Applied Databases

Higher Diploma in Science in Data Analytics

1 Description

This document describes the final project specification for the Applied Databases module.

2 Marks

This project is worth 60% of the marks for the module.

2.1 Marking Scheme

85% of the marks will be awarded for implementing the functionality described in this document.

15% of the marks will be awarded for innovation and extra functionality.

Please describe your innovation (if any) in a document entitled *innovation.doc* which should be stored in the root folder of your submission.

2.1.1 Plagiarism

Plagiarism will be dealt with in accordance with the institute's Plagiarism policy.

3 Submission

Your submission should be a zipped file called your student number .zip e.g. G001234567.zip.

It should be uploaded to the *Project* section of Moodle for this module before **Sunday May 19th 2019 at 11:55pm** and contain the following:

- MySQL.txt
  This .txt file (no formatting) should contain your answers to section 4.1 of this specification.

- Normalisation.doc
  This word document should contain answers to section 4.2 of this specification.

- MongoDB.txt
  This .txt file (no formatting) should contain your answers to section 4.3 of this specification.

- Python
  This folder should contain the python file(s) containing your answers to section 4.4 of this specification.

- Innovation.doc
  This optional word document should describe any extra features you have implemented in addition to the requirements in this document.
  For example, if you have added extra functionality to the python program, it can be briefly described here.

4 Functionality

4.1 MySQL

Import the *world* database from *world.sql* to MySQL and write queries to satisfy the following.

4.1.1 Get people who have visited a particular country

Write a MySQL procedure called `get_ppl_visited_country` that takes one parameter of type `varchar(52)` which represents the name particular country.

The procedure should display the following details of people who have visited that country:

- The person's ID
- The person's name
- The name of the city/cities the person visited in the country
- The date the person arrived in the city/cities
- The country's full name

NOTE: The country name does not have to be an exact match e.g. "Irel" should match "Ireland", "tina" should match "Argentina" etc.

```
mysql> call get_ppl_visited_country("land");
+----------+------------+-----------+-------------+-------------+
| personid | personname | name      | dateArrived | name        |
+----------+------------+-----------+-------------+-------------+
|        2 | Alan       | Arnhem    | 2005-04-14  | Netherlands |
|        4 | Sara       | Zürich    | 1999-01-20  | Switzerland |
|        3 | Sean       | Dordrecht | 2000-06-20  | Netherlands |
|        1 | Tom        | Dordrecht | 2002-02-11  | Netherlands |
+----------+------------+-----------+-------------+-------------+
4 rows in set (0.00 sec)

Query OK, 0 rows affected (0.03 sec)
```

Figure 1 Example call to procedure get_ppl_visited_country

4.1.2 Rename Continent

Write a function that called `ren_continent` that takes one parameter which is a Continent name and returns the New Name associated with the Continent name passed to it as follows:

Original Name	New Name
North America, South America	Americas
Oceania	Australia
Antartica	South Pole

Table 1 Original and New continent names

4.1.3 Country with biggest population per continent

Give the MySQL command to show the continent, and the name and population of the country with the biggest population in each continent.

NOTE: Only include countries where the population is greater than 0.

4.1.4 Minimum city population of youngest person(s)

Give the MySQL command to show the name/names of the city/cities with the lowest population, that the youngest person/persons has/have visited.

4.1.5 Update City Populations

Write a single MySQL command to increase the population of South African cities depending on their district as follows:

District	City Population Change
Eastern Cape	+1,000
Free State	+2,000
Western Cape	-10,000

Table 2 Changes to populations of South African cities based on District

4.1.6 Country Independence

Write a MySQL query show the name and year of independence of each country, as well as a column called "Desc" which has the following information.

- If the country was never independent – "n/a"
- If the country became independent – the "governmentform" of the country should be shown with the following extra information:
    - If the country became independent less than 10 years ago, the word "New" should be prepended to "governmentform".
    - If the country became independent between 10 and 49 years ago, the word "Modern" should be prepended to "governmentform".
    - If the country became independent between 50 and 100 years ago, the word "Early" should be prepended to "governmentform".
    - If the country became independent more than 100 years ago, the word "Old" should be prepended to "governmentform".

    - In addition if the population of the country is more than 100 million, the word "Large" should be prepended to "governmentform".

Examples:

Name	Indepyear	Desc
Aruba	NULL	n/a
Bangladesh	1971	Modern Large Republic
Zimbabwe	1980	Modern Republic

Table 3 Name, Independence Year and Description of countries

4.2 Normalisation

4.2.1 Database Design

Examine the following database (consisting of one table) that was designed to store the following information:

- Student ID
- Student Name
- Student Dob
- Modules Student is studying

Students can enrol in the college before deciding which modules to take, and not all modules are offered each year.

The following database, consisting of one table with the primary key = studentID and moduleID, was designed.

Give your opinion, using examples from the data below, on whether or not the current database is good or bad.

studentID*	studentName	dob	moduleID*	moduleName
1	Sean	2000-01-03	100	Applied Databases
2	Bill	1990-04-23	100	Applied Databases
3	Tom	1973-12-10	101	Java Programming
3	Tom	1973-12-10	104	Mobile Apps
4	Mary	1991-04-12	101	Java Programming
4	Mary	1991-04-12	102	Computer Architecture
5	Joe	1982-06-29	100	Applied Databases
5	Joe	1982-06-29	104	Mobile Apps

Table 4 Proposed database table

4.3 MongoDB

Import the file *mongo.json* to a collection called *docs* and write queries to satisfy the following.

4.3.1 Average Engine Size

Give the MongoDB command to find the average engine size.

4.3.2 Categorise County Populations

Give the MongoDB command to categorise documents based on their populations as follows:

- Between 0 and 49,999
- Between 50,000 and 99,999
- Between 100,000 and 149,000
- Over 150,000

For each population range the names of the counties in the range should be printed:

```
{ "_id" : 0, "counties" : [ "Leitrim" ] }
{ "_id" : 50000, "counties" : [ "Westmeath" ] }
{ "_id" : 100000, "counties" : [ "Mayo" ] }
{ "_id" : "Other", "counties" : [ "Galway", "Dublin" ] }
```

Figure 2 Documents categorised by population

4.3.3 Redefine County Populations

Give the MongoDB command to redefine documents based on their populations as follows:

- Less than 100,000 – “Small County”
- Greater than 99,999 – “Big County”

For each county the _id, name and pop should be shown as follows:

```
{ "_id" : "G", "name" : "Galway", "pop" : "Big County" }
{ "_id" : "WH", "name" : "Westmeath", "pop" : "Small County" }
{ "_id" : "MO", "name" : "Mayo", "pop" : "Big County" }
{ "_id" : "LM", "name" : "Leitrim", "pop" : "Small County" }
{ "_id" : "D", "name" : "Dublin", "pop" : "Big County" }
```

Figure 3 Documents redefined based on population

4.4 Python

4.4.1 Python program

Write a python program that displays a main menu as follows:

```
World DB
--------

MENU
====
1 - View 15 Cities
2 - View Cities by population
3 - Add New City
4 - Find Car by Engine Size
5 - Add New Car
6 - View Countries by name
7 - View Countries by population
x - Exit application
Choice:
```

Figure 4 Main Menu

The choices are as follows:

- **1**

  The user is shown the first 15 cities in the *world* database:

```
Choice: 1
1  |  Kabul  |  AFG  |  Kabol  |  1780000
2  |  Qandahar  |  AFG  |  Qandahar  |  237500
3  |  Herat  |  AFG  |  Herat  |  186800
4  |  Mazar-e-Sharif  |  AFG  |  Balkh  |  127800
5  |  Amsterdam  |  NLD  |  Noord-Holland  |  731200
6  |  Rotterdam  |  NLD  |  Zuid-Holland  |  593321
7  |  Haag  |  NLD  |  Zuid-Holland  |  440900
8  |  Utrecht  |  NLD  |  Utrecht  |  234323
9  |  Eindhoven  |  NLD  |  Noord-Brabant  |  201843
10  |  Tilburg  |  NLD  |  Noord-Brabant  |  193238
11  |  Groningen  |  NLD  |  Groningen  |  172701
12  |  Breda  |  NLD  |  Noord-Brabant  |  160398
13  |  Apeldoorn  |  NLD  |  Gelderland  |  153491
14  |  Nijmegen  |  NLD  |  Gelderland  |  152463
15  |  Enschede  |  NLD  |  Overijssel  |  149544
```

Figure 5 First 15 cities

- **2**

  The user is asked to enter in <, > or = and a number.

  If < and 999 were entered, the program would return all cities with a population of < 999.

```
Choice: 2

Cities By Population
--------------------
Enter < > or = : <
Enter population : 999
61   ¦  South Hill   ¦  AIA   ¦  û   ¦  961
62   ¦  The Valley   ¦  AIA   ¦  û   ¦  595
1791   ¦  Flying Fish Cove   ¦  CXR   ¦  û   ¦  700
2316   ¦  Bantam   ¦  CCK   ¦  Home Island   ¦  503
2317   ¦  West Island   ¦  CCK   ¦  West Island   ¦  167
2728   ¦  Yaren   ¦  NRU   ¦  û   ¦  559
2805   ¦  Alofi   ¦  NIU   ¦  û   ¦  682
2806   ¦  Kingston   ¦  NFK   ¦  û   ¦  800
2912   ¦  Adamstown   ¦  PCN   ¦  û   ¦  42
3333   ¦  Fakaofo   ¦  TKL   ¦  Fakaofo   ¦  300
3538   ¦  Cittó del Vaticano   ¦  VAT   ¦  û   ¦  455
```

Figure 6 Cities with population < 999

If > and 8000000 were entered, the program would return all cities with a population of > 8000000.

The same logic would apply for =.

```
Choice: 2

Cities By Population
--------------------
Enter < > or = : >
Enter population : 8000000
206   ¦  Sòo Paulo   ¦  BRA   ¦  Sòo Paulo   ¦  9968485
939   ¦  Jakarta   ¦  IDN   ¦  Jakarta Raya   ¦  9604900
1024   ¦  Mumbai (Bombay)   ¦  IND   ¦  Maharashtra   ¦  10500000
1890   ¦  Shanghai   ¦  CHN   ¦  Shanghai   ¦  9696300
2331   ¦  Seoul   ¦  KOR   ¦  Seoul   ¦  9981619
2515   ¦  Ciudad de México   ¦  MEX   ¦  Distrito Federal   ¦  8591309
2822   ¦  Karachi   ¦  PAK   ¦  Sindh   ¦  9269265
3357   ¦  Istanbul   ¦  TUR   ¦  Istanbul   ¦  8787958
3580   ¦  Moscow   ¦  RUS   ¦  Moscow (City)   ¦  8389200
3793   ¦  New York   ¦  USA   ¦  New York   ¦  8008278
```

Figure 7 Cities with population > 8000000

- **3**

  The user is asked to enter details of a new city as shown, the city is then added to the *world* database.

```
Choice: 3

Add New City
------------
Enter city name : Galway
Country Code : IRL
District : Connaught
Population : 80000
```

Figure 8 New city added

If the user enters an incorrect country code a suitable error message should be shown:

```
Choice: 3

Add New City
------------
Enter city name : Galway
Country Code : IRE
District : Connaught
Population : 80000
*** ERROR ***: CountryCode IRE does not exist
```

Figure 9 New city not added

- **4 (Find Car by Engine Size)**

  The user is asked to enter an engine size. All details of any cars in the *docs* collection in the imported Mongo database with that engine size are shown:

```
Choice: 4

Show Cars by Engine Size
------------------------
Engine Size : 1.3
2  |  11-LM-988  |  1.3  |  ['LM']
99  |  99=2=23  |  1.3
```

Figure 10 Cars with engine size = 1.3

- **5 (Add New Car)**

  The user is asked to enter an _id, reg and engine size:

```
Choice: 5

Add New Car
-----------
_ids : 9
Enter Reg : 12-G-12
Engine Size : 1.0
```

Figure 11 New Car added

These are then used to create a new document in mongodb as follows in the *docs* collection in the imported Mongo database:

```
{ "_id" : 9, "car" : { "reg" : "12-G-12", "engineSize" : 1 } }
```

Figure 12 Document created based on values entered

- **6 View Countries by Name**

  The user is asked to enter a country name, or part thereof.
  Any country that contains those letters should be displayed:

```
Choice: 6

Countries by Name
-----------------
Enter Country Name : ir
United Arab Emirates  ¦  Asia  ¦  2441000  ¦  Zayid bin Sultan al-Nahayan
C¶te d'Ivoire  ¦  Africa  ¦  14786000  ¦  Laurent Gbagbo
Ireland  ¦  Europe  ¦  3775100  ¦  Mary McAleese
Iran  ¦  Asia  ¦  67702000  ¦  Ali Mohammad Khatami-Ardakani
Iraq  ¦  Asia  ¦  23115000  ¦  Saddam Hussein al-Takriti
Kiribati  ¦  Oceania  ¦  83000  ¦  Teburoro Tito
Libyan Arab Jamahiriya  ¦  Africa  ¦  5605000  ¦  Muammar al-Qadhafi
Pitcairn  ¦  Oceania  ¦  50  ¦  Elisabeth II
Virgin Islands, British  ¦  North America  ¦  21000  ¦  Elisabeth II
Virgin Islands, U.S.  ¦  North America  ¦  93000  ¦  George W. Bush
```

Figure 13 Countries listed by Name

- **7 (View Countries by population)**
  The user is asked to enter in <, > or = and a number.
  If > and 800000000 were entered, the program would return all countries with a population of > 800000000.
  The same logic would apply for < and =.

```
Choice: 7

Countries by Pop
----------------
Enter < > or = : >
Enter population : 800000000
CHN  ¦  China  ¦  Asia  ¦  1277558000
IND  ¦  India  ¦  Asia  ¦  1013662000
```

Figure 14 Countries listed by population

- **x (Exit Application)**
  The program terminates

- **Anything Else**
  The menu is shown again.

NOTES

- After menu options 1 – 7 are selected, the menu is re-shown.

- For menu options 6 and 7 the information should be read from the database **only once.**
  E.g. If the user chooses 6 (View Countries by Name) or 7 (View Countries by Population) the countries are read from the database and stored in the program.

  If the user chooses 6 or 7 again, the information is **not** read from the database again. Instead, the information read the first time option 6 or 7 was chosen is used.