Exercise 1 (SQ/ vs. NoSQL):

We will firstly describe the relational database management models, then the NoSQL database management models before comparing them. A relational database can be described as a collection of items with pre-established relationships between them. All of these items are organised into tables where columns represent item attributes. Each row of the table represents a single piece of data. In order to identify each item in a table uniquely, a primary key should be defined for each table. Examples of relational databases include MySQL, Postgres, Microsoft SQL Server and SQLite. We will now look at NoSQL database management models, NoSQL also known as "not only SQL" are non-tabular databases and store data differently than relational tables. NoSQL databases come in a variety of categories based on their data model. There are three main types of categories which are document, key-value, wide-column, and graph. They provide flexible schemas and are easily scalable with large amounts of data and high user loads. Some examples of non-relational databases include MonoDB, Apache Cassandra and Redis.

To compare relational and NoSQL database management models, we will examine the pros and cons of each model. Firstly, the pros for a relational database are as follows. It is great for structured data and turning it into categories. Due to these structured categories, your data is consistent in input which makes it easy to navigate. Relationships can be clearly defined between data points. It is great for complex queries as it uses an existing query language (SQL) which is widely known. Another huge pro is that the transactions are secure, which leads to high reliability in the database. Cons of a relational database include the fact that it requires an up-front schema definition which means resources will need to be devoted to the setup and maintenance of the database. There can also be no adaption to changing requirements, dynamic changes to an item affect all the other items in the same table. Finally, the data processing may be slow or else it requires expensive hardware to achieve high performance.

Some pros of non-relational databases include its flexible data model, it is not confined to a structured group. Rapid adaption to changing requirements leads you to perform functions that allow for greater flexibility. Dynamic changes to an item do not affect the other items in the table. This means the data and analysis can be more dynamic and allow for more variant inputs. Another pro includes the storage of huge amounts of data with little structure which has extremely high performance. Some cons of non-relational databases include its low reliability and manual query language. As a result, non-relational databases may require more programming knowledge than the alternative, meaning your team may have to learn new types of query languages. As well as these cons, it is also difficult to verify data integrity and consistency.

If I was to answer which method is better, it would most certainly depend on the context, what kind of information I am storing and how it might evolve. I will start by giving my impressions as to what context the relational database is a better option. When ACID properties are essential, relational database are the appropriate choice. ACID properties include atomicity, consistency, isolation, and durability. For organisations that need to store structured, predictable data, a relational database is the best option. It offers a level of maturity and widespread support that remains unrivalled by current NoSQL alternatives.

I will now be discussing in which contexts I believe non-relational databases are a better choice. I am of the opinion that the bigger the data set, and the less reliable you need the database to be, the more likely a non-relational database is a better fit. As well as this, the size of resources you can devote to the setup and maintenance of the database plays a huge factor. If the team is smaller, non-relational databases are a better choice as it takes less time to manage. If BASE properties are necessary, non-relational databases are the appropriate choice. BASE properties include basically available, soft state, and eventually consistent.

However, after discussing both options and giving my impression as to why each one is better in certain contexts, these aren't our only options. There is another solution which contains a combination of both technologies where relational and NoSQL databases can be configured together and cooperate without barriers. As discovered in the article "Comparison between relational and NoSQL databases" Kosovare Sahatqija states that they "can combine the benefits of each technology solution to create a more effective system" (Sahatqija et al., 2018). There are certain scenarios for which it would be beneficial to either add NoSQL databases to an existing relational database or vice versa. These hybrid databases exist and are a type of real-time database system that uses both on-disk and in-memory data storage by deciding which is most suitable for a given task. The benefit this creates for the user is flexibility which provides a balance that can then be struck between performance, cost, and persistence.

Exercise 2 (Hadoop/Spark):

To begin this report, I will start by giving a brief outline of both the Spark and Hadoop technologies. Apache Spark is a unified computing that is described as fast, flexible and developer friendly. This leading platform is best suited for stream processing, extensive amounts of SQL, machine learning and batch processing. This processing framework has the flexibility desired by many users to rapidly perform processing tasks on massive amounts of data. As well as this, it has capabilities to distribute data processing tasks across multiple computers, either on its own or in conjunction with other computing tools. It is one of the most actively developed open-source engines, which makes it highly appealing to companies involved in Big Data. Spark supports multiple widely used programming languages such as Python, Java, Scala, and R. Libraries are included for a diverse range of tasks starting from SQL to streaming and machine learning and has the flexibility to run anywhere from a laptop to a cluster of thousands of servers. The effortlessness and scalability of this system make it so appealing to perform Big Data processing.

We will now move on to look at Hadoop. Hadoop is similar to Spark in that it is a set of opensource software programs and procedures that are used as the "backbone" of big data operations. It is commonly used for distributed storage and processing of huge amounts of data on various computer clusters. Hadoop it most widely used for storing data and processing on commodity hardly. This refers to inexpensive, off the shelf, readily available hardware as opposed to purpose-built hardware that is designed to carry out specific IT functions. Hadoop has the flexibility to store a huge amount of data of any kind. This flexibility combined with it's gigantic processing power and the capability to handle an immense number of concurrent tasks or jobs are what make Hadoop so appealing to companies. Nested at the heart of Hadoop's functions is its ability to perform this handling and processing in a distributed order. To be able to accomplish this, Hadoop's architecture was influenced by Google File System (GFS) and Google MapReduce. Hadoop's flexibility that we alluded to earlier allows companies to alter their systems as their goals and needs change truly accentuating this flexibility, all while using cheap inexpensive materials.

I have described a brief overview of both the Spark and Hadoop technologies; I will now attempt to get into the nitty-gritty of the technical differences between these technologies. We are going to look at Hadoop's technologies under the headings of HDFS, YARN, MapReduce and Hadoop Common.

HDFS – This is one of the major components of Hadoop that was modelled based on the file system developed by Google. It manages the actions including storing, distributing and access data. This can be done across many different servers and is able handle structured and unstructured data.

YARN – YARN is short for, Yet Another Resource Negotiator. YARN is one of Hadoop's main components also as it is important for allotting the resources of the system to individual applicating that run on a Hadoop cluster. As well as this, it is essential for the scheduling of various jobs that are executed on distinct cluster nodes and allocates compute resources such as CPU and memory to application.

MapReduce – MapReduce is the fixed processing tool that is used to run applications on a large scale within the cluster nodes. It is responsible for the splitting and running of larger computations into more compact ones that can be dispersed across distinct cluster nodes and ultimately within the processing jobs.

Hadoop Common – Hadoop Common refers to the collation of common libraries and utilities that assist other Hadoop modules.

We will now delve into Spark's technologies that are its key components under the following headings: Spark Core, Spark SQL, Spark Streaming and Structured Streaming and MLlib.

Spark Core – Spark Core is the basis of the whole software; it refers to the delivery engine that allows job schedules and basic operations to take place.

Spark SQL – Spark SQL is a module that facilitates the end user to optimize the processing of data by carrying out SQL statements. Alternatively, it can access the SQL execution engine through Spark's dataset API.

Spark Streaming and Structured Streaming – These are modules that include the ability to stream process. Streaming in Spark acquires the data from various sources which include HDFS, Kafka and Kinesis for example. They then divide this into smaller, micro-batches in order to recognise it as a continuous stream. However, on the other

hand, Structured Streaming is a far newer concept that is built upon the basis of Spark SQL and intended to decrease latency and streamline programming.

MLlib – MLlib consists of a machine learning library that is built into Spark. It comprises of ML algorithms and tools for feature selection and creating ML pipelines.

We have so far given an overview of the two technologies and had a more in-depth look at these technologies by comparing their technical differences. I will now discuss in what circumstances I would choose Spark over Hadoop and vice versa. To start, I will outline Hadoop's limitations then discuss the solution proposed by Spark to combat these limitations.

I/O Bound operations: Limitation – Due to the fact that Hadoop relies on local disk storage, each execution that is carried out incurs an I/O overhead. The potential to harmonise concurrent I/O application formed the foundation of distributed computing within Hadoop. Leveraging this capability efficiently requires an immense level of expertise. As a result, users are deterred from working with extremely large datasets. **Solution –** Unlike Hadoop, Spark can store its information in random access memory, which is up to one thousand times faster than reading from a disk.

MapReduce programming (MR) Model: Limitation – Coding efficient MapReduce programs is not straightforward for those not familiar with Java or Hadoop. **Solution –** Unlike Hadoop, Spark allows the user to write and carry out code in a multiple number of languages without forcing a specific language on the user.

Non-MR Use Cases: Limitation – Even simple tasks such as filters and joins are difficult to perform as these simple actions would have to be expressed in terms of a MapReduce program. For example, a join would have to adopt key-value pairs if it was across two files. **Solution –** Spark SQL, Spark Streaming, and other components of Spark which we have looked at earlier allows the users to perform these common tasks such as SQL joins, aggregations etc.

Programming APIs: Limitation – Developers working in other popular languages such as Python, R and Scala had little to no interest for implementing their solutions in Hadoop as the central programming language is Java. **Solution –** There are commonly used APIs for Python, R and Scala. For example, SparkR is a package that permits R programmers have direct access to Spark data. This enables Spark to become immediately available to a much wider community.

I have outlined the reasons and circumstances that I would choose Spark over Hadoop. We will now move on to look at the reason that I would choose Hadoop over Spark. Using the same method above, I will firstly outline the limitations of Spark and the solutions proposed by Hadoop.

Cost: Limitation – Spark has a higher running cost as a result of relying on in-memory computing for real-time data processing. This requires it to use high quantities of RAM to spin up nodes. **Solution -** Hadoop runs at a lower cost since it relies on any disk storage type for data processing. If the user or company has a limited budget, this is a circumstance where building data analysis infrastructure in Hadoop may be a better option.

Scalability: Limitation – Spark relies on the fault tolerant HDFS for large volumes of data. **Solution -** Hadoop quickly scales to accommodate the demand via Hadoop Distributed File System (HDFS)

Security: Limitation - Spark is less advanced when compared with MapReduce. For example, the security settings within Spark are automatically set to "off" by default, which can leave you vulnerable to attack. **Solution -** Hadoop MapReduce can enjoy all the Hadoop security benefits and integrate with Hadoop security projects, like Knox Gateway and Apache Sentry.

In conclusion, Hadoop and Spark don't actually compete with one another, instead, they complement each other quite well. Spark cannot completely replace Hadoop as Hadoop brings huge datasets under control by commodity systems. Whereas Spark provides real-time, in-memory processing for those data sets that require it. When we combine, Apache Spark's ability to achieve rapid processing power, advanced data and a variety of integration support with Hadoop's extremely low-cost application on commodity hardware, it gives the best results. Hadoop compliments Apache Spark capabilities.

Exercise 3 (Paper review):

The paper that I have chosen to write my review on is “Yesquel: scalable SQL storage for Web applications” by Marcos K. Aguilera, Joshua B. Leners and Michael Walfish. To begin, I will discuss what the question/challenge is that this paper addresses then follow it up by explaining what I believe the motivation for this paper is. The paper introduces the challenges associated with both SQL and NoSQL systems, illustrating that SQL systems are costly to scale and not suitable for hierarchical data storage. Whereas NoSQL systems scale well but drop many of the consistent features that we see in SQL systems such as joins, secondary indexes and transactions, making them unable to perform basic queries that we see in SQL systems. Another challenge highlighted is the emergence of large web applications and the inability of ordinary SQL systems to deal with this this increase, causing bottlenecking. As well as this, three further challenges associated with NoSQL were presented, first being that it shifts the complexity to the application. The second is due to the excess number of NoSQL systems, users can have a difficult time picking the most suitable feature that is required for the application in advance. The last issue highlighted with NoSQL is that each system has its own particular interface which, in turn creates a “lock in” problem, making it very difficult for developers to switch to another NoSQL system later on. What the authors are trying to get across here is that there are challenges associated with both SQL and NoSQL systems. As a result, I believe the motivation behind this research is to attempt to find a solution to these systems with obvious limitations for web applications specifically. This is where Marcos, Joshua and Michael go on to describe Yesquel which has a new architecture and distributed data structure called YDBT that might be a solution. In order to find out, they evaluate Yesquel, testing it against common NoSQL systems such as Redis and popular SQL systems such as MySQL.

This was certainly not the first paper to attempt to scale SQL systems. There have been various pieces of related work outlined in this paper. The previous paper “Distributed sql query processing using key-value storage system” by Marcos K. Aguilera, Joshua Leners and Michael Walfish further elaborate on this architecture and also test its performance. Limitations from other related work include the fact that these solutions are targeted towards other applications such as data analytics and online transaction processing as opposed to directly serving web applications. Database replication is a widely used and researched, however there are some limitations to this method also. Replication can scale the reads but is not as effective to scale the writes. Another well researched area is that of distributed database systems. Typically, there are two architectures, “shared nothing” and “shared disk”. Shared nothing need to be partitioned carefully in order to be scaled, which requires a knowledgeable database administrator. Whereas shared disk systems need to implement distributed locking and caching protocols in order to scale. The limitation we can see with both of these methods is that there can be a scalability bottleneck. Discussed in this paper is the article “MoSQL, A Relational Database Using NoSQL Technology” by Alexander Tomic which introduces MoSQL, a SQL system on top of a NoSQL storage system, similar to Yesquel. However, the limitation identified with MoSQL is its lack of optimization, where it is reported that it is x3 slower than MySQL. This is an unacceptable performance hit and Yesquel’s strives to aim for higher goals.

The solution proposed to solve the challenges regarding SQL and NoSQL systems for web applications include the introduction of Yesquel, which through evaluation and testing has performed as well as Redis, a popular NoSQL system and better than MySQL, a popular SQL database in terms of handling queries at scale. The problem with SQL systems is that they cannot perform as well as NoSQL systems however the problem with NoSQL systems is that they cannot perform basic SQL queries. This paper attempts to introduce Yesquel as a solution as it has performance related to that of NoSQL systems such as Redis while also being able to handle SQL queries at scale. We will

now delve into how they went about testing Yesquel. Performance testing was carried out alongside Redis, a popular NoSQL system used in Twitter, Pinterest, Instagram, StackOverflow Craigslist etc., MySQL, a popular SQL System and MySQL-NDB, a distributed database system, for the purposes of comparison. Each system was configured to operate in memory, Redis, for example is done by disabling persistence, in MySQL we use memory storage engine and for MySQL-NDB we enable the diskless option. All these are carried out to ensure consistency in the testing. All tasks carried out assume a data set with one million key-value pairs with 64 bit key and 100 byte values. Three operations are performed on each of the systems, which are Read, Update and Scan. To implement this workload in Redis, we invoke these operations using C which is the official interface. However, there is no scan operation so an index must be manually pre-populated over all of the keys. For this case, the Scan is more of an iteration. However, for Yesquel, MySQL and MySQL-NDB, implementing the workload operations are far more straight forward and the following SQL statements can be implemented.

$\text{read}(k)$	SELECT value FROM table WHERE key=k
$\text{update}(k)$	UPDATE table SET value=f(value) WHERE key=k
$\text{scan}(k,n)$	SELECT value FROM table WHERE key $\geq k$ LIMIT n

When testing this workload on each of these systems, two clients are assigned to a server, each running thirty-two threads within a closed loop. The testing was run for two minutes and ultimately removed the first and last fifteen seconds of results. Testing was carried out over one second intervals where standard deviation and averages were both recorded for the results. In terms of latency, histograms were created in buckets of 0.1ms granularity and the median and ninety-ninth percentile values were recorded. To analyse resource consumption results, memory was measured using top, conversely network bandwidth was recorded using ifstat. Each of these were recorded with one-second granularity where the averages were computed across one-second periods. To evaluate network bandwidth, inbound and outbound bandwidth were added and ultimately divided by the operation carried out to obtain bytes for each operation. To investigate CPU consumption, the starting and ending CPU time were subtracted and again, divided for each operation to acquire the CPU time per operation. Finally, this paper is aimed to investigate how Yesquel deals with web applications. As a result, testing is carried out against a real web application. Wikipedia was chosen due to the fact that it's underlying system, MediaWiki uses SQL. Testing was carried out using one server to load 190K Wikipedia pages and then ultimately scaling to 48 servers. To summarise the testing, the same workloads of Read, Update and Scan were evaluated across four systems including Yesquel, Redis, MySQL and MySQL-NDB. The items tested on included the performance, the latency, resource consumption, network bandwidth and CPU consumption. Finally, testing was carried out on a real web application which was Wikipedia. Tests were completed with one server and then multiple (48) servers to investigate scalability.

We have discussed the testing in depth, we will now take a look at the results obtained from this testing in the paper. The results achieved through evaluation and testing are that Yesquel performed nearly as well as Redis, its throughput is at maximum 18% lower for each workload. This can be explained by the fact that Redis uses hash table to lookup keys and retains a single version of the data. On the other hand, MySQL performs worse than Yesquel for two major reasons. Firstly, it deals with all SQL requests at the main, central server. Secondly, the SQL processing is heavier than Yesquel's processing as it uses SQLite. Similarly, MySQL-NDB performs worse than Yesquel due to the same heavier processor and also for every read or update operation it must incur multiple round

trips between client and storage servers, whereas Yesquel only incurs one or two trips for the same operation. If we look at the results for the remaining tests carried out, Yesquel appears to use more network bandwidth than either Redis or MySQL. As well as this, Yesquel exhausts more memory than all the other systems and Yesquel also expends more client CPU than either Redis or MySQL. This highlights that Yesquel tends to expend a lot more resources to carry out operations. The authors attempted to scale the tests by adding more servers (48) and more clients (96). MySQL was excluded due to the fact that it cannot run with more than one server. Similarly, MySQL-NDB has an internal limit of 32 servers, so the results were recorded up to this point. Unsurprisingly, the systems scaled linearly across the board. We will move on to examine the results of Yesquel on our real web application, Wikipedia. Using a single server, Yesquel performs roughly two times better than MySQL and latency is also half that of MySQL. However, Yesquel consumes more memory, consumes more bandwidth, and consumes more client CPU than MySQL. When scaled to 48 servers, Yesquel performs 41 times better, representing near linear scalability. Latency is also kept low with a median of 41ms and a 99^{th} percentile of 69ms. These results combine to illustrate to us that Yesquel can be scaled to suit web applications.

To conclude, I do like the attempt at trying to offer all the features of SQL (including the ability to manipulate data based on structured query language) without sacrificing the performance and scalability of NoSQL. I touched on a similar implementation in exercise 1 about a hybrid approach to both a SQL and NoSQL database and this paper attempts to meet a similar goal. What I also like about this paper is that the authors highlighted and proved that Yesquel is not optimised for data analytics or online transaction processing but instead proved its purpose for web applications. In terms of limitations and what more I would have liked to see in this paper, I would have liked to see Yesquel tested against other hybrid models such as MoSQL that was mentioned in this paper or Altibase (hybrid database that permits extraordinary flexibility and vigorous functionalities) instead of the standard SQL and NoSQL database systems.

Bibliography:

Exercise 1:

K. Sahatqija, J. Ajdari, X. Zenuni, B. Raufi and F. Ismaili, "Comparison between relational and NOSQL databases," 2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2018, pp. 0216-0221, doi: 10.23919/MIPRO.2018.8400041.

Lee, K. (2020). Database Management — NoSQL vs. SQL (or MongoDB vs. MySQL). Medium. Retrieved from https://medium.com/analytics-vidhya/database-management-nosql-vs-sql-or-mongodb-vs-mysql-cfa351caf25a.

Lo Duca, A. (2021). Relational VS Non Relational Databases. Medium. Retrieved from https://towardsdatascience.com/relational-vs-non-relational-databases-f2ac792482e3.

S. Rautmare and D. M. Bhalerao, "MySQL and NoSQL database comparison for IoT application," 2016 IEEE International Conference on Advances in Computer Applications (ICACA), 2016, pp. 235-238, doi: 10.1109/ICACA.2016.7887957.

Exercise 2:

A.Verma, A. H. Mansuri and N. Jain, "Big data management processing with Hadoop MapReduce and spark technology: A comparison," *2016 Symposium on Colossal Data Analysis and Networking (CDAN)*, 2016, pp. 1-4, doi: 10.1109/CDAN.2016.7570891.

Hadoop vs. Spark: What's the Difference?. (2021). Retrieved from https://www.ibm.com/cloud/blog/hadoop-vs-spark

Kalron, A. (2020). How do Hadoop and Spark Stack Up?. Retrieved from https://logz.io/blog/hadoop-vs-spark/

Kovachev, D. (2019). A Beginner's Guide to Apache Spark. Retrieved from https://towardsdatascience.com/a-beginners-guide-to-apache-spark-ff301cb4cd92

Marr, B. (2021). What is Hadoop?. Retrieved from https://bernardmarr.com/what-is-hadoop/

Pointer, I. (2020). What is Apache Spark? The big data platform that crushed Hadoop. Retrieved from https://www.infoworld.com/article/3236869/what-is-apache-spark-the-big-data-platform-that-crushed-hadoop.html

Sinha, S. (2019). Spark vs Hadoop: Which is the Best Big Data Framework? [Blog]. Retrieved from https://www.edureka.co/blog/apache-spark-vs-hadoop-mapreduce

Tobin, D. (2021). Spark vs Hadoop MapReduce. Retrieved from https://www.xplenty.com/blog/apache-spark-vs-hadoop-mapreduce/

Why choose Apache Spark over Hadoop for your Big Data project?. (2018). [Blog]. Retrieved from https://www.datasciencecentral.com/profiles/blogs/why-choose-apache-spark-over-hadoop-for-your-big-data-project

Exercise 3:

AGUILERA, M. K., LENERS, J. B., AND WALFISH, M.
Distributed SQL query processing using key-value storage system, Dec. 2012. United States Patent Application

20140172898, filed 13 December 2012.

Marcos K. Aguilera, Joshua B. Leners, and Michael Walfish. 2015. Yesquel: scalable sql storage for web applications. In Proceedings of the 25th Symposium on Operating Systems Principles (SOSP '15). Association for Computing Machinery, New York, NY, USA, 245–262. DOI:https://doi.org/10.1145/2815400.2815413

TOMIC, A. MoSQL, A Relational Database Using NoSQL Technology. PhD thesis, Faculty of Informatics, University of Lugano, 2011.