**Problem 1 Commands**

1. Using “echo”, print your “Candidate Number” (find it on BART) to screen. Take a screenshot and include it in your submission.

echo Candidate Number: 244636

1. Move inside the directory *DATE\_FILES*

cd OneDrive\ -\ University\ of\ Exeter/Second\ Year/Data\Science/Assignment/data/DATE\_FILES

1. Count the number of files in this directory

ls -1 | wc -l

1. Print the names of the first 8 files in this directory, along with information about owner,   
   date, and size.

ls -l | awk 'NR>1 {print $9, $3, $6, $7, $5}' | head -8

1. Move to the parent directory

cd ..

1. Create a new directory there, named *second\_10\_days*

mkdir second\_10\_days

1. Copy from the *DATE\_FILES* directory the files that are related to the days 10-19 of every   
   month (140 files) to the newly created directory

cp DATE\_FILES/\*\_??\_[1][0-9] second\_10\_days/

 ls -1 second\_10\_days | wc -l

1. Move inside *second\_10\_days* directory, and append the line “This is the last Line” to the   
   end of file *2011\_04\_10*

cd second\_10\_days

echo "This is the last line" >> 2011\_04\_10

1. Write a one-liner command to append the line “This is the last Line of X”, where X is the   
   name of the file, to the end of every file in the directory *second\_10\_days*

for file in second\_10\_days/\*; do echo "This is the last Line of $(basename "$file")" >> "$file"; done

1. Using Bash: create a bash file *P1K.sh*. Write your code from J to it. Run the file *P1K.sh* file.

nano P1k.sh

chmod +x P1k.sh

./P1k.sh

cd second\_10\_day

tail -n 1 $(ls | head -n 10)

**Problem 3 Commands**

1. Create a new database in SQLite named *P3.db*

sqlite3 P3.db

1. Have a look at the two data files *US\_codes.txt* and *US\_population.csv*. Create two tables   
   *US\_Code* and *US\_Pop* with column headings that match these two data frames. The former does not have headings. Use the following headings:   
   ‘CountryCode','ZipCode','City','StateFull','State2','CountyFull','FIPSCountyCode',   
   'MunicipalityFull','MunicipalityCode', ‘Latitude’, ' Longitude', 'Accuracy' The latter has headings, but you need to use the following headings:   
   ’Geo\_ID’,’Zip’,’Gender’,’AgeRange’,‘Population’  
   Make sure you choose appropriate types and constraints for columns, and appropriate primary keys and foreign keys, if any. Use ‘ZipCode’ as a Primary Key for *US\_Code.* For *US\_Pop*, add a column called ‘ID’ which would be the Primary Key of that table. ‘ID’ should have an auto-increment feature (it starts with 1 and increases automatically for each record).

CREATE TABLE US\_Code (

        CountryCode TEXT,

        ZipCode TEXT PRIMARY KEY,

        City TEXT,

        StateFull TEXT,

        State2 TEXT,

        CountyFull TEXT,

        FIPSCountyCode TEXT,

        MunicipalityFull TEXT,

        MunicipalityCode TEXT,

        Latitude REAL,

        Longitude REAL,

        Accuracy INT

);

CREATE TABLE US\_Pop (

        ID INTEGER PRIMARY KEY AUTOINCREMENT,

        Geo\_ID TEXT,

        Zip TEXT,

        Gender TEXT,

        AgeRange TEXT,

        Population INT

);

1. Insert the data from the two files into the two tables. Make sure you don’t insert the column heading from the file *US\_population.csv*. Explain what you did about that.

CREATE TABLE US\_Pop\_Interim (

        Geo\_ID TEXT,

        Zip INTEGER,

        Gender TEXT,

        AgeRange TEXT,

        Population INTEGER

);

.mode csv

.import US\_population.csv US\_Pop\_Interim --skip 1

(to skip the header row in the import process, I used the .import command with the --skip option to skip the first row )

INSERT INTO US\_Pop (Geo\_ID, Zip, Gender, AgeRange, Population) SELECT \* FROM US\_Pop\_Interim;

DROP TABLE US\_Pop\_Interim;

.mode tabs

.import US\_codes.txt US\_Code

1. Write an SQL query to print the total population per gender (use the *US\_Pop* table only)

SELECT Gender, SUM(Population) AS Total\_Population

FROM US\_Pop

GROUP BY Gender;

1. Write an SQL query to print the total population per gender, but join the two tables.   
   Explain why you see different numbers from part D

SELECT US\_Pop.gender, SUM(US\_Pop.population) AS total\_population

FROM US\_Pop

INNER JOIN US\_Code ON US\_Pop.Zip = US\_Code.ZipCode

GROUP BY US\_Pop.gender;

The query in part E uses an inner join to join US pop and Us code tables. The inner join creates a set that only contains rows from the US pop for which there is a matching zip code in the US code table. The total population is then calculated using this resultant set. There is a difference in the numbers because there may be certain rows / entries in US pop for which there is no row with a matching zip code in the US code table and so when the population is calculated from the set resulting from the inner join there is a lower total as there are less rows/entries in the resultant set than in the US pop table. .

1. Write an SQL query to print the total population per age group (use the *US\_Pop* table   
   only)

SELECT AgeRange, SUM(Population) as TotalPopulation

FROM US\_Pop

GROUP BY AgeRange

ORDER BY CAST(SUBSTR(AgeRange, 1, INSTR(AgeRange, '-') - 1) AS INTEGER) ASC,

         CAST(SUBSTR(AgeRange, INSTR(AgeRange, '-') + 1) AS INTEGER) ASC

;

1. Write an SQL query to print the Top 10 largest states (full name) in terms of population   
   size

SELECT US\_Code.StateFull, SUM(US\_Pop.Population) as TotalPopulation

FROM US\_Pop

JOIN US\_Code ON US\_Pop.Zip = US\_Code.ZipCode

GROUP BY US\_Code.StateFull

ORDER BY TotalPopulation DESC

LIMIT 10;

1. Write an SQL query to print the number of existing counties (not countries) in the data   
   base

SELECT COUNT(DISTINCT CountyFull) AS NumCounties

FROM US\_Code;

1. Write an SQL query to print the total population per gender and age group for the county named “Middlesex”.

SELECT US\_Pop.Gender, US\_Pop.AgeRange, SUM(US\_Pop.Population) AS TotalPopulation

        FROM US\_Pop

        JOIN US\_Code ON US\_Pop.Zip = US\_Code.Zipcode

        WHERE US\_Code.CountyFull = 'Middlesex'

        GROUP BY US\_Pop.Gender, US\_Pop.AgeRange

        ORDER BY CAST(SUBSTR(AgeRange, 1, INSTR(AgeRange, '-') - 1) AS INTEGER) ASC,

                 CAST(SUBSTR(AgeRange, INSTR(AgeRange, '-') + 1) AS INTEGER) ASC;

**Problem 4 Commands**

1. Import the *tvshows.json* file in your local MongoDB under the database name *TV\_Shows* with a collection name *shows*

mongoimport --db TV\_Shows --collection shows --file tvshows.json --jsonArray

1. Write a query that counts the number of documents in the *shows* collection
   * First have to go into mongo shell and switch to tv show database

mongosh

use TV\_Shows

db.shows.countDocuments()

1. Write a query that prints all information of the show named *"How I Met Your Mother"*

db.shows.find({name: "How I Met Your Mother"}).pretty()

1. Write a query that prints shows that are stored in the collection in the order 27th -28th

db.shows.find().limit(2).skip(26).pretty()

1. Write a query that counts the number of shows that were still running at the time the   
   data was collected

db.shows.find({status: 'Running'}).count()

1. Write a query that counts the number of shows that have an indicated web channel.

db.shows.countDocuments({ webChannel: { $ne: null } })

1. Write a query that counts the number of “Comedy” shows in the *shows* collection

db.shows.count({genres: "Comedy"})

1. Write a query that prints the one show that has the following three genres: 'Drama',   
   'Thriller', and 'Comedy'

db.shows.find({genres:{$all:["Drama","Thriller","Comedy"]}}).pretty()

1. Write a query that prints only the name and genres of shows that have (at least) both of   
   the following two genres: 'Drama' and 'Comedy'

db.shows.find({genres: {$all: ['Drama', 'Comedy']}}, {name: 1, genres: 1, \_id: 0})

1. Write a query that prints the average rating of ‘Comedy’ shows

db.shows.aggregate([ {$match: {genres: "Comedy"}}, {$group: {\_id: null, avgRating: {$avg: "$rating.average"}}} ])

1. Write a query that prints a unique list of all genres

db.shows.distinct("genres")