COMP 1835
Graph and Modern Databases.

Student ID - 001002629

1th April 2021

Contents

List of Figures

1 Part 1: Peer Review

1.1 Summary

Reviewing a paper titled "NoSQL Database Vs Relational Database" the main structure of the paper is good, with clearly labelled headings which makes navigating through easy to read. However there is an unclear and very loosely defined problem that this paper is investigating. In doing so leaves this paper extremely broad and with this in mind, this paper has no novel concepts to it and is a purely confirmatory in nature.

The literature review starts with an investigation into write performance between these technologies,t he author implies that an experiment conducted by Lourenço, Cabral, Bernardino and Vieira found that the best write performance was found to be from Cassandra, but there is no mention of the specifics of this experiment such as the size of data used, goes on to state that MongoDB has low throughput performance yet *"MongoDB is capable of providing high throughput, but mainly when it is used as a single server instance."*(Lourenço et al. 2015b) Which directly contradicts what this author is trying to convey to the reader. MongoDB is described to have adequate performance when compared to MsSQL with no context to what metrics the reader is referring to, where as *"when comparing SQL to MongoDB, MongoDB has better runtime performance for inserts, updates and simple queries. SQL performed better when updating and querying non-key attributes, as well as for aggregate queries."*(Parker, Poe, and Vrbsky 2013) There are also statements within the papers cited by the author that *"the specific nature of our records might have an influence in the tests"* (Lourenço et al. 2015a) and other limitations that are noted in this paper include hardware used such as not having Solid State Drives and the small cluster size used along with noting that the Operating System used was sub-optimal for NoSQL databases and that future work that could provide these experiments on more optimal hardware for NoSQL databases needs to be conducted. This is where Parker, Poe, and Vrbsky 2013 have different results as they have used Solid State Drives to increase the read/write speeds and running both databases on a single machine. The detail in which the results are presented show that using *"the time for MongoDB to update Users by their First Name ranges from 3 times slower than SQL for Test Case 2, to more than 5 times slower for MongoDB for Test Case 3"* while, *"SQL Server is 13 times slower than MongoDB for Test Case 4 when updating by Department ID."* (Parker, Poe,and Vrbsky 2013)

While being critical of Nayak, Poriya, and Poojary 2013 is justified in this case as the title of the paper alludes comparisons between these competing technologies yet concludes with *"This paper describes the pros and cons of NOSQL databases. This paper also describes the advantages and disadvantages of each of the data stores and cases when a particular data stored can be used"* (Nayak, Poriya, and Poojary 2013) With little comparison actually being described by in this article, only a very short very generalised list of advantages and disadvantages are discussed between the two technologies.

Concluding the related works is open to ambiguous interpretation, stating that *"most of the research articles and journals were based on experimenting with the performance of NoSQL and relational databases and comparing them againt each database to see which would perform better"*. As there was no mention of NoSQL's ability to handle semi and unstructured data or the volume of data that each database is designed for even though this this mentioned in the abstract of Parker, Poe, and Vrbsky 2013, furthermore this abstract goes on to describe the performance is evaluated on modest, not large datasets.

The author tries to create a comparison between NoSQL and Relational Databases but misses out on comparisons of features that each technology brings, however the author does includes security aspects that each model deals with as for instance Kunda and Phiri 2017 state NoSQL databases handle security by implementation in middleware while relational databases implement authentication, authorization, encryption, integrity and auditing. These are key components that need to be addressed as security of data should be integral to business decisions are made.

For features that NoSQL has include flexibility, which is probably its greatest strength, and the different models that fall under the umbrella of NoSQL databases and the ability to consume different types of data. For instance Apache Cassandra is described to be *"a leading transactional, scalable, and highly-available distributed database."*(Chebotko, Kashlev, and Lu 2015) which is column orientated also distributed work loads across multiple nodes which can be made up of commodity hardware making it economical when compared to scaling vertically with SQL databases.

1.2 Major Issues

This paper describes BASE as *"Basically Available:Guaranteed Availability"* which the writer attributes to Mohamed, Altrafi, and Ismail 2014, where in reality Mohamed, Altrafi, and Ismail 2014 describe BASE as *"Basically available, Soft state, Eventual consistency"*. This is a major error and shows a lack of understanding in the properties of NoSQL databases, or a lack of reading completed by the author as they would have found the BASE acronym was coined by Eric Brewer who was also instrumental with CAP Theorem (Brewer 2000) which itself a acronym for Consistency, Availability and tolerance to network Partitions. This theory goes on to state that only two of these three states in a shared data environment can be achieved at any one time. With Consistency being data across the server, Availability being access to data being permanent and tolerance to network Partitions referring to failures with hardware or network flow.

There is only a single paper referenced for section three, but it is cited multiple times, nine times to be exact. So the author has paraphrased an entire section of work rather then look for multiple reliable sources to form a well rounded argument while making comparisons between NoSQL and relational databases. As this paper is titled "NoSQL Database Vs Relational Database" and this section is where the comparison takes place, using a single source

for the crux of the argument displays a lack of critical thinking and can lead an individual to an incorrect conclusion about a subject if the resource that they have used to base work on is incorrect in its self. From this the author has stated that NoSQL does not have schema while comparing data models, the correct comparison would be that NoSQL does not have a schema that is ridged like relational databases. Instead they are schema-less, meaning that the database does not require a schema for data to be stored and conform to this defined schema throughout the life time that it is stored, Meurice and Cleve 2017 describe this as, *"no explicit database schema is declared by developers. "*. Instead NoSQL can use key-value pairs or JSON documents when dealing with data which bring rich information that is not limited by a global ridged schema. Also when listing what data models that NoSQL has, the author mentions key value, graph and document stores but completely excludes column-family models. For example Apache Cassandra would be a great well documented example from the column-family model.

There are only mentions on differences between scalability, but the author does not expand on how this effects the comparison. As vertical scaling for relational databases requires expensive hardware to gain extra performance while NoSQL can run on consumer grade hardware which makes it cost effective when scaling horizontally.

1.3 Minor Issues

While the layout of this report is generally well structured and easy to follow, the language used to convey the authors message makes it hard to follow. There are countless grammatical errors, for instance within the introduction there is a missing space between words in the final paragraph which could have been rectified with the individual proof reading their own work. This also continues with consistency, MongoDB is used as an example throughout this paper, but is also referred to as Mongodb without capitalisation of the last two letters in some instances. Again individual proof reading would have caught these small mistakes and remove the inconsistency.

There is also an instance of initialism with regards to DML without first introducing the term in full. In this case Data Manipulation Language (DML) should have been used first and shown readers what this particular initialism explicitly refers too, then any mentions after this initial introduction consistently used when referring back to their point.

1.4 Presentation and Style

As mentioned within Minor Issues, reading of this paper does not flow in a natural manner which makes digesting the information that the author is trying to convey difficult to ingest. There are also many instances of very short sentences which breaks the flow while reading and when this is combined with grammatical errors, disrupts the flow of this paper. It is also hard to hear the authors own voice and ideas in this paper, possibly due to the compounded small mistakes.

From reading through this paper, it would seem that they are comfortable working with Relational Databases, its concepts and use cases, but are new to the concepts of NoSQL and the use cases and are not comfortable with this new information. The main take from this paper is that the presentation feels rushed and lacks the completeness this it truly deserves.

There is ambiguity in the conclusion of this paper as the author has stated that *"research article were using mostly MS SQL"* for their research, does this mean that the author also found other relational databases that were compared to NoSQL but just did not include them in this paper, or does it mean that these papers did not fit a predisposition held by the author so were not included. As this raises the more questions than are answered in the conclusion, which is not a good way to end a report.

1.5 Reference Comments

Referencing within this paper is mixed with Vancouver style author-number within the reference list and Harvard style author-date used in text. So although individually using either style is fine to use, unless stipulated by criteria, mixing these two different styles in the same paper creates an exposition that this author did not fully understand the differences between the two styles or how to implement these consistently with the application they had chosen to write this report in.

Quoting other peoples work within this paper has not been properly defined with italicised quotation marks while in text, this error could lead people who have read these works to assume that this author is claiming work that is not their own, which could lead to plagiarism accusations. There are references within text, but it is confusing what exactly is referenced due to the prior point regarding quotations.

The references themselves are missing vital information that would make them complete, for instance none of the items in this list have a date when they were accessed, missing Digital Object Identifier (DOI) numbers for the papers that do have them which makes it difficult to cross reference what the author of this paper has written with bodies of work that they have read.

1.6 Recommendation

Recommendations for this paper would be a rejection, based on the incompleteness of this body of work and issues with referencing. Reviewing work on data models to complete this section with four types of NoSQL database models and use cases for each would also help bring this paper up to standard. Use of more than a single source when it comes to the comparison of each technology which will bring a well rounded argument. Possibly implementing a table as a quick overview of each and then expanding on each point made. There are no references to use cases for these differing technologies as each of the different types of NoSQL databases have different uses cases where they are designed for

use. Inclusion of works found with research that makes use of other relational database would also remove the ambiguity found in the conclusion.

1.7 Conclusion

Peer reviewing a paper has been a learning curve, it has not been a case a simply writing constructive criticisms but also these criticisms must be justified with examples to backup the comments. English is not everyone's first language and writing a coherent report on a technical subject is not everyone's strength, but everyone has the ability to make improvements if the willingness is there to do so and if they are shown where their mistakes have been made.

Reference

Brewer, Eric A (2000). "Towards robust distributed systems". In: *PODC*. Vol. 7. 10.1145. Accesses 20 Feb 2021. Portland, OR, pp. 343477–343502.

Chebotko, A., A. Kashlev, and S. Lu (2015). "A Big Data Modeling Methodology for Apache Cassandra". In: *2015 IEEE International Congress on Big Data*. Accesses 20 Feb 2021, pp. 238–245. DOI: 10.1109/BigDataCongress.2015.41.

Kunda, Douglas and Hazael Phiri (2017). "A comparative study of NoSQL and relational database". In: *Zambia ICT Journal* 1.1. Accesses 20 Feb 2021, pp. 1–4.

Lourenço, João et al. (Sept. 2015a). "Comparing NoSQL Databases with a Relational Database: Performance and Space". In: *Services Transactions on Big Data* 2. Accesses 20 Feb 2021, pp. 1–14. DOI: 10.29268/stbd.2015.2.1.1.

Lourenço, João Ricardo et al. (Aug. 2015b). "Choosing the right NoSQL database for the job: a quality attribute evaluation". In: *Journal of Big Data* 2.1. Accesses 20 Feb 2021, p. 18. ISSN: 2196-1115. DOI: 10.1186/s40537-015-0025-0. URL: https://doi.org/10.1186/s40537-015-0025-0.

Meurice, L. and A. Cleve (2017). "Supporting schema evolution in schemaless NoSQL data stores". In: *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. Accesses 24 Feb 2021, pp. 457–461. DOI: 10.1109/SANER.2017.7884653.

Mohamed, Mohamed, Obay Altrafi, and Mohammed Ismail (May 2014). "Relational Vs. NoSQL databases: A survey". In: *International Journal of Computer and Information Technology (IJCIT)* 03. Accesses 15 Feb 2021, p. 598.

Nayak, A., A. Poriya, and Dikshay Poojary (Jan. 2013). "Article: Type of nosql databases and its comparison with relational databases". In: *International Journal of Applied Information Systems* 5. Accesses 20 Feb 2021, pp. 16–19.

Parker, Zachary, Scott Poe, and Susan V. Vrbsky (2013). "Comparing NoSQL MongoDB to an SQL DB". In: *Proceedings of the 51st ACM Southeast Conference*. ACMSE '13. Accesses 20 Feb 2021. Savannah, Georgia: Association for Computing Machinery. ISBN: 9781450319010. DOI: 10.1145/2498328.2500047. URL: https://doi.org/10.1145/2498328.2500047.

Bibliography

Stiller-Reeve, Mathew (Oct. 2018a). *How to write a thorough peer review*. Accessed: 28 Jan 2021. URL: https://www.nature.com/articles/d41586-018-06991-0.

2 Part 2: Implementation

2.1 Assignment 1 JSON

JavaScript Object Notation (JSON) is used to serve as a simple and readable way for representing structured data that uses JavaScript as syntax. Commonly used as a template to transfer data from point to point as to make sure that information that is transferred is correct and in the right format. Using key value pairs to represent data makes this type of document more human readable and according to Nurseitov et al. (2009) *"Results indicate that JSON is faster and uses fewer resources than its XML counterpart"*, although these experiments were conducted in 2009 they still are of interest due to the performance comparison.

For this implementation information from the Formula 1 2020 season has been converted into a JSON format, to be more specific this JSON document contains information from the first round in the 2020 season held at the Red Bull Ring hosted in Spielberg, Austria. This was used as an example to show JSON as there is a large quantity of data that is produced from an F1 race weekend. The structure for this document has a parent JSON object named grandPrixData where all other items are nested. Properties for this object include *year* and *round*, which both are of type integer to signify the year and what round which the event was held. String properties include *location*, *date* and *circuit* to get more geographically localised information where this event was held. The property *note* is an array of strings which add extra information about incidents that happened during this race. There are 20 drivers that compete in the Formula 1 World Championship, but they all have the same properties when it concerns data for this particular case. As such we can see drivers property as a more complex array of objects. For a single driver object in this implementation, each one has a *name* which is their name in string form, *number* which is an integer and corresponds to the number that that particular driver has on their car, *team* which is who they are driving for and is a string. There are three objects nested within the driver object that are linked to qualifying, fastest lap and race result data.

Looking at the qualifying object first there is an array that holds up to three strings that represent fastest lap times as during the qualifying session there are three sessions often referred to as Q1, Q2 and Q3. During Q1 all 20 drivers compete with the slowest five unable to reach Q2 and are placed 16^{th} to 20^th based on their times. Q2 then has 15 drivers with the slowest 5 taking 11^th to 15^th and Q3 the remaining 10 drivers then compete for the top ten starting positions. The $qualifying$ object as a property *position* which is an integer and signifies where that particular driver has started for the event.

Fastest lap object has properties $lapNumber$ which is an integer referring to what lap that particular driver set their own fastest lap on, *time* property is a string that contains the time this driver set and finally *averageSpeed* that is a number as it is accurate to three decimals is the the average speed that the driver managed during their fastest lap.

Race results objects have four properties which are *position* that is an integer, *time/retired* that is a string that has either the time that the driver set for the total race or if they did not manage to finish this particular event a DNF for their time. *Points* property is dependant on where the driver finished, with the top ten drivers receiving points ranging from 25 to 1 point with any drivers finishing 11^th or below receiving no points. *Laps* property is also an integer and refers to how many laps the driver completed during the event regardless of if they finished the race or not.

Figure 1: Screenshot of JSON validation

Figure 1 shows the JSON document being validated against the schema for this document has been proved valid.

2.2 Assignment 2 - Redis

Redis is a open source database that uses in memory data structure rather than reading and writing to physical disks, this give it the advantage of speed when but is restricted by the physical size of memory a server has. This makes Redis extremely useful for gamification of apps, where score boards are added to applications for uses to see and compare scores.

This particular implementation is used for video game rating that would have real time counts and scores associated with a particular game. There are ten Hashmaps that relate directly to a Video game and its properties with two sorted sets that hold the rating of a game and the total number of users who have rated a game. This particular example would be used to save the server having to read and write countless times to a physcial hard drive while allowing a application to update in real time and show user ratings and how many users have rated.

```
127.0.0.1:6379> hvals game8
 1) "Forza Horizon 4"
 2) "Racing"
 3) "Playground Studios"
 4) "28 Sept 2018"
 5) "Single Player"
 6) "Multiplayer 2-12"
 7) "Xbox Live cross-platform multiplayer"
 8) "Online Co-Op 2-6"
 9) "HDR10"
10) "Xbox"
11) "PC"
12) "17.99"
```

Figure 2: Redis Hash Map - getting all values

Figure 2 displays all the values contained in a hash map, which can be seen as an object from a programming perspective, as hash maps can contain string, integer values and lists within them.

```
127.0.0.1:6379> hget game2 name
(nil)
127.0.0.1:6379> hset game2 name "Cyberpunk 2077"
(integer) 1
127.0.0.1:6379> hget game2 name
"Cyberpunk 2077"
127.0.0.1:6379>
```

Figure 3: Redis Hash Map - Changing a field value

Figure 3 is how to set a value in a hash map, from line one we can see that the return in nil which tells the user that it does not exist, the second line is inserting a string value into the hash map, breaking down the second command, HSET refers to setting hash map values, game2 is the hash that is being amended, name is the key and Cyberpunk 2077 is the value that is being inserted. the third line is a confirmation that the change has occurred.

```
127.0.0.1:6379> expire game10 120
(integer) 1
127.0.0.1:6379> TTL game10
(integer) 113
127.0.0.1:6379> persist game10
(integer) 1
127.0.0.1:6379> TTL game10
(integer) -1
127.0.0.1:6379>
```

Figure 4: Redis Expire Commands

Figure 4 uses the EXPIRE command which would be useful to remove a items from memory if they have not been used within the set time limit as unused keys in memory would be a wasteful use of resources. The second command in Figure 4 displays how long in seconds this particular key has left before it is deleted from memory. While the last command PERSIST removes the EXPIRE command so that this key will not be deleted.

```
127.0.0.1:6379> zrange totalRatings 3 6
1) "game4"
2) "game9"
3) "game8"
4) "game2"
```

Figure 5: Redis zrange command

Figure 5 uses zrange which can only be used for sorted sets and returns, in this case, the range of keys that have scores between 3 and 6. This can be used to fetch other keys that have similar scores.

```
127.0.0.1:6379> zscore totalRatings game3
"365"
127.0.0.1:6379> zincrby totalRatings 300 game3
"665"
127.0.0.1:6379>
```

Figure 6: Redis Increment value in sorted list

Figure 6 increments a score by a specified value for keys that are sorted lists. Figure 6 has the score before and after the increment of 300 is applied to game3.

2.3 Assignment 3 - Cassandra

Cassandra is an open source, NoSQL, wide column database that uses Cassandra Query Language (CQL) for programming. Favouring data replication over multiple nodes to keep reliability high. This implementation has six columns with a complex primary key made up of the game_name and platform_name column due to video games being released on multiple platforms, sometime with different prices and different release dates depending one global region.

Game_Name and Platform_Name are both varchar data type and not to restrict the characters that can be entered for names of games or platforms, developer column is a text data type and lets the users know which studio developed that particular game, multiplayer column is a boolean, a simple true or false if the game is multiplayer or not. Price is set as a double data type and release_date is a date data type.

```
cqlsh:comp1835> select * from Game_Platform
           ... where price < 20.00 allow filtering;

 game_name          | platform_name    | developer          | multiplayer | price | release_date
--------------------+------------------+--------------------+-------------+-------+--------------
         Until Dawn |               pc | Supermassive Games |       False |  9.99 |   2015-08-28
         Until Dawn |              ps4 | Supermassive Games |       False | 15.99 |   2015-08-28
         Bloodborne |              ps4 |   FromSoftware Inc |       False | 19.99 |   2015-03-24
          minecraft |               pc |      mojang studios |        True | 18.76 |   2011-11-18
          minecraft |              ps4 |      mojang studios |        True | 19.99 |   2011-11-18
  League of Legends |               pc |          Riot Games |        True |     0 |   2009-08-27
              cs:go |               pc |  Valve Corporation |        True | 14.99 |   2012-08-12
            Gears 5 |               pc |       The Coalition |        True | 19.99 |   2019-09-06
            Gears 5 |         xbox one |       The Coalition |        True | 13.49 |   2019-09-06
            Gears 5 | xbox series x|s |       The Coalition |        True | 13.49 |   2019-09-06
   Sunset Overdrive |               pc |    Insomniac Games |        True | 14.99 |   2018-12-16
   Sunset Overdrive |         xbox one |    Insomniac Games |        True | 11.99 |   2014-08-28
 Gran Turismo Sport |              ps4 |  polyphony digital |        True | 15.99 |   2017-10-17
            F1 2020 |               pc |         Codemasters |        True | 13.49 |   2020-07-10
            F1 2020 |           stadia |         Codemasters |        True |  9.99 |   2020-07-10
 Grand Theft Auto V |               pc |     Rockstar Games |        True | 12.58 |   2013-09-17
 Grand Theft Auto V |              ps4 |     Rockstar Games |        True | 17.13 |   2013-09-17

(17 rows)
```

Figure 7: Cassandra Select All From Game_Platform Table

Figure 7 selects everything from the Game_Platform table but also has a where clause in the price column to be under 20.00.

```
cqlsh:comp1835> select * from Game_Platform
           ... where Platform_Name = 'xbox one' Allow Filtering;

 game_name          | platform_name | developer       | multiplayer | price | release_date
--------------------+---------------+-----------------+-------------+-------+--------------
          minecraft |       xbox one |  mojang studios |        True | 22.99 |   2011-11-18
            Gears 5 |       xbox one |   The Coalition |        True | 13.49 |   2019-09-06
   Sunset Overdrive |       xbox one | Insomniac Games |        True | 11.99 |   2014-08-28
            F1 2020 |       xbox one |     Codemasters |        True | 54.99 |   2020-07-10
 Grand Theft Auto V |       xbox one |  Rockstar Games |        True | 24.99 |   2013-09-17

(5 rows)
```

Figure 8: Cassandra Select All From Platform_Game table where platform name equals xbox one

Filtering the Platform_Game table for only xbox platform games returns five game names.

```
cqlsh:comp1835> select * from game_platform
           ... where game_name = 'Rimworld' allow filtering;

 game_name | platform_name | developer       | multiplayer | price | release_date
-----------+---------------+-----------------+-------------+-------+--------------
  Rimworld |            pc |  Ludeon Studios |       False | 17.48 |   2018-09-17

(1 rows)
cqlsh:comp1835> update game_platform
           ... set price = 26.99
           ... where game_name = 'Rimworld' AND platform_name = 'pc';
cqlsh:comp1835> select * from game_platform where game_name = 'Rimworld' allow filtering;

 game_name | platform_name | developer       | multiplayer | price | release_date
-----------+---------------+-----------------+-------------+-------+--------------
  Rimworld |            pc |  Ludeon Studios |       False | 26.99 |   2018-09-17

(1 rows)
```

Figure 9: Cassandra Update value sequence

Figure 9 returns the properties where the game name is Rimworld, the start of the process the price for this game is 17.48, but by using the SET command and the combined primary key of the game name and platform, this is changed to 26.99.

```
cqlsh:comp1835> select count (*) from Platform_Game
           ... where Platform_Name = 'stadia' allow filtering;

 count
-------
     2

(1 rows)
```

Figure 10: Cassandra Count number of occurrences from Platform_Game table

Figure 10 counts the number of rows that have the Platform_Name stadia, in this case it returns a count of two and could be useful for counting large datasets to see what platform has most titles associated with it.

```
cqlsh:comp1835> select avg(price) as Average_Price
           ... from Game_Platform
           ... Where Platform_Name = 'xbox one' allow filtering;

 average_price
---------------
         25.69

(1 rows)
```

Figure 11: Cassandra Average price of games where Platform_Name equals xbox one

Figure 11 uses the function *avg*() with the column price entered into it and outputs the returned value into a temporary column named average_price. There is also a WHERE clause that limits this average calculation to platform

names that are equal to xbox one, the return value only can be seen in Figure 11 as 25.69.

2.4 Assignment 4 - MongoDB

MongoDB is a document database which allows great flexibility with the data that can store, the implementation is on camp sites around the United Kingdom. This structure has two nested lists that store the campsites facilities, which can differ from site to site, and also what pitches are available at the site. The Id has been specified for each of the twenty documents that are in the collection as an integer, rising incrementally from 1 to 20. The Name key value pair is a string to represent that sites name, price key value pair is a float as pricing for camp sites are not rounded to the nearest pound.

```
> db.comp1835.find({Location:"North England"}).pretty()
{
        "_id" : 1,
        "Name" : "Windermere",
        "Location" : "North England",
        "facilities" : [
                {
                        "toliet" : "Flushing Toliet",
                        "Shower" : "Showers",
                        "wifi" : "Club Site Wi-Fi",
                        "Pets" : "Welcome"
                }
        ],
        "pitches" : [
                {
                        "type1" : "Grass Only",
                        "type2" : "Grass With Electric Hook-up",
                        "type3" : "Hardstanding",
                        "type4" : "Ready Camp tents"
                }
        ]
}
{
        "_id" : 2,
        "Name" : "Bowness on Windermere",
        "Location" : "North England",
        "facilities" : [
                {
                        "toilet" : "Flushing",
                        "Shower" : "Shower",
                        "wifi" : "Club Site WiFi"
                }
        ],
        "pitches" : [
                {
                        "type1" : "Hardstanding"
                }
        ]
}
{ "_id" : 3, "Name" : "ravenglass", "Location" : "North England" }
```

Figure 12: MongoDB Find with location

```
> db.comp1835.find({"facilities.children" : "Childrens play area"}).pretty()
{
        "_id" : 11,
        "Name" : "Walton on Thames",
        "Location" : "Surry",
        "price" : 5.3,
        "facilities" : [
                {
                        "Pets" : "Welcome",
                        "children" : "Childrens play area",
                        "fishing" : "Fishing"
                }
        ],
        "pitches" : [
                {
                        "type1" : "Grass only",
                        "type2" : "Grass With Electric Hook-up"
                }
        ]
}
{
        "_id" : 20,
        "Name" : "Bude",
        "Location" : "Cornwall",
        "price" : 6.65,
        "facilities" : [
                {
                        "children" : "Childrens play area",
                        "Pets" : "Welcome",
                        "toliet" : "Flushing",
                        "shower" : "Showers",
                        "wifi" : "Club site wi-fi",
                        "disabilities" : "Dedicated accessible facilities"
                }
        ],
        "pitches" : [
                {
                        "type1" : "Grass only",
                        "type2" : "Grass With Electric Hook-up",
                        "type3" : "Hardstanding",
                        "type4" : "Ready camp tents"
                }
        ]
}
```

Figure 13: MongoDB query within nested list

```
> db.comp1835.find({$where: function() {return(this.price < 6.00)}} ).pretty()
{
        "_id" : 11,
        "Name" : "Walton on Thames",
        "Location" : "Surry",
        "price" : 5.3,
        "facilities" : [
                {
                        "Pets" : "Welcome",
                        "children" : "Childrens play area",
                        "fishing" : "Fishing"
                }
        ],
        "pitches" : [
                {
                        "type1" : "Grass only",
                        "type2" : "Grass With Electric Hook-up"
                }
        ]
}
{
        "_id" : 13,
        "Name" : "Rhandirmwyn",
        "Location" : "Wales",
        "price" : 5.9,
        "facilities" : [
                {
                        "Pets" : "Welcome",
                        "toliet" : "Flushing",
                        "shower" : "Showers",
                        "wifi" : "Club site wi-fi"
                }
        ],
        "pitches" : [
                {
                        "type1" : "Grass only",
                        "type2" : "Grass With Electric Hook-up",
                        "type3" : "Hardstanding",
                        "type4" : "Ready Camp tents"
                }
        ]
}
```

Figure 14: MongoDB Where query price less then 6.00

2.5 Assignment 5 - Neo4j

Neo4j is a graph database that excels in creating and displaying relationships between nodes that are present. The following graph has been created using data from the first eight chapters of Final Fantasy VII Remake which is a video game with countless interactions between player characters and non player characters.

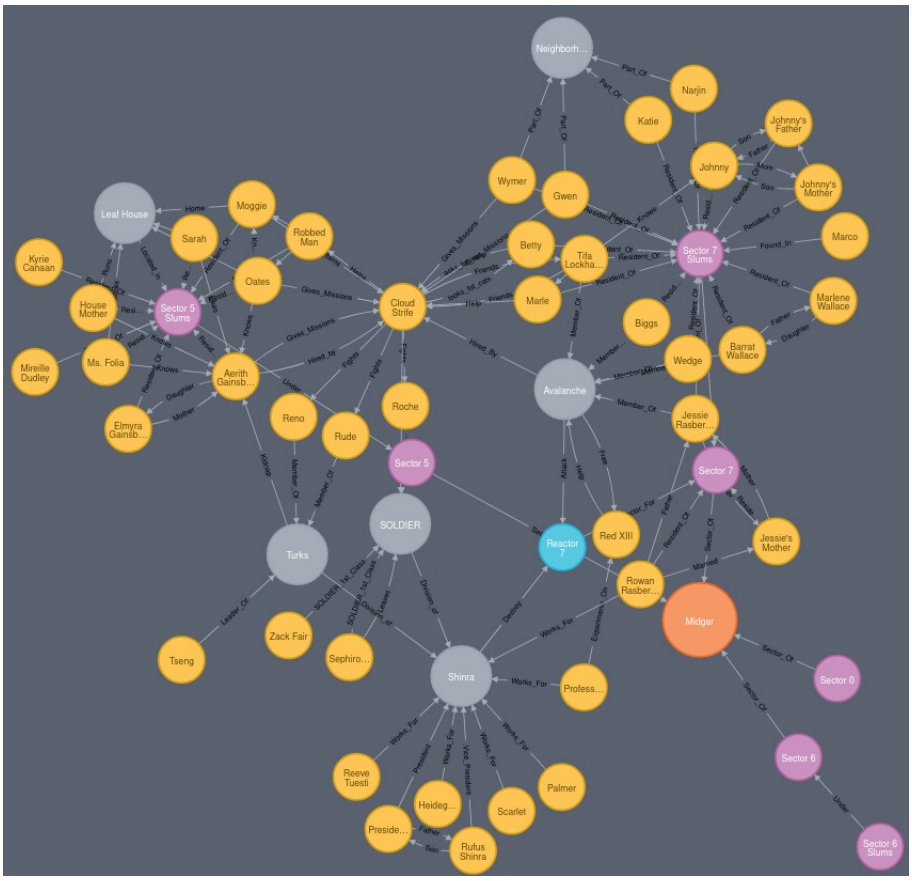

Figure 15: Overview of Neo4j Graph

$$match\ (n)\ return\ n \tag{1}$$

Figure 15 queries the entire graph for all nodes and their relationships. In Figure 15 there are 58 nodes that have been colour coded and resized to make reading the information a easier and 115 relationships that connect the nodes. To distinguish between nodes characters have been coloured yellow, organisation nodes have been made larger and coloured grey, the city of Midgar has been coloured orange with its sectors coloured pink and finally the single reactor coloured blue.

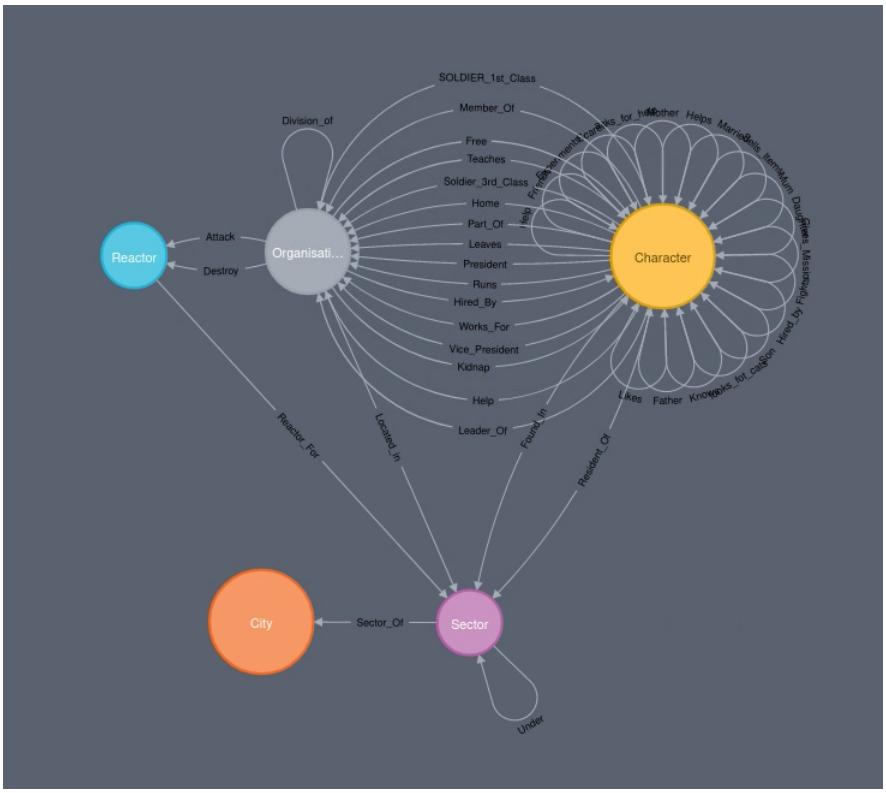

Figure 16: Neo4j Database Visualisation Schema

$$db.schema.visualization \tag{2}$$

Figure 16 is a visualisation of the schema which better shows hows this graph is built, displaying the main node types and all the relationships that are attached to them. There are five distinct nodes that are on display that are used thought out this graph with the character node having many unique relationships between different character nodes.

Figure 17: Neo4j Organisation Node Order By Name

$$\begin{aligned} &match\ (o.Organisation) \\ &return\ o.Name\ ORDER\ BY\ o.Name \end{aligned} \tag{3}$$

Figure 17 is a table of all the organisation names that have been included in the graph and they have been ordered alphabetically, there has not been a limit set for this particular query as there are not enough nodes to warrant limiting this query. If there were a large number of nodes that had been returned, limiting the return could save on resources that are used and improve the performance.

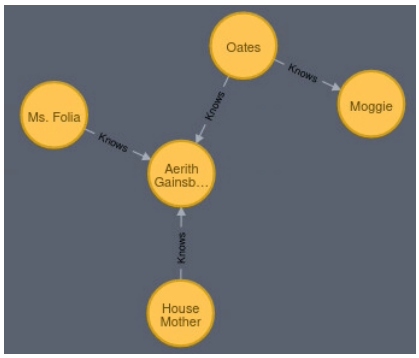

Figure 18: Finding Second Neighbours for a Node by relationship

$$\begin{aligned} &match\ (c) - [: Knows * ..2] - (b) \\ &where\ c.Name\ =\ "Aerith\ Gainsborough" \\ &return\ distinct\ c, b \end{aligned} \tag{4}$$

Figure 18 is a query that displays up to two neighbours away from the node named "Aerith Gainsborough" that have the relationship "knows". This particular query has returned five nodes and four relationships that are within two

neighbours. The restriction of the relationship type could be changed to any other relationship that is directly connected to the node to see how many returns it has. In context to Final Fantasy VII this is not a major surprise that this particular graph has been returned as all these characters live within the same sector as each other which would increase the likely hood of knowing an individual.

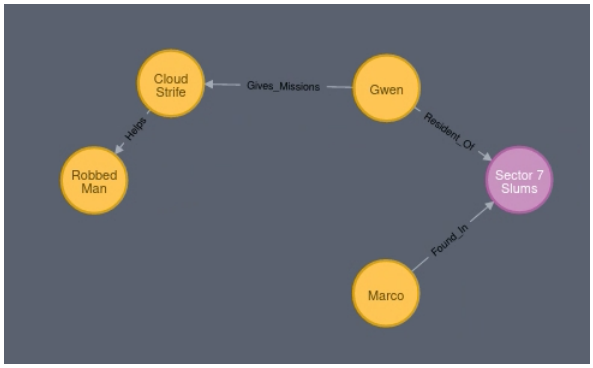

Figure 19: Shortest path between two nodes

$$\begin{aligned} &match\ p\ =\ shortestPath((c1:Character)-[*]-(c2:Character)) \\ &where\ c1.Name\ =\ "Marco"\ AND\ c2.Name = "RobbedMan" \\ &return\ p \end{aligned} \tag{5}$$

Figure 19 shows the shortest path between two nodes, "Robbed Man" and "Marco", this is a surprise as there are only five nodes in total between these two characters that are located in two different districts in the game and are never in the same place at the same time. Breaking down the syntax for the query :Characters specifies the type of node to search for, c1 and c2 represent property name of the nodes that are being used to search between, the relationship between these pair of nodes do not have any restrictions which is denoted by the * located between them.

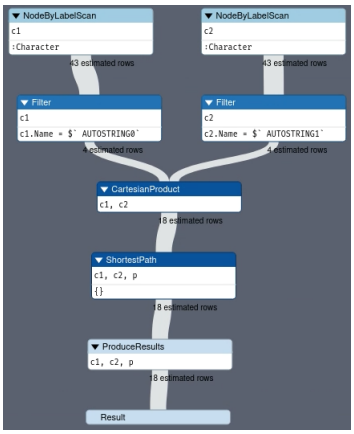

Figure 20: Shortest path between two nodes Profiling

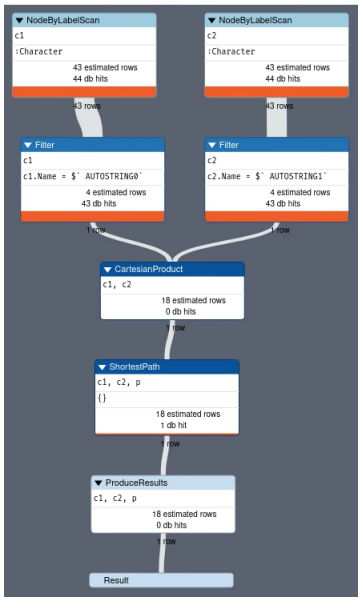

Figure 21: Shortest path between two nodes Profiling With Execution

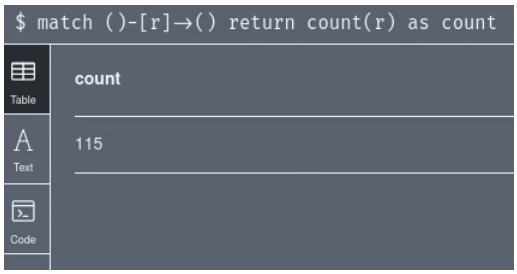

Figure 22: Count of all relationships

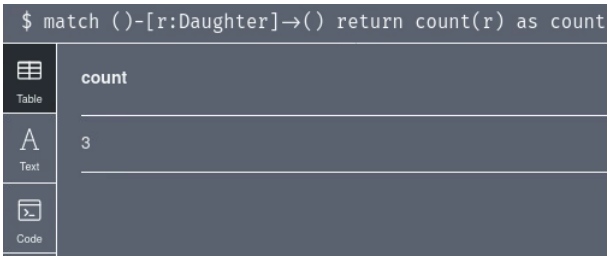

Figure 23: Count of daughter relationships

Figure 24: Count of all relationships connected to Cloud Strife and Nodes

3 Conclusion

This module has been very interesting with a look into how modern databases work, it has been the first time that I have implemented any of these databases and i have particularly enjoyed Neo4j. Even though time to complete this coursework had been grossly underestimated and most of that time was spent planning and executing the graph database as this type of database is like nothing I have implemented before, it is missing twelve to thirteen more character nodes and their relationships with the existing nodes which would make the graph even more complex than it had already become.

The peer review within this module, and I do understand that it is a requirement that we, as individuals learn how to execute a peer review, it seems out of place to have this particular requirement here. Although learning this process will undoubtedly help improve my own reports and documentation.

There is no one to blame for poor use of time but myself with not finishing the fourth task in full due to underestimating the time needed to fully write out complex enough queries that would satisfy the task. Unfortunately I do feel this report shows that most of the time had been spent on Neo4j, but I have enjoyed the challenge graph database presented.

Reference

Championship, Formula One World (July 2020). *Standings*. Accesses 01 March 2021. URL: https://www.formula1.com/en/results.html/2020/races/1045/austria/race-result.html.

Nurseitov, Nurzhan et al. (2009). "Comparison of JSON and XML data interchange formats: a case study." In: *Caine* 9. Accesses 24 Feb 2021, pp. 157–162.