design without negatively affecting the performance. This claim was confirmed by a representative of the case company as new tables and columns have been added into the database throughout the process.

Join-operations were used in the new database. According to Ramakrishnan and Gehrke (2000) join-operation is the most used practice to combine data from several tables. The amount of text used in queries was lower when no join-operations were used but the actual query times became faster when join-operations were used. In the old database there were no join-operations used as there weren’t any relations built between the tables. This was also the reason for several duplicates in the old database. The same data had to be stored in several tables because it couldn't be reached via relations. The new database is highly based on relations which made it a necessity to use join-operations to retrieve data from the database.

Normalization was applied in the new database as according to Dorsey and Rosenblum (2006) it helps to build a structure that is easy to be maintained and it eases the data retrieving from the database. Accroding to Wise (2000) redundancy is eliminated with normalization which was verified in this case. The old database was not normalized so there was a lot of redundancy whereas no redundancy is found in the new database because of normalization.

Indexing was applied in the new database. Indexing is basically done to make the queries faster (Elmasri & Navathe, 2009; Ramakrishnan & Gehrke, 2000). Indexes in the new database were set on attributes that are queried often such as *configuration_id* and *software_id*. Those attributes are also foreign keys that are used to relate the tables to each other as instructed by Wise (2000).

The use of attributes such as *start_time*, *end_time* and *finish_date* in the new database help with maintaining the database. Archiving is easy because one can define a time frame from which the data is archived by using the previously mentioned attributes.

To sum things up, it turned out that a MySQL database can be both; slow and fast. The outcome is highly dependent on the database design. I was able to meet the main requirement given by the case company, good performance, by applying normalization, indexing, relations and join-operations in the new database. None of the previously mentioned techniques were used in the old database. The second main requirement given by the case company, good maintainability, was met by applying normalization and date and time attributes. Thanks to normalization it is now easy to add new tables into the database when needed which eases maintenance. This doesn't require much effort by the user and new tables don't affect the functionality of the existing database. The use of time and date attributes help with archiving which eases maintenance as well.

It is unknown if it would have been possible to achieve better performance with the old database if some of the previously mentioned techniques were applied to it. It was requested by the case company that I design and implement a new database instead of refactoring the old one.

10. Discussion and implications

In this section the main findings of the thesis including the findings from the earlier research and my empirical findings are presented. This is done by answering the research questions. The sub-research questions are answered first, and the main research question is answered last.

The first sub research question was "*What database management systems do currently exist?*".

At the time of this study there existed 284 different database management systems; relational and non-relational DBMSs. The five most popular relational DBMSs were MySQL, PostgreSQL, Microsoft SQL server, Oracle and DB2. (DB-Engines, 2016a.) Relational databases were more popular than non-relational databases (Feuerlicht, 2010). At the time of this study I was able to find earlier research and comparison about the following non-relational databases; Aerospike, Cassandra, CouchDB, Couchbase, HBase, MongoDB and Voldemort. The number of existing DBMSs, 284, was high, but there was earlier research available only about the most popular DBMSs. The previously mentioned matters imply that nowadays there are many DBMSs to choose from but there is a batch of DBMSs that is more popular and more researched.

The second sub research question was "*What are the differences between relational and non-relational database management systems in terms of usage and performance?*".

Non-relational DBMSs can handle unstructured data better than relational DBMSs (Leavitt, 2010) and non-relational DBMSs do better performance-wise than relational DBMSs as the amount of data increases (Lai, 2009; Leavit, 2010). Non-relational databases are mostly open source (Leavitt, 2010) so it costs less to take them into use compared to relational DBMSs. All the seven non-relational DBMSs studied in thesis were open-source. Two out of the five most popular relational DBMSs, MySQL and PostgreSQL, were open-source while the other three were commercial products.

The data models of non-relational DBMSs are simpler compared to the data models of relational DBMSs which make non-relational DBMSs faster (Leavitt, 2010). It is also known that as the performance increases, the precision tends to decrease simultaneously (Leavit, 2010; Nayak et al., 2013) which implies that the usage areas of relational and non-relational DBMSs are different.

According to Nayaka et al. (2013) non-relational databases were difficult to maintain. The lack of support of SQL means that manual query programming is needed (Leavitt, 2010). This implies that the usage of non-relational databases requires more skills and time so the threshold to start using non-relational DBMSs is high.

The previously mentioned matters imply, in short, that non-relational DBMSs perform better when the amount of data get higher but relational DBMSs are more precise, reliable and easier to maintain.

The third sub research question was "*What are the differences between different relational database management systems?*".

I studied the differences between MySQL and PostgreSQL as they were the two most popular open-source relational DBMSs at the time of this study according to DB-Engines

(2016a). Commercial products weren't studied because it was a requirement given by the case company that the new database had to be cost-free.

According to the official website of MySQL, MySQL is the most popular open-source relational DBMS (MySQL, 2016a). Also the findings about popularity and free online support presented earlier in this thesis in Table 3 advocate that claim; MySQL is more used than PostgreSQL.

There haven't been made up-to-date comparisons of MySQL and PostgreSQL. However, those DBMSs have been compared before. According to Starcu-Mara and Bauman (2008), Giacomo (2005) and Malaysian Public Sector (2009) MySQL had better read-performance, better write-performance, better stability, better scalability and better API support than PostgreSQL. It should be mentioned that the differences between the two DBMSs are very minor.

The previously mentioned matters imply that there aren't much differences between MySQL and PostgreSQL. However, when taking minor differences into account, it can be deduced that MySQL is better than PostgreSQL from performance and usability point of view.

The fourth sub research question was "*What are the differences between different non-relational database management systems?*".

Different non-relational DBMSs have different approaches for storing data, but none of them is relational. The three most common types of non-relational databases are key-value stores, column-oriented databases and document-based stores. (Leavitt, 2010.)

Key-value store systems can store structured and unstructured data (Leavitt, 2010). A key-value consists of two parts. The first part is the key and the second part is the value. Key-value store systems are known for high concurrency, fast look-ups and options for mass storage. The lack of schema is considered as the biggest weakness of key-value store systems. Key-value databases are used, for example, in online shopping websites. (Nayak et al., 2013.)

A column-oriented database consists of a set of individual columns that are vertically partitioned. The columns are stored into the database separately which makes it possible to not necessary read entire rows when queries are executed. The needed attributes are read instead. (Adabi et al., 2013.) Column-oriented databases are known for high scalability. The column-oriented approach is more efficient than row-store approach (Nayak et al., 2013) that is used in relational databases. Column-oriented databases are used, for example, in data mining and analytic applications (Nayak et al., 2013).

Document-based stores consist of data as collections of documents (Leavitt, 2010). Document-bases stores are schemaless (Couchbase, 2016). This means that the sizes of documents are not pre-defined (Leavitt, 2010). Document-based stores are known for flexibility, great performance and great horizontal scalability. It is not good to use document-based stores when relations and normalization are needed. Document-based stores are used, for example, in blog software. (Nayak et al., 2013.)

Lourenco et al. (2015) studied the differences between seven non-relational DBMSs: Aerospike, Cassandra, Couchbase, CouchDB, HBase, MongoDB and Voldemort. They differed from each other in terms of quality attributes and database models. The compared quality attributes were availability, consistency, durability, maintainability, read-performance, recovery time, reliability, robustness, scalability, stabilization time, and write performance. The results were presented earlier in this thesis in Table 6.

All seven DBMSs were open-source. Two of them were key-value store systems, two of them were column-oriented databases and three of them were document-based databases. All seven non-relational DBMSs differed from each other in terms of quality attributes.

I researched the popularity and free online support in the same way as with the relational DBMSs. Differences were found in terms of popularity and free online support. However, MongoDB could be considered as the most popular non-relational databases with the most amount of free online support out of the seven DBMSs. The results were presented earlier in this thesis in Table 7.

The previously mentioned matters imply that there are differences between non-relational databases. They are designed to solve different kind of problems in terms of data storage. Some of them are more popular than the others. They also differ from each other in terms of quality attributes.

The fifth sub research question was "*How do different database management systems match the needs of a software testing team when the stored data is heavily filtered?*".

I studied the differences of nine DBMSs: two relational DBMSs and seven non-relational DBMSs. After the preliminary comparison I came up with three DBMSs that matched the needs and the requirements best. The DBMSs were MySQL, Cassandra and HBase. However, one DBMS had to be chosen. After the final comparison between the three DBMSs, it turned out that MySQL was the best DBMS to meet the requirement given by the case company.

The main research question was "*What database design principles and techniques should be followed to make a database well-performing and easily maintainable?*"

It turned out that by applying normalization, indexing, relations and join-operations, the new database became very well-performing. Normalization was applied as instructed in section 4.8 Database normalization. Normalization of databases is considered important because it helps to build a logical structure that is easy to be maintained. The formal reason for normalization is the ease of retrieving data from the database. (Dorsey & Rosenblum 2006). The data was divided into several tables in the new database. By doing this the sizes of the tables remained lower and no duplications of data is stored.

Relations were built between the tables, and join-operations are used to combine the data from several tables as mentioned by Ramakrishnan and Gehrke (2000). Indexing makes the queries faster (Elmasri & Navathe, 2009; Ramakrishnan & Gehrke, 2000). Indexes should be built on the attributes that the wanted speeded up queries are related to. (Ramakrishnan & Gehrke, 2000.)

None of the previously mentioned techniques were applied in the old database. The performance of the queries were much better in the new database where those principles and techniques were applied.

Good maintainability was met by applying normalization and date and time attributes in the tables. Normalization makes easy to add new tables into the database when needed which eases maintenance. This doesn’t require much effort by the user and new tables don’t affect the functionality of the existing database. The use of time and date attributes help with archiving which eases maintenance as well. Time and date attributes were used in the old database but no other previously mentioned design principles or techniques were used in the old database.

The findings in my thesis imply that there are nowadays large amount of DBMSs to choose from. Not all DBMS are designed to match the same needs. The suitability of a certain DBMS depends highly on the case and which attributes are valued. There are many good choices to choose from. In the case of this thesis the stored data is heavily filtered which means that there wasn’t a necessary need to design and implement a non-relational database. Relational databases perform well when the data is structured and the amount of data is not very high. A database can be both slow and fast even though the same DBMS is used; MySQL in this case. Performance and maintainability can be improved by applying design principles and techniques such as normalization and indexing.

11. Conclusions

This thesis provided an overview of the process of designing, implementing and evaluating a database for a software testing team in a real life case. Different relational and non-relation database management systems were studied. Prompt looks into data analytics and distributed databases were taken as well.

The final outcome of this thesis was the artefact: the new database. The new database outperformed the old database and the representative of the case company was satisfied with the outcome – the new database filled the requirements given by the case company. That was the main contribution of this thesis. The database design principles and techniques applied that made the new database well-performing and easily maintainable in this case were also discussed.

Other contributions were the following. This thesis strengthened the utility of following the design science research guidelines presented by Hevner et al. (2004). All the seven guidelines were followed which led to a satisfying outcome in the end. The guidelines were followed the following way. 1. **Design as an artifact**: The artifact, the new database, was designed, implemented and evaluated. It addressed the case company's problem which was the lack of a proper database for a software testing team's internal use. 2. **Problem relevance**: A software testing team needed a database that has a better performance and better maintainability than the old database. The new database improves daily work processes. Work processes inside the organizational unit became more effective when the new database was taken into use. 3. **Design evaluation**: The database was evaluated experimentally by simulation. The query times of the new and old database were compared and the new database turned out to be faster and the case company was satisfied with the results. 4. **Research contribution**: The main contributions were the new database and the results from the evaluation. There is nowadays a big boom about non-relational DBMSs and their better performance compared to non-relational DBMS. It turned out that a relational DBMS can become very effective as well when proper techniques, such as indexing and normalization, are applied in the design phase. In short, the research contribution was the following. It is not necessary to switch from a relational DBMSs to a non-relational DBMS when better performance is desired. A relational DBMS can perform very well as well when it is properly designed by applying techniques found from research and literature. This became true in this case when the amount of stored data wasn't very high. 5. **Research rigor**: The methods for constructing the new database from the existing research and literature. The method for evaluating the database depended on the case company. They wanted a better performance so query times were compared in a real environment with real data. 6. **Design as a search process**: The design phase was based on the existing research and literature found about databases. The requirements came from the case company. The design and the test cycle was repeated until the requirements were met which took three iterations rounds. 7. **Communication of research**: This study communicated with technology-oriented audience by explaining why particular technologies and techniques were used when conducting the work, and this study communicated with management-oriented audience by explaining that the better performing database improved daily work within the software testing team in the case company.

This thesis also strengthened the conception of following guidelines and knowledge from previous database research and literature. The outcome of doing so was satisfying in this case. Normalization and indexing were applied as they were told to increase the database performance which was confirmed in this study. A black box testing was conducted for the database also which confirmed the functionality of the database.

I presented my thesis to other software testing teams inside the case company. The feedback was positive; other teams were also interested to take the database into use. By the time I finished this thesis, one team inside the company had already taken the database into use.

11.1 Limitations of the study

The assumptions about the utility of other database management systems were based on previous research and literature. The final database was designed, implemented and evaluated using only one database technology. However, this was agreed with the case company because it would have taken a lot more time to go through the same process using different database management systems. There would have been more detailed results about the utility of other DBMSs if the database had been implemented using different technologies.

It was known that at the time of this study there existed 284 different DBMSs. Only 9 of them were studied more deeply in this thesis. It is now unknown if there had been another more suitable DBMSs to meet the requirements in this thesis. However, it should be mentioned that it was difficult to find knowledge about other DBMSs at the time of this study.

Most of the literature used in this study is from books or scholarly sources. It should be mentioned that non-scholarly source were used as well. For example, the database normalization instructions by Wise (2000) and the database rankings from DB-Engines (2016a) are non-scholarly. However, it turned out that normalization by Wise (2000) was valuable in this case.

11.2 Recommendation for future research

In the context of the case company it would be beneficial to research what changes should be made to the database if one wanted to store unstructured data efficiently into it. This could be an actual real life scenario as there were other teams interested in the outcome of this thesis and not all the software testing teams within the case company are storing as heavily filtered data as in the stability testing team. Some teams are storing log data that can be considered big data. The amount of data is much higher and it is more complex because it is not filtered. My suggestion for a future research question would be "*How to combine a relational database management system and a non-relational database management system?*".

References

Abadi, D., Boncz, P., Harizopoulos, S., Idreos, S., & Madden, S. (2013). The Design and Implementation of Modern Column-Oriented Database Systems. *Foundations and Trends in Databases, 5*(3), 197-280.

Aerospike. (2016). Aerospike NoSQL Database Technology. Retrieved April, 6, 2016, from http://www.aerospike.com/technologies/

Al-Shamailh, A. (2015). An Experimental Comparison of ER and UML Class Diagrams. *International Journal of Hybrid Information Technology*, *8*(2), 279-288.

Banafa, A. (2014). Small Data vs. Big Data: Back to the basics. Retrieved May, 30, 2016 from https://www.linkedin.com/pulse/20140703195144-246665791-small-data-vs-big-data-back-to-the-basics

Bryant, C. (2014). A Guide to Open Source Cloud Computing Software. Retrieved November, 22, 2016 from http://www.tomsitpro.com/articles/open-source-cloud-computing-software,2-754-8.html

Cassandra. (2016). Retrieved April, 6, 2016, from http://cassandra.apache.org/

Chen, M., Mao, S., & Liu, Y. (2014). Big Data: A Survey. *Mobile Networks and Applications*, *19*(2), 171-209.

Churcher, C. (2007). *Beginning Database Design: From Novice to Professional*. Apress.

Codd, E. F. (1970). A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, *13*(6), 377-387.

Conrad, T. (2004). *PostgreSQL vs. MySQL vs. Commercial Databases: It's All About What You Need*. Retrieved May, 12, 2016, from: http://www.devx.com/dbzone/Article/20743

Couchbase. (2016). What is Couchbase Server? Retrieved April, 6, 2016, from http://www.couchbase.com/nosql-databases/couchbase-server

CouchDB. (2016). Retrieved April, 6, 2016, from http://couchdb.apache.org/

DB-Engines. (2016a). DB-Engines Ranking. Retrieved April, 25, 2016, from http://db-engines.com/en/ranking

DB-Engines. (2016b). System Properties Comparison MySQL vs. PostgreSQL. Retrieved May, 5, 2016, from http://dbengines.com/en/system/MySQL%3BPostgreSQL

DeWitt, D. J., Naughton, J. F., & Schneider, D. A. (1991). Parallel Sorting on a Shared-Nothing Architecture using Probabilistic Splitting. In *Proceedings of the First International Conference on Parallel and Distributed Information Systems, 1991.*, (pp. 280-291). IEEE.

Dorsey, P., & Rosenblum, M. (2006). *Oracle PL/SQL For Dummies*. Indianapolis, Indiana: Wiley Publishing Inc.

Douglas, K., & Douglas, S. (2003). *PostgreSQL: A Comprehensive Guide to Building, Programming, and Administering PostgreSQL Databases*. SAMS publishing.

Dulin, O. (2015). Ten Questions to Consider Before Choosing Cassandra. *The Dulin Report. Software engineering in the age of the cloud*. Retrieved, June, 16, 2016, from https://thedulinreport.com/2015/08/08/ten-questions-to-consider-before-choosing-cassandra/

Elmasri, R., & Navathe, S. (2009). *Fundamentals of database systems.* Pearson Education.

Elnaffar, S., Martin, P., & Horman, R. (2002, November). Automatically Classifying Database Workloads. In *Proceedings of the eleventh international conference on Information and knowledge management* (pp. 622-624). ACM.

Feuerlicht, G. (2010, April). Database Trends and Directions: Current Challenges and Opportunities. In *DATESO* (pp. 163-174).

Garcia-Molina, H., Ullman, J. D., & Widom, J. (2009). *Database Systems: The Complete Book. Second edition*. New Jersey: Prentice Hall.

Giacomo, M. D. (2005). MySQL: Lessons Learned on a Digital Library. *Software, IEEE, 22*(3), 10-13.

Hauer, P. (2015). Why Relational Databases are not the Cure-All. Strength and Weaknesses. Retrieved April, 16, 2016, from http://blog.philipphauer.de/relational-databases-strength-weaknesses-mongodb/

Han, J., Haihong, E., Le, G., & Du, J. (2011, October). Survey on NoSQL database. In *Pervasive computing and applications (ICPCA) on 2011 6th international conference* (pp. 363-366). IEEE.

HBase. (2016). Retrieved April, 6, 2016, from https://hbase.apache.org/

Hevner, A., March, R., Salvatore, T., Park, J. & Ram, S. 2004. Design Science in Information Systems Research. *MIS Quarterly, 28*(1), 75–105.

Hippel, E. V., & Krogh, G. V. (2003). Open Source Software and the "Private-Collective" Innovation Model: Issues for Organization Science. *Organization science, 14*(2), 209-223.

IBM. (2016a). What is HBase? Retrieved June, 16, 2016, from https://www-01.ibm.com/software/data/infosphere/hadoop/hbase/

IBM. (2016b). IBM Knowledge Center. Retrieved June, 16, 2016, from https://www.ibm.com/support/knowledgecenter/SSPT3X_2.1.2/com.ibm.swg.im.infosphere.biginsights.bigsql.doc/doc/bsql_create_index.html

Lai, E. (2009). No to SQL? Anti-database movement gains steam. *Computerworld Software, July, 1.* Retrieved April, 7, 2016, from http://www.computerworld.com/article/2526317/database-administration/no-to-sql--anti-database-movement-gains-steam.html

Lake, P., & Crowther, P. (2013). *Concise guide to databases*. Springer, London.

Leavitt, N. (2010). Will NoSQL databases live up to their promise?. *Computer, 43*(2), 12-14.

Letkowski, J. (2015). Doing database design with MySQL. *Journal of Technology Research, 6*, 1.

Levin, S. (2015). Choosing a Database for Analytics. Retrieved May, 30, 2016, from https://segment.com/blog/choosing-a-database-for-analytics/

Li, G., Ooi, B. C., Feng, J., Wang, J., & Zhou, L. (2008, June). EASE: An Effective 3-in-1 Keyword Search Method for unstructured, Semi-structured and Structured data. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data* (pp. 903-914). ACM.

Loukides, M. (2012). The NoSQL movement. How to think about choosing a database. Retrieved November, 22, 2016, from http://radar.oreilly.com/2012/02/nosql-non-relational-database.html

Lourenço, J. R., Cabral, B., Carreiro, P., Vieira, M., & Bernardino, J. (2015). Choosing the right NoSQL database for the job: a quality attribute evaluation. *Journal of Big Data, 2*(1), 1-26.

Mahajan, G. (2012). Query Optimization in DDBS. *International Journal of Computer Applications & Information Technology, 1*(1).

Maier, D. (1983). The theory of relational databases (Vol. 11). Rockville: Computer science press.

Malaysian Public Sector. (November, 2009). COMPARISON REPORT ON MySQL and PostgreSQL. *OPEN SOURCE SOFTWARE (OSS) PROGRAMME*. Retrieved May, 12, 2016, from: http://knowledge.oscc.org.my/practice-areas/rnd/benchmark-report/comparison-report-on-mysql-and-postgresql/view?searchterm=mysql%20and%20postgresql

McNulty, E. (2014). SQL VS. NOSQL – WHAT YOU NEED TO KNOW. *Dataconomy*. Retrieved June, 16, 2016, from http://dataconomy.com/sql-vs-nosql-need-know/

Mesmoudi, A., & Hacid, M. S. (2014, March). A Test Framework for Large Scale Declarative Queries: Preliminary Results. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing* (pp. 858-859). ACM.

Minsky, N. (1974). Another look at data-bases. *ACM SIGMOD Record, 6*(4), 9-17.

MongoDB. (2016). Introduction to MongoDB. Retrieved April, 6, 2016, from https://docs.mongodb.org/manual/introduction/

Moniruzzaman, A. B. M., & Hossain, S. A. (2013). NoSQL Database: New Era of Databases for Big data Analytics-Classification, Characteristics and Comparison. *International Journal of Database Theory and Application, 6*(4), 1-13.

MySQL. (2016a). MySQL 5.1 Reference Manual. Retrieved April, 25, 2016, from http://dev.mysql.com/doc/refman/5.1/en/what-is-mysql.html

MySQL. (2016b). MySQL Workbench 6.3. Retrieved August, 23, 2016, from https://www.mysql.com/products/workbench/

MySQL. (2016c). Transaction Isolation Levels. Retrieved November, 22, 2016 from https://dev.mysql.com/doc/refman/5.7/en/innodb-transaction-isolation-levels.html

Nayak, A., Poriya, A., & Poojary, D. (2013). Type of NOSQL Databases and its Comparison with Relational Databases. *International Journal of Applied Information Systems, 5*(4), 16-19.

Neo4j. (2016). Graph Databases for Beginners: Why Graphs Are The Future. Retrieved April, 14, 2016, from http://neo4j.com/blog/why-graph-databases-are-the-future/

Plattner, H. (2009, June). A Common Database Approach for OLTP and OLAP Using an In-Memory Column Database. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data* (pp. 1-2). ACM.

PostgreSQL. (2016). Transaction Isolation. Retrieved November, 22, 2016, from https://www.postgresql.org/docs/9.2/static/transaction-iso.html

Pressman, R. S. (2010). *Software Engineering: A Practitioner's Approach*. The McGraw-Hill Companies.

Project Voldemort. (2016). Retrieved April, 6, 2016, from http://www.project-voldemort.com/voldemort/

Ramakrishnan, R., & Gehrke, J. (2000). *Database management systems*. Osborne/McGraw-Hill.

Sharma, K., & Bhardwaj, A. (2014). Types of Queries in Database System. *International Journal of Computer Applications & Information Technology*, *7*(2), 149.

Silberschatz, A., Stonebraker, M., & Ullman, J. (1996). Database Research: Achievements and Opportunities Into the 21st Century. *ACM Sigmod Record, 25*(1), 52-63.

Stancu-Mara, S., & Baumann, P. (2008, September). A Comparative Benchmark of Large Objects in Relational Databases. In *Proceedings of the 2008 international symposium on Database engineering & applications* (pp. 277-284). ACM.

Stonebraker, M. (2010). Errors in Database Systems, Eventual Consistency, and the CAP Theorem. *Communications of the ACM, BLOG@ ACM.*

Strauch, C. (2011). NoSQL Databases. *Lecture Notes, Stuttgart Media University.* Retrieved April, 7, 2016, from http://www.christof-strauch.de/nosqldbs.pdf

Techopedia. (2016). In-Memory Database (IMDB). Retrieved June, 16, 2016, from https://www.techopedia.com/definition/28541/in-memory-database

Tutorialspoint. (2016). HBase – Shell. Retrieved June, 16, 2016, from http://www.tutorialspoint.com/hbase/hbase_shell.htm

Vardi, M. Y. (1982, May). The complexity of relational query languages. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing* (pp. 137-146). ACM.

Walls, J. G., Widmeyer, G. R., & El Sawy, O. A. (1992). Building an Information System Design Theory for Vigilant EIS. *Information systems research, 3*(1), 36-59.

Ward, J. S., & Barker, A. (2013). Undefined By Data: A Survey of Big Data Definitions.

WikiVS. (May, 2016). MySQL vs PostgreSQL. Retrieved March, 19, 2016, from https://www.wikivs.com/wiki/MySQL_vs_PostgreSQL

Wise, B. (2000, July). Database Normalization and Design Techniques. Retrieved April, 18, 2016, from http://www.phpbuilder.com/columns/barry20000731.php3

Wisnesky, R. (2014). Functional Query Languages with Categorical Types. *Doctoral dissertation, Harvard University.*

Özsu, M. T., & Valduriez, P. (2011). *Principles of Distributed Database Systems, Third Edition.* New York: Springer.

Appendix A. The generated SQL scripts

1. A new database named *stabilitydb* is created and the new database is taken into use for the upcoming commands.

```
CREATE SCHEMA IF NOT EXISTS `stabilitydb` DEFAULT CHARACTER SET utf8 ;
USE `stabilitydb` ;
```

2. A new table named *configuration* is created along with its attributes.

```
CREATE TABLE IF NOT EXISTS `stabilitydb`.`configuration` (
  `configuration_id` INT NOT NULL AUTO_INCREMENT,
  `configuration_name` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`configuration_id`))
ENGINE = InnoDB;
```

3. A new table named *software* is created along with its attributes.

```
CREATE TABLE IF NOT EXISTS `stabilitydb`.`software` (
  `software_id` INT NOT NULL AUTO_INCREMENT,
  `software_version` VARCHAR(45) NULL,
  PRIMARY KEY (`software_id`))
ENGINE = InnoDB;
```

4. A new table named *fault* is created along with its attributes, indexes and relations to tables *software* and *configuration*.

```
CREATE TABLE IF NOT EXISTS `stabilitydb`.`fault` (
  `fault_id` INT NOT NULL AUTO_INCREMENT,
  `actCritical` TEXT NULL,
  `inactCritical` TEXT NULL,
  `software_id` INT NOT NULL,
  `configuration_id` INT NOT NULL,
  PRIMARY KEY (`fault_id`, `software_id`, `configuration_id`),
  INDEX `fk_fault_software1_idx` (`software_id` ASC),
  INDEX `fk_fault_configuration1_idx` (`configuration_id` ASC),
  CONSTRAINT `fk_fault_software1`
    FOREIGN KEY (`software_id`)
    REFERENCES `stabilitydb`.`software` (`software_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_fault_configuration1`
    FOREIGN KEY (`configuration_id`)
    REFERENCES `stabilitydb`.`configuration` (`configuration_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

5. A new table named *configuration_info* is created along with its attributes, index and relation to table *configuration*.

```
CREATE TABLE IF NOT EXISTS `stabilitydb`.`configuration_info` (
  `idconfiguration_info_id` INT NOT NULL AUTO_INCREMENT,
  `configuration_id` INT NOT NULL,
  `hw` VARCHAR(45) NULL,
  `units` TEXT NULL,
  `local_pc_ip` VARCHAR(45) NULL,
  `profile` VARCHAR(75) NULL,
  PRIMARY KEY (`idconfiguration_info_id`, `configuration_id`),
  INDEX  `fk_configuration_info_configuration1_idx`  (`configuration_id`
ASC),
  CONSTRAINT `fk_configuration_info_configuration1`
    FOREIGN KEY (`configuration_id`)
    REFERENCES `stabilitydb`.`configuration` (`configuration_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

6. A new table named *test_run* is created along with its attributes, indexes and relations to tables *configuration* and *fault*.

```
CREATE TABLE IF NOT EXISTS `stabilitydb`.`test_run` (
  `test_run_id` INT NOT NULL AUTO_INCREMENT,
  `start_time` DATETIME NULL,
  `configuration_id` INT NOT NULL,
  `fault_id` INT NOT NULL,
  `software_id` INT NOT NULL,
  `uptime` VARCHAR(20) NULL,
  PRIMARY   KEY   (`test_run_id`,   `configuration_id`,   `fault_id`,
`software_id`),
  INDEX `fk_test_run_configuration1_idx` (`configuration_id` ASC),
  INDEX `fk_test_run_fault1_idx` (`fault_id` ASC, `software_id` ASC),
  CONSTRAINT `fk_test_run_configuration1`
    FOREIGN KEY (`configuration_id`)
    REFERENCES `stabilitydb`.`configuration` (`configuration_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_test_run_fault1`
    FOREIGN KEY (`fault_id` , `software_id`)
    REFERENCES `stabilitydb`.`fault` (`fault_id` , `software_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

7. A new junction table named *configuration_has_software* is created with its attributes, indexes and relations to tables *configuration* and *software*.

```
CREATE TABLE IF NOT EXISTS `stabilitydb`.`configuration_has_software` (
  `configuration_id` INT NOT NULL,
  `software_id` INT NOT NULL,
  PRIMARY KEY (`configuration_id`, `software_id`),
  INDEX  `fk_configuration_has_software1_software1_idx`  (`software_id`
ASC),
  INDEX             `fk_configuration_has_software1_configuration1_idx`
(`configuration_id` ASC),
  CONSTRAINT `fk_configuration_has_software1_configuration1`
    FOREIGN KEY (`configuration_id`)
    REFERENCES `stabilitydb`.`configuration` (`configuration_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_configuration_has_software1_software1`
    FOREIGN KEY (`software_id`)
    REFERENCES `stabilitydb`.`software` (`software_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

8. A new table named *test_equipment_a* is created along with its attributes.

```
CREATE TABLE IF NOT EXISTS `stabilitydb`.`test_equipment_a` (
  `test_equipment_a_id` INT NOT NULL AUTO_INCREMENT,
  `test_equipment_a_name` VARCHAR(45) NULL,
  `technology` VARCHAR(45) NULL,
  `team` VARCHAR(45) NULL,
  PRIMARY KEY (`test_equipment_a_id`))
ENGINE = InnoDB;
```

9. A new table named *configuration_has_test_equipment_a* is created along with its attributes, indexes and relations to tables *configuration* and *test_equipment_a*.

```
CREATE            TABLE            IF            NOT            EXISTS
`stabilitydb`.`configuration_has_test_equipment_a` (
  `configuration_id` INT NOT NULL,
  `test_equipment_a_id` INT NOT NULL,
  PRIMARY KEY (`configuration_id`, `test_equipment_a_id`),
  INDEX  `fk_configuration_has_dmts_dmts1_idx`  (`test_equipment_a_id`
ASC),
  INDEX                 `fk_configuration_has_dmts_configuration1_idx`
(`configuration_id` ASC),
  CONSTRAINT `fk_configuration_has_dmts_configuration1`
    FOREIGN KEY (`configuration_id`)
    REFERENCES `stabilitydb`.`configuration` (`configuration_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_configuration_has_dmts_dmts1`
    FOREIGN KEY (`test_equipment_a_id`)
    REFERENCES `stabilitydb`.`test_equipment_a` (`test_equipment_a_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

10. A new table named *test_equipment_a_test_run* is created along with its attributes, index and relation to table *test_equipment_a*.

```
CREATE TABLE IF NOT EXISTS `stabilitydb`.`test_equipment_a_test_run` (
  `test_equipment_a_test_run_id` INT NOT NULL AUTO_INCREMENT,
  `test_case_name` VARCHAR(45) NULL,
  `start_time` VARCHAR(45) NULL,
  `end_time` VARCHAR(45) NULL,
  `test_equipment_a_id` INT NOT NULL,
  PRIMARY KEY (`test_equipment_a_test_run_id`, `test_equipment_a_id`),
  INDEX `fk_dmts_test_run_dmts1_idx` (`test_equipment_a_id` ASC),
  CONSTRAINT `fk_dmts_test_run_dmts1`
    FOREIGN KEY (`test_equipment_a_id`)
    REFERENCES `stabilitydb`.`test_equipment_a` (`test_equipment_a_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

11. A new table named *test_equipment_b_results* is created along with its attributes, index and relation to table *configuration*.

```
CREATE TABLE IF NOT EXISTS `stabilitydb`.`test_equipment_b_results` (
  `test_equipment_b_result_id` INT NOT NULL,
  `runtime` VARCHAR(100) NULL,
  `text` VARCHAR(45) NULL,
  `configuration_id` INT NOT NULL,
  PRIMARY KEY (`test_equipment_b_result_id`, `configuration_id`),
  INDEX                 `fk_nemo_outdoor_results_configuration1_idx`
(`configuration_id` ASC),
  CONSTRAINT `fk_nemo_outdoor_results_configuration1`
    FOREIGN KEY (`configuration_id`)
    REFERENCES `stabilitydb`.`configuration` (`configuration_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

12. A new table named *test_equipment_c* is created along with its attributes.

```
CREATE TABLE IF NOT EXISTS `stabilitydb`.`test_equipment_c` (
  `test_equipment_c_id` INT NOT NULL AUTO_INCREMENT,
  `ip` VARCHAR(45) NULL,
  `phone` VARCHAR(45) NULL,
  `cluster` VARCHAR(5) NULL,
  `ftp_server` VARCHAR(45) NULL,
  `ue` INT NULL,
  `comments` TEXT NULL,
  `sector` VARCHAR(10) NULL,
  PRIMARY KEY (`test_equipment_c_id`))
ENGINE = InnoDB;
```

13. A new junction table named *configuration_has_test_equipment_c* is created along with its attributes, indexes and relations to tables *configuration* and *test_equipment_c*.

```
CREATE            TABLE            IF            NOT            EXISTS
`stabilitydb`.`configuration_has_test_equipment_c` (
  `configuration_id` INT NOT NULL,
  `test_equipment_c_id` INT NOT NULL,
  PRIMARY KEY (`configuration_id`, `test_equipment_c_id`),
  INDEX  `fk_configuration_has_genis_genis1_idx`  (`test_equipment_c_id`
ASC),
  INDEX                `fk_configuration_has_genis_configuration1_idx`
(`configuration_id` ASC),
  CONSTRAINT `fk_configuration_has_genis_configuration1`
    FOREIGN KEY (`configuration_id`)
    REFERENCES `stabilitydb`.`configuration` (`configuration_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_configuration_has_genis_genis1`
    FOREIGN KEY (`test_equipment_c_id`)
    REFERENCES `stabilitydb`.`test_equipment_c` (`test_equipment_c_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

14. A new table named *testcase_results* is created along with its attributes, indexes and relations to tables *configuration* and *software*.

```
CREATE TABLE IF NOT EXISTS `stabilitydb`.`testcase_results` (
  `testcase_results_id` INT NOT NULL AUTO_INCREMENT,
  `timestamp` VARCHAR(45) NULL,
  `testcase` VARCHAR(45) NULL,
  `result` TEXT NULL,
  `fail_reason` VARCHAR(45) NULL,
  `configuration_id` INT NOT NULL,
  `software_id` INT NOT NULL,
  PRIMARY     KEY     (`testcase_results_id`,     `configuration_id`,
`software_id`),
  INDEX   `fk_testcase_results_configuration1_idx`   (`configuration_id`
ASC),
  INDEX `fk_testcase_results_software1_idx` (`software_id` ASC),
  CONSTRAINT `fk_testcase_results_configuration1`
    FOREIGN KEY (`configuration_id`)
    REFERENCES `stabilitydb`.`configuration` (`configuration_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_testcase_results_software1`
    FOREIGN KEY (`software_id`)
    REFERENCES `stabilitydb`.`software` (`software_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

15. A new table named *cpu_load_data* is created along with its attributes, index and relation to table *configuration*.

```
CREATE TABLE IF NOT EXISTS `stabilitydb`.`cpu_load_data` (
  `data_id` INT NOT NULL AUTO_INCREMENT,
  `node_name` VARCHAR(45) NULL,
  `period_start` VARCHAR(45) NULL,
  `period_finish` VARCHAR(45) NULL,
  `cpu_load_info` TEXT NULL,
  `configuration_id` INT NOT NULL,
  PRIMARY KEY (`data_id`, `configuration_id`),
  INDEX `fk_cpu_load_data_configuration1_idx` (`configuration_id` ASC),
  CONSTRAINT `fk_cpu_load_data_configuration1`
    FOREIGN KEY (`configuration_id`)
    REFERENCES `stabilitydb`.`configuration` (`configuration_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

16. A new table named *testcase_setting* is created along with its attributes, index and relation to table *configuration*.

```
CREATE TABLE IF NOT EXISTS `stabilitydb`.`cpu_load_data` (
  `data_id` INT NOT NULL AUTO_INCREMENT,
  `node_name` VARCHAR(45) NULL,
  `period_start` VARCHAR(45) NULL,
  `period_finish` VARCHAR(45) NULL,
  `cpu_load_info` TEXT NULL,
  `configuration_id` INT NOT NULL,
  PRIMARY KEY (`data_id`, `configuration_id`),
  INDEX `fk_cpu_load_data_configuration1_idx` (`configuration_id` ASC),
  CONSTRAINT `fk_cpu_load_data_configuration1`
    FOREIGN KEY (`configuration_id`)
    REFERENCES `stabilitydb`.`configuration` (`configuration_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

17. A new table named *testprofile_results* is created along with its attributes, indexes and relations to tables *configuration* and *software*.

```
CREATE TABLE IF NOT EXISTS `stabilitydb`.`testprofile_results` (
  `testprofile_results_id` INT NOT NULL AUTO_INCREMENT,
  `test_name` VARCHAR(45) NULL,
  `finish_date` DATETIME NULL,
  `result` VARCHAR(10) NULL,
  `fail_reason` TEXT NULL,
  `date` VARCHAR(45) NULL,
  `time` VARCHAR(45) NULL,
  `ue_count` VARCHAR(45) NULL,
  `configuration_id` INT NOT NULL,
  `software_id` INT NOT NULL,
  PRIMARY    KEY    (`testprofile_results_id`,    `configuration_id`,
`software_id`),
  INDEX `fk_testprofile_results_configuration1_idx` (`configuration_id`
ASC),
  INDEX `fk_testprofile_results_software1_idx` (`software_id` ASC),
  CONSTRAINT `fk_testprofile_results_configuration1`
    FOREIGN KEY (`configuration_id`)
    REFERENCES `stabilitydb`.`configuration` (`configuration_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_testprofile_results_software1`
    FOREIGN KEY (`software_id`)
    REFERENCES `stabilitydb`.`software` (`software_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

18. A new table named *test_equipment_d_alarms* is created along with its attributes, indexes and relations to tables *configuration* and *software*.

```
CREATE TABLE IF NOT EXISTS `stabilitydb`.`test_equipment_d_alarms` (
  `test_equipment_d_id` INT NOT NULL AUTO_INCREMENT,
  `alarms` TEXT NULL,
  `start_time` VARCHAR(45) NULL,
  `insert_time` VARCHAR(45) NULL,
  `configuration_id` INT NOT NULL,
  `software_id` INT NOT NULL,
  PRIMARY    KEY    (`test_equipment_d_id`,    `configuration_id`,
`software_id`),
  INDEX `fk_rnc_alarms_configuration1_idx` (`configuration_id` ASC),
  INDEX `fk_rnc_alarms_software1_idx` (`software_id` ASC),
  CONSTRAINT `fk_rnc_alarms_configuration1`
    FOREIGN KEY (`configuration_id`)
    REFERENCES `stabilitydb`.`configuration` (`configuration_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_rnc_alarms_software1`
    FOREIGN KEY (`software_id`)
    REFERENCES `stabilitydb`.`software` (`software_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

19. A new table named *memory_run* is created along with its attributes, indexes and relations to tables *configuration* and *software*.

```
CREATE TABLE IF NOT EXISTS `stabilitydb`.`memory_run` (
  `memory_run_id` INT NOT NULL AUTO_INCREMENT,
  `start_time` VARCHAR(45) NULL,
  `usage_labels` VARCHAR(45) NULL,
  `configuration_id` INT NOT NULL,
  `software_id` INT NOT NULL,
  PRIMARY KEY (`memory_run_id`, `configuration_id`, `software_id`),
  INDEX `fk_memory_run_configuration1_idx` (`configuration_id` ASC),
  INDEX `fk_memory_run_software1_idx` (`software_id` ASC),
  CONSTRAINT `fk_memory_run_configuration1`
    FOREIGN KEY (`configuration_id`)
    REFERENCES `stabilitydb`.`configuration` (`configuration_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_memory_run_software1`
    FOREIGN KEY (`software_id`)
    REFERENCES `stabilitydb`.`software` (`software_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

20. A new table named *memory_usage* is created along with its attributes, indexes and relation to table *memory_run*.

```
CREATE TABLE IF NOT EXISTS `stabilitydb`.`memory_usage` (
  `usage_id` INT NOT NULL AUTO_INCREMENT,
  `fetch_time` DATETIME NULL,
  `card_name` VARCHAR(45) NULL,
  `global_usage` TEXT NULL,
  `process_usage` TEXT NULL,
  `memory_run_id` INT NOT NULL,
  `configuration_id` INT NOT NULL,
  `software_id` INT NOT NULL,
  PRIMARY   KEY   (`usage_id`,   `memory_run_id`,   `configuration_id`,
`software_id`),
  INDEX   `fk_memory_usage_memory_run1_idx`   (`memory_run_id`   ASC,
`configuration_id` ASC, `software_id` ASC),
  CONSTRAINT `fk_memory_usage_memory_run1`
    FOREIGN KEY (`memory_run_id` , `configuration_id` , `software_id`)
    REFERENCES    `stabilitydb`.`memory_run`    (`memory_run_id`    ,
`configuration_id` , `software_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```