**Big Data - Project 1**

**Shane Cooke – 17400206**

This report contains the details of work done for the COMP30770 project. In this project, I cleaned the reddit\_20212 data set using a variety of different methods, created a MySQL database, populated this database using bash, created a MongoDB database and also populated this database using bash. The code for the project has been submitted alongside this report, its use is described both in the comments of the .sh file and the README.txt.

**Cleaning a Dataset with Bash**

1. In order to drop the specified columns (1, 2, 3, 34) from the file, I used the “cut” function along with the “–d” option, which deletes the specified columns from the input file. I used the original “reddit\_20212.csv” file as input and created a new file called “reddit\_done.csv” for the output of this process. The code I have used for this step can be found below and also in the “remove\_initial.sh” file.

**cut -d**, -f1,2,3,34 **--complement** reddit\_20212.csv > reddit\_done.csv

1. To remove the empty columns from the file, I first created a variable called “no\_of\_rows” to store the number of rows in the input csv. I then used a for loop to iterate through “reddit\_done.csv” and count the occurrences of each variable in each column of the file. If the number of occurrences of a variable is equal to the number of rows, and the value of each variable is equal to null, the index number of the column is added to “array”. I then saved the values in “array” to the “remove” variable and used this variable with the “cut” function and the “-d” option, to remove the empty columns from “reddit\_done.csv”. I then saved the resulting file to “reddit\_done1.csv”. The code for this task is too large to paste below, but it can be found in the “r\_empty.sh”.
2. Some of the columns in the file are classed as uninformative, which means they only contain a single value. This code is very similar to the above code with a few minor alterations. To remove the uninformative variables from the file, I first created a variable “no\_of\_rows” to store the number of rows in the csv. I then used a “for” loop to iterate through the “reddit\_done\_1.csv” file and count the occurrences of each variable in each column of the file. If the number of occurrences of a variable is equal to the number of rows, the index number of the column is added to “array”. I then saved the values in “array” to the “remove” variable and used this variable, along with the “cut” function and the “-d” option, to remove the uninformative columns from “reddit\_done1.csv” and save the result to “reddit\_done2.csv”. The code for this task is too large to paste below, but it can be found in the “remove\_uninformative.sh” file.
3. Two variables in the file called “created\_utc” and “retrieved\_on” contain time in seconds since epoch, which we want to convert to a month. In order to do this, I used the “awk” function to specify which columns I wanted to act on (8, 27) and then used the “strftime” function to convert the date into the specified format. The “%b” option is used to convert the given date into only the month name. I took “reddit\_done2.csv” as input and saved the resulting file to “reddit\_done3.csv”. The code I used to do this can be found below and also in the “fix\_date.sh” file.

**awk -F**, **-v** OFS=**","** **'NR>=2 {$8=strftime("%b",$8);$27=strftime("%b",$27)};1'** reddit\_done2.csv > reddit\_done3.csv

1. In order to count the number of posts made in each month, I began by using the “cut” function and the “-d” option to specify which column I wanted to focus on. I then used the “sed” function to remove the first row, as I didn’t need that row for counting. I then used the “uniq” function with the “-c” option, which counts how many times a line has been repeated in the file. The code I used for this task can be found below and also in the “count\_posts.sh” file.

**echo** **"Posts created: "**

**cut -d","** -f8 reddit\_done3.csv | **sed** **"1 d"** | uniq **-c**

**(6)**

**(a)** In order to, convert all of the letters in the title column to lower-case, I began by isolating column 37 using the “cut” function with the “-d” option and then saved the resulting file to “lower.csv”. Next I used the tr function along with the ‘[:upper:]’ ‘[:lower:]’ syntax to convert all of these letters to lower-case. I then used the “awk” function to paste this cleaned column back into the full csv file. I used the “reddit\_done3.csv” and “lower.csv” files as input and saved the result to the “reddit\_done4.csv” file. I then deleted “lower.csv” as I no longer needed the file. The code for this task can be found below and also in the “tolower.sh” file.

**cut -d','** -f37 <**"reddit\_done3.csv"** | tr **'[:upper:]''[:lower:]'** > lower.csv

**awk** **'BEGIN{FS=",";OFS=","}FNR==NR{a[NR]=$1;next}{$37=a[FNR]}1'** lower.csv reddit\_done3.csv > reddit\_done4.csv

**rm** lower.csv

**(b)** To remove all punctuation from the title column, I again used the “cut” function with the “-d” option, which isolated the specified column (37). I then used the “tr” function with the “-d” option and the ‘[:punct:]’ syntax, which deletes all punctuation from the input file. I saved this cleaned title column as “nopunct.csv”. Next, I used the “awk” function to paste the formatted column back into the original csv file, taking “nopunct.csv” and “reddit\_done4.csv” as input, and using “reddit\_done5.csv” as my output file. I then deleted “nopunct.csv” as I no longer needed the file. The code for this task can be found below and also in the file “delchar.sh”.

**cut -d','** -f37 <**"reddit\_done4.csv"** | tr **-d** **'[:punct:]'** > nopunct.csv

**awk** **'BEGIN{FS=",";OFS=","}FNR==NR{a[NR]=$1;next}{$37=a[FNR]}1'** nopunct.csv reddit\_done4.csv > reddit\_done5.csv

**rm** nopunct.csv

**(c)** In order to remove the stop words from the title column of the csv, I again began by using the “cut” function with the “-d” option to remove column (37) from the “reddit\_done5.csv” file and save it to “stopwords.csv”. I then used a “while” loop to read through every word in the title column and used “sed” with the “-ri” option to remove all occurrences of any stop words. I used “snowball\_expanded.txt” as a dictionary of stop words and took this as input for the “sed” function. I used the “\b” option to ensure that only full strings would be removed from the file. I then used “awk” to paste “stopwords.csv” back into “reddit\_done5.csv” and saved the output to “reddit\_done6.csv”. I then deleted “stopwords.csv” as we no longer have any use for it. The code for this task can be found below and also in the “stopwords.sh” file.

**cut -d','** -f37 <**"reddit\_done5.csv"** > stopwords.csv

while IFS= read **-r** word; do **sed** -ri **"s/( |)\b$word\b//g"** stopwords.csv; done < snowball\_expanded.txt

**awk** **'BEGIN{FS=",";OFS=","}FNR==NR{a[NR]=$1;next}{$37=a[FNR]}1'** stopwords.csv reddit\_done5.csv > reddit\_done6.csv

**rm** stopwords.csv

**(d)** Finally, to stem all necessary words in the title column I first created variables for my input file, output file and focus column. I then placed the title column into a variable called “new\_col” and used “cut” along with “-d” option to remove this column from the input file and save it to “title.csv”. I then used a “while” loop to iterate through the “diffs.txt” file which contains the words to be stemmed in the first column, and the words after stemming in the second column. In this loop, each word in the first column of “diffs.txt” is placed in “old\_string” and each word in the second column is placed in the “new\_string” variable. The loop then iterates through the “title.csv” file and when it finds a match with the first column of “diffs.txt”, it replaces this with the corresponding string in the second column of “diffs.txt”. The replacement is done using the “sed” function with the “-i -e” option and “\b” is again used to ensure only full strings are replaced. Lastly, I use the “awk” function to paste the new stemmed column back into our input file and then save it to our output file. I then remove the “title.csv” file as we no longer have any use for it. The code for this task is too large to paste below, but it can be found in the “stem.sh” file.

**Data Management**

**Creating the SQL database:**

1. Firstly, I created the database and named it “dba1” using the below command.

create database dba1;

Then, I created the first table “user” and inserted the values “author\_id”, “author” and “author\_cakeday”. I decided the types for these values should be varchar(20), varchar(60) and varchar(20) respectively. I set “author\_id” to be the primary key of this table and I set “author\_id” and “author” to not null. I used the command below to complete this task.

create table user ( author\_id varchar(20) not null, author varchar(60) not null, author\_cakeday varchar(20), primary key (author\_id) );

Next, I created the “subreddit” table and inserted the values “subreddit”. I chose varchar(60) as the type for this value, set it to not null and assigned it as the primary key of this table.

create table subreddit ( subreddit varchar(60) not null, primary key (subreddit) );

Finally, I created the “post” table and inserted the values “id”, “author\_id”, “subreddit”, “created\_month” and “title”. For these values, I chose the types varchar(20), varchar(20), varchar(60), varchar(10) and varchar(200) respectively. I set “id”, “subreddit” and “created\_month” to not null, and made it so that “post(author\_id)” references “user(author\_id)” and “subreddit(subreddit)” references “post(subreddit)”. I used the command below to complete this task.

create table post ( id varchar(20) not null, author\_id varchar(20) REFERENCES user(author\_id), subreddit varchar(60) not null REFERENCES subreddit(subreddit), created\_month varchar(10) not null, title varchar(200) );

**Populating the SQL database:**

1. I populated the database using bash script “populate.sh”. In this script, I used a while loop to iterate through the input file (reddit\_10K.csv) and place the necessary columns into their associated variables. For instance, the line below saves column 5 from the input file into the variable “author\_id”:

author\_id=`**echo** **$var** | **cut -d","** -f5`

After all of the necessary columns from the input file are placed in the correct variable, the loop begins to populate the SQL database using the command:

mysql **-h** localhost **-D** **'dba1' -e** **"INSERT INTO user VALUES (**

I used “reddit\_10K.csv” as input, which is a file containing the first 10,000 rows from the full-size file. I also implemented a counter, so that it was possible to see what row of the database was being populated at any given time. The full code segment for this task can be found in “populate.sh”.

**Queries for the SQL database:**

1. **(a)**

**List of author names:**

I chose to use the “select” function to list all author names from the table user. The select function returns all of the records from the specified table and field.

SELECT author FROM user;

**(b)**

**List of all posts’ title with their author’s name and the subreddit they were posted in:**

I chose to use the “select” function to list all titles, authors and subreddit pairs. I then used the “where” function to connect the post, user and subreddit tables. I then chose to ORDER the resulting table by order 1,2,3.

SELECT post.title, user.author, subreddit.subreddit

FROM post, user, subreddit

WHERE post.author\_id = user.author\_id

AND post.subreddit = subreddit.subreddit

ORDER BY 1,2,3;

**(c)**

**List of (subreddit, month) pairs, and the number of posts made in the subreddit during this month:**

I chose to use the “select” function to list all of the subreddit and created month pairs from the post table.

SELECT subreddit, created\_month FROM post;

Next, I used the “select” function again to list all subreddits and their frequencies within the table. I used the “group by” function to dictate how the data would be grouped and used the “ordered” function to specify that I wanted the data in descending order.

SELECT subreddit, count(\*)

FROM post

GROUP BY subreddit

ORDER BY count(\*) desc;

**Creating and populating the MongoDB collection:**

1. I began by reading my input file “reddit\_1K.csv” into the “input” variable. This csv contains the first 1000 lines of the full-size csv. I then used a for loop to iterate through the input file and assign each value in each column to a variable. For instance, this line of code reads column 16 into the “id” variable:

id=$(**cut -d','** -f16 <<<**"$var"**)

I then populated the MongoDB using “–eval” to execute the MongoDB commands, and the MongoDB “insert” function to place all of the variable values into the collection. I named my database “reddit\_DB” and I named my collection “reddit\_C”. The full code can be found in “populateMongoDB.sh” and I have given a snippet of the MongoDB “insert” command below.

mongo reddit\_DB **--eval** \

**'db.reddit\_C.insert({archived: "'"$archived"'"**

**Queries for the MongoDB collection:**

1. **(a)**

**List of author names:**

I chose to use the “find” function to list all of the author names in the MongoDB collection. The find function returns the specified number of records in whichever field chosen.

db.test.find({}, {author:1})

**(b)**

**List of all posts’ title with their author’s name and the subreddit they were posted in:**

I again chose to use the “find” function to list all titles, posts and author names from the MongoDB collection. I used the same format as the above command and simply added in title and subreddit.

db.test.find({}, {title:1, author:1, subreddit:1})

**(c)**

**List of (subreddit, month) pairs, and the number of posts made in the subreddit during this month:**

First, I used the “find” command again to select all values from “subreddit” and “created\_utc”. I then used the “aggregate” command which allows us to input our collection which we would like to query. Lastly, I use the “$sortByCount” command to both count the frequency of each subreddit in the collection, and to order the resulting list in descending order.

db.test.find({}, {subreddit:1, created\_utc:1})

db.test.aggregate([{$sortByCount:"$subreddit"}])

1. With mySQL, I had to change previous records stored in the database in order to apply the new structure because if there are multiple subreddits per post, subreddit can no longer be a primary key, and this change will affect previous records. I attempted to add multiple subreddits per post without changing the structure of the database but failed.

With MongoDB, I did not have to change previous records stored in the database to apply the new structure because MongoDB has “update”, “updateMany” and “replaceOne” commands which can facilitate our need to add multiple subreddits per post.

1. With mySQL, the previous queries do not work, because in order to facilitate multiple subreddits, previous records and keys have been changed, which will not allow the queries to work as they have previously.

With MongoDB, the previous queries still work because there has been no change to previous records. All records remain as they were initially when the queries were being implemented, and so, they still work as they did previously.

**Reflection**

**[1]**

My experience of using CLI (Bash) for Big Data tasks has been very good. I find it extremely suitable for Big Data tasks. When comparing the speed, precision and access Bash gives me to the data, with languages like Python for example, I found that Bash outperformed them in many ways. While customisation and ease of access may be better in other languages such as python, I found bash to be much quicker, much more compact and much more precise than any of its counterparts. The speed in which the bash scripts would iterate through the huge input csv file astounded me, and I found it extremely suitable for Big Data tasks compared to many other methods I have studied thus far.

**[2]**

I would like to begin this comparison between NoSQL and relational database management models by outlining the key characteristics of each model. Relational database models provide a store of related tables which have a fixed schema, store data in rows and columns, use SQL to manage data, and support ACID guarantees. Each row contains a unique instance of data for the categories defined by the columns and the use of primary keys is possible to uniquely identify rows. NoSQL database models are high-performance, non-relational data stores. They are easy to use, scalable, resilient and available, and they contain unstructured or semi-structured data.

Both of these database management models are optimised for different scenarios, and arguments for which management model is better than the other, wholly depends on what context the model is being used in.

The relational model thrives under medium to large workloads, especially when the workload volume is consistent. It also thrives when working with complex queries and reports, when write safety is a requirement, when users are relatively centralised, and when the application will be deployed to large or high-end software. However, relational models struggle in terms of scalability and complexity. Users must scale relational databases on powerful servers and if your data doesn’t fit into tables in the database, a re-design of the entire structure must take place.

The NoSQL model is best when there are very high-volume workloads that require data on large scales, when your data is dynamic and frequently changing, when data can be expressed without relationships, when you need fast writes and write safety isn’t critical, and when your data requires a large geographic distribution. However, NoSQL databases have little or no defined standards which means it is very difficult to compare any two databases. They are very new compared to the relational model and so, are far less stable and are more complex.

Taking all of these factors into account it is very hard to say which database management model is outright better than the other. In general, NoSQL tends to suit modern applications and databases better than its counterpart. The increasingly complex and changeable data from modern day applications requires a model that is very flexible and that doesn’t need to be immediately defined. The relational model tends to perform better than the NoSQL model under traditional circumstances where data is very structured and definite. Relational databases also have very defined rules and standards which makes comparing this type of database very easy.

Neither database management is better than the other in all contexts. Both have strengths and both have weaknesses depending on the context which the model is used in. One could say that in modern times, NoSQL databases are much more flexible and can be used for a wider variety of uses which is fair, however, there are still many scenarios where relational databases outperform NoSQL databases, and for that reason, without the context in which the database is being used, I think it is very difficult to outright say one model is better than the other.

**[3]**

For this task, I have chosen to write my report on the “Dynamo: Amazon’s Highly Available Key-value Store” research paper.

The main motivation for this paper is the fact that, as the largest e-commerce operation in the world, even the slightest outage has significant financial consequences for Amazon, and their reputation is such that a small outage can massively impact customer trust. The sheer scale of the infrastructure necessary to run amazon means that tens of thousands of servers and data centres are being used at any given time, and at this scale, small and large parts of this infrastructure fail on a continuous basis. As such, Amazon needs a cost-effective system that treats failure handling without impacting availability or performance. Amazon is composed of hundreds of services that have to work in tandem with each other in order to deliver the functionality that Amazon demands. Traditional relational databases cannot fulfil this task, and as such, Amazon must such for a new and innovative solution.

Amazon looked at many different possible solutions to this problem before deciding on a final plan to implement. They started by looking at peer-to-peer systems that have tackled the problem of data storage and distribution, such as Freenet and Gnutella. These are examples of unstructured P2P networks, however Amazon also looked at structured P2P networks such as Pastry and Chord.

They also looked at many different distributed file systems and databases while searching for a solution to their problem. Compared to P2P storage systems that only support flat namespaces, distributed file systems like Ficus and Coda support hierarchical namespaces, which made distributed file systems more suited towards Amazon’s needs. Systems such as Bigtable and Antiquity where also looked at, however due to these systems being limited in scalability and availability, they did not fully satisfy Amazon’s extensive list of needs.

Amazon’s solution to this problem is the Dynamo system. Dynamo is a highly available data storage technology that addresses the needs of all of Amazon’s important services. Some of the key advantages of the Dynamo system outlined by Amazon are:

* Scalability.
* Version size is decoupled from update rate.
* High availability.
* Synchronization.
* Preserves symmetry and avoids a centralised registry.

Dynamo allows service owners to customize their storage system to meet their desired performance, durability and consistency. Unlike many popular commercial data stores, Dynamo exposes data consistency and reconciliation logic issues to the developers, which means that should a failure happen within the system, it should be possible to find a solution relatively quickly compared to other systems.

Amazon found that the Dynamo system provided the desired levels of availability and performance, and the system has been largely successful in handling server failures, data centre failures and network partitions. Dynamo is very scalable and allows the user to scale up and down to their desired level. Amazon have been extremely happy with the customization levels of Dynamo to meet their desired performance, durability and consistency levels and have been a benchmark case in the use of systems such as these. Being the largest online retailer in the world produced a very unique problem for Amazon, Dynamo was the unique solution which they required. It’s success in one of the most challenging application environments has set a precedent for highly available applications.

I thoroughly enjoyed reading this paper and found the insight into Amazon’s thought process extremely interesting. Dynamo is an extremely interesting system, and reading how customizable, scalable, flexible and consistent it is, even while working with data levels on such a huge scale, is quite incredible. I really liked how the paper was written and found it very informative and easy to read, even though there are so many difficult technical terms throughout the report. It is tough to outline any obvious limitations of the Dynamo system because of its customisability and scalability, however, due to Dynamo being a relatively new technology, and due to how fast the needs of Amazon are evolving, it is quite easy to imagine Amazon needing even more from a database system than Dynamo can provide. Dynamo works for Amazon currently, but will it be able to continue to handle the data with the almost perfect failure handling that Amazon expects is a question that remains to be answered. In saying this, by reading the research paper it is clear to see that Amazon is extremely happy with the performance of Dynamo and it seems as though they have no plans to change system anytime soon.