School of Computer Science

COMP30770

Project 1
CLI (Bash) & Data Management for Big Data

Name	Brian Byrne
Student Number	18391933
Date	14-02-2020
Number of Pages	8

This report details the work done for the COMP30770 project. In this project I worked with a Reddit big dataset using bash to clean the dataset, before using the bash scripts to create both relational and non-relational databases using MySQL and MongoDB respectively, before preforming various queries to access the data from the databases.

This report is structured into three sections. The first section explains how I used multiple bash scripts to preform basic data cleaning tasks on the csv file such as removing empty columns and rewording data for easier human interpretation. The second section consists of creating both a relational database using bash scripts and MySQL, and then a NoSQL database using MongoDB and then using several queries to demonstrate the functionality of these databases. The final task involves answering questions on the use of CLI, comparing relational and non-relational database models and finally a short report on a relevant research paper by Werner Vogels' titled "*Building reliable distributed systems at a worldwide scale demands trade-offs between consistency and availability.*".

The code for the project has been submitted alongside this report, its use is described in README.txt

1 Cleaning a Dataset with Bash

1) In order to drop these columns, I used the cut command that we used in the labs with the –complement call to remove the listed columns and the > notation to write the remaining columns to a new temporary csv file called "*cleaned_reddit_data*" for further cleaning. The script for this process can be seen in the file q1.sh.

2) The goal of this exercise was to write a bash script that removed the empty columns of the CSV.

   To remove the remaining additionally empty columns I used the followed the hint given in the question. I got the number of columns using a SED call for the amount of strings separated by commas in the first line of the file. I then copied my cleaned_reddit_data.csv to a temp.csv to preform updates later in the following loops. To begin I looped backwards through the columns using a for loop so that if a column was removed, the updated file could still be looped through as normal despites having less columns. I set a bool variable to false and looped through each cell in each column using a nested for loop that started in column 2 (ignoring the header cell). If the column contained a non-empty cell, then I set the bool variable to true. I did this using an if statement that checked for a non-empty cell. At the end of the loop I removed all columns where the bool was false using an if statement and the cut command with the iteration "i" as the column number, and wrote the updated file to a new csv using the > notation. I updated my CSV after each column had been removed as the cut function only cuts one line per iteration from the existing to file. This script can be found in the q2.sh file.

3) The goal of this exercise was to identify uninformative columns in the dataset i.e. columns where the value of each cell was the same.

   Similar to the last question I used a for loop to loop through each column of the dataset and a bool variable to identify which column was to be removed. I used multiple pipes in a statement that wen through each column, and the sort and uniq calls to identify all the unique cell types. I then used the "wc -l" call to count the number of unique entries in the columns and assigned that number to a variable. I then wrote a short algorithm consisting of an if statement that would change the bool variable to true if the number of unique entries was not equal to 1 (meaning there were at least 2 different types of entries in the column, so that column was informative. If the bool was false and the same entry was entered into every cell, then that column would be removed. I again used the copy function and the > notation to constantly update my file. This script can be found in the q3.sh file.

4) The goal of question 4 was to convert the time from seconds since epoch to the name of the month when that second occurred.

   Here I used two for loops and the cut function to loop through the specific columns that had the time in seconds in them. I made a variable month and used the date command and '+%B' to return a string of the name of the date to the variable. I then used the "sed -i" command to directly replace the occurrences of epoch time in each cell with the name of the relevant month. Both of the for loops are practically identical except the cut command is accessing different columns. This script can be found in the q4.sh file.

5) The purpose of question 5 was to identify how many posts had been made in each month.

   Using the most recent file (temp3.csv), I used a one-line piped command that went through the entries of the column "created_utc" and used the sort pipe uniq call again with -c to count each entry type (in this case month) in the column, displaying the month and the number of times it occurred in the column. This script can be found in the q5.sh file.

6) Question 6 of part one was divided into four different parts, all centred around the title column. We were tasked with converting all the letters to lowercase, removing punctuation, removing some "stop words" and reducing the words using a basic stemming technique.

   I had an issue using the "sed" function to replace the existing strings in the column with there lowercase equivalents, mainly due to the presence of permalinks and special characters like dashes and colons, so I decided to take a different approach. I cut used the cut command to separate the files into three different sections: the first being everything before the title column, the second being the title column and the third being the remaining columns. During the cut command for the title column I used pipe and the “tr '[:upper:]' '[:lower:]'” command to convert that line to lower case. Furthermore, I wrote all of these sections to three csv files and then uses the paste command to append the files into an outputted “temp4.csv”, where the title column was all lower case and the other columns remained the same, in the same order. This script can be found in the q6a.sh file.

For removing punctuation from the title column, I used an algorithm very similar to my previous question, in this case I again split the file into 3 different sections and using the tr command after the pipe, however using the tr -d '[:punct:]' command, with the -d to prevent commas from being removed and all the data in the table being moved into one column. I once again used the paste command to combine all three sections and write to the updated temporary csv file. In this case temp5.csv. . This script can be found in the q6b.sh file.

2 Data Management

1) I created the database using the following SQL commands:

```
create database project1;
use project1;
```

I then created the internal tables of the database using the following commands:

```
create table user (
        author_id varchar(20) not null,
        author varchar(40) not null,
        author_cakeday varchar(8),
        primary key (author_id)
        );

 create table subreddit (
    subreddit varchar(30) not null,
    primary key (subreddit)
        );


 create table post (
   id varchar(10) not null,
   author_id varchar(20) not null,
    subreddit varchar(30) not null,
   created_month varchar(10) not null,
    title varchar(300), not null,
   primary key (id),
   constraint fk_author_id foreign key (author_id) references user (author_id),
  constraint fk_subreddit foreign key (subreddit) references subreddit (subreddit)
  );
```

2) I populated the MySQL database with the populateMySQL.sh script. It reads the latest dataset line by line and uses the cut command to create relevant variables from the dataset columns, starting on the second line to ignore headers. My username was “root” and I had a simple password ‘s’. then I used the call insert into and dot notation to specify by both the database and the table. I then added the variable names in the positions where they should be entered in the table in accordance to the column headers.

3) I wrote the following queries to answer the questions:

   a) `SELECT author FROM user;`

      Very basic call that lists the authors names stored in one table.

   b)
   ```
   SELECT
     title,
     author,
     subreddit
   FROM
     post p
   INNER JOIN user u
     ON p.author_id = u.author_id;
   ```
      Retrieving data from two tables where we get the authors name from the table user and author_id is a foreign key in post and the primary key in user. We made the connection between the tables using the INNER JOIN call.

   c)
   ```
   SELECT subreddit, created_month, COUNT(*)
   FROM post GROUP BY subreddit,   created_month;
   ```
      Counts the occurrences of each subreddit as (one occurrence= one post) using the count call and a simple group by call will pair the subreddits by month as specified in the question.

4) I populated the MongoDB database with the populateMongo.sh script. I read in the temp5.csv file like for my MySQL database. I used the while loop -d to make sure it used the delimiter to separate the file by commas when it read it in line by line. I then used the insert call to add a new NoSQL document to the database by manually inserting the headers as keys in the statement and using the a var variable as an array to populate the dataset of it’s coupled key.

5) I translated my previous queries from SQL to mongo: My simple SQL queries became:

a) db.myRedditCollection.find({}, {author:1})
   A quite basic query that returns a list of all data related to the key author in the myRedditCollection in the database.
b) db.myRedditCollection.find({}, {title:1,author:1,subreddit:1})
   There is no need for complexity or foreign keys here because all of the data is stored here dynamically, so this will return the resulting rows with the relevant data.
c) db.myRedditCollection.aggregate(group:{subreddit:1,created_utc:1},count{sum:1})
   Here we use the aggregate call externally and then we use the group function to group the subreddit by the month it was created in. in the second parentheses we call the count call to count the number of posts per pairing.

3 Reflection

1) From using CLI (Bash) for this assignment, I have found that bash is very efficient when using built in functions that deal with entire columns such as cut, sort, uniq, etc. The functions run these functions more or less instantaneously. However, depending on the size of your dataset, the running time can vary greatly. I noticed this particularly when I was using loops and nested loops that looped through every cell in the dataset. For question 4 I had to wait hours for the script to finish running the 68,000 lines, where the code converted the Epoch time to the month name. Nowadays, 68,000 lines would be considered relatively small for a Bid Dataset, and so Bash may not be the most time effective way to process data, as bash scripts could run for days preforming basic operations of each cell in a column.

2) Relational database management models such as SQL and non- relational (NoSQL) database management models have many unique differences when compared. Relational databases contain data storage tables that contain data on specific entries. These tables can form relations with other tables in the databases. They have predefined, structured schema which enables data retrieval by using relative query languages and relational algebra on the database. In general, relational databases allow for easy understanding as they are quite simple in structure and are designed for easy access to accurate data.

   Non-relational databases contain data that is not limited to specific tables. It allows for handling of unstructured data, which makes it easier to store agile open-source data. This allows for much dynamic and scalable data, with simple readability. While this makes NoSQL ideal for processing larger datasets, it should also be noted that non-relational databases often have less functionality than their relational counterparts, often limited by the specific database management system the user is running the data on.

Overall, we can see that relational databases are vertical, table-based databases where columns are used to split the data and we can also see that NoSQL databases are generally wide-columned and horizontal databases where each row can be dynamic in size and data. A scenario where a non-relational database like MongoDB would be preferable would be in an area of rapid data growth and change, such as a user dataset of an app like TikTok that has many of new users joining the service every day. On the other hand, a relational database like MySQL would be suited for smaller datasets, where specific traits are prioritises, such as a dataset of employees in different departments of a large company.

3) Werner Vogels' publication "*Building reliable distributed systems at a worldwide scale demands trade-offs between consistency and availability.*" Is an investigation into how the ever growing databases of large corporations are sacrificing the accuracy and reliability of data to cater for their billions and trillions of system processes. Vogels' in this paper identifies methods currently being used by companies to process large amounts of data, pointing out faults and issues with the consistency of this techniques and suggests the application of basic practices to help improve the balance of consistency and availability to enable more accurate data processing.

   Vogels addresses issues arising from asynchronous replication of data, an outcome of poor consistency. Weak consistency means the system doesn't guarantee that it will return and updated value, as the update has been delayed, and the old, inconsistent value is read by accident. He explains to us how when there is such a large database with trillions of requests that once had a low probability of occurring in a smaller database, are now guaranteed to happen. In his opinion, these new problems have made the idea of eventual consistency obsolete.

   While Vogels tells us that this is a very complex problem to address, he offers an example remedy to this poor practice in the form of Amazon's Dynamo." *A system that has brought all of these properties under explicit control of the application architecture is Amazon's Dynamo*" states Vogels. Here Vogels describes the positive work of Amazon to combat this issue with their data management system Dynamo. He explains how Dynamo is a non-relational or NoSQL data system that the when the an instance is created on the Dynamo architecture, it is often spread to multiple data centres around the world, making sure the probability of potential inconsistencies with this instance are minimalised. Furthermore, he explains that this solution is not air-tight, and inconsistencies are still possible, but that this methodology is a huge leap in the right direction for improved consistency and accuracy in Big Data Management.

Another area Vogels addresses is the partition and loss of data due partitions. In some cases, he informs us that often nodes containing the old information and new information cannot reach eachother, offering another solution, again used by Amazon where the nodes select a new free node, and merge their information to prevent partition. Vogels excellently supports this point with a vivid example in the form of an Amazon users' old cart and new car merging to prevent the loss of the old carts contents, something that could easily occur on sites with less agile data storage techniques.

Moreover, Vogels clearly explains the basic mathematics of consistency and inconsistency with the "W+R>N" where W is the number of replicas needed to acknowledge an update before an update is completed, R is the number of replicas accessed when the data is accessed, and N is the number of nodes that store the data. This guarantees consistency because the R set and W set always overlap ensuring accuracy and efficiency. He goes on to show us that often with big datasets, there is a insufficient number of free nodes for "W+R>N", and consistency cannot be guaranteed as a result.

To conclude, this paper is a very well structured and researched paper. While Vogels is sympathetic for issue, the only real solution he offers for improving large scale data management, involves global physical storage infrastructure, and even so this still doesn't solve the issue of inconsistency. For me this raises two critiques about this well-formed argument of a valid, real-world and relevant issue. The first issue is that his solutions are based on the Amazon's worldwide hardware network is realistically no help to the vast majority of firms who do not possess the physical infrastructure or capital of one of the worlds must lucrative companies like Amazon. Secondly, even Amazon with their wealth of resources still cannot completely eradicate the issue at hand, only reduce its probability. Overall, this problem is very relevant and one that seems to require a lot more research.