Higher Diploma in Science in Data Analytics	
Document Title:	Final Project - Applied Databases
Student No.:	G00364778
Student :	Gerhard van der Linde

Final Project - Applied Databases

MySQL.txt

4.1.1 Get people who have visited a particular country

MySQL_1.txt

```
    CREATE DEFINER=`root`@`localhost` PROCEDURE `get_ppl_visited_country`(land varchar(52))
        DETERMINISTIC
    BEGIN
        SELECT p.personID, p.personname, c.Name, v.dateArrived, y.Name FROM world.hasvisitedcity as v
        #SELECT * FROM world.hasvisitedcity as v
        left join world.city as c
        on c.ID=v.cityID
        left join world.person as p
        on p.personID=v.personID
        left join world.country as y
        on c.CountryCode=y.Code
        where y.Name like concat('%',land,'%')
        order by p.personname;
    END
```

Run the procedure

```
call get_ppl_visited_country('land');
```

Result

```
+----------+------------+-----------+-------------+-------------+
| 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 |
+----------+------------+-----------+-------------+-------------+
```

4.1.2 Rename Continent

MySQL_2.txt

```
CREATE FUNCTION `ren_continent`(original varchar(52)) RETURNS varchar(52)
    DETERMINISTIC
BEGIN
    if original in ('North America','South America')  then
        return 'Americas';
    elseif  original in ('Oceanoa') then
        return 'Australia';
    elseif original in ('Antarctica') then
        return 'South Pole';
    else
        return original;
    end if;
END
```

4.1.3 Country with biggest population per continent

MySQL_3.txt

```
SELECT c.Name, c.Continent, c.Population FROM world.country as c
where c.Population in (
    SELECT max(d.Population)  FROM world.country as d
    where d.Population > 0
    group by d.Continent
);
```

```
+--------------------+---------------+------------+
| Name               | Continent     | Population |
+--------------------+---------------+------------+
| Australia          | Oceania       |   18886000 |
| Brazil             | South America |  170115000 |
| China              | Asia          | 1277558000 |
| Nigeria            | Africa        |  111506000 |
| Russian Federation | Europe        |  146934000 |
| United States      | North America |  278357000 |
+--------------------+---------------+------------+
```

4.1.4 Minimum city population of youngest person(s)

MySQL_4.txt

```
SELECT c.Name, c.Population FROM world.city as c
where c.Population in (SELECT min(Population) FROM world.hasvisitedcity as v
    left join world.person as p
    on p.personId=v.personID
    left join world.city as c
    on c.ID=v.cityID
    where age=(SELECT min(age) FROM world.person)
);
```

```
+-----------+------------+
| Name      | Population |
+-----------+------------+
| Dordrecht |     119811 |
+-----------+------------+
```

4.1.5 Update City Populations

MySQL_5.txt

```
update world.city set Population =
case
    when District = 'Western Cape' then Population -10000
    when District = 'Eastern Cape' then Population +1000
    when District = 'Free State' then Population +2000
    else Population
end
where CountryCode like 'ZAF'
and District in ('Western Cape', 'Eastern Cape', 'Free State')
```

Before Query execution

```
+-----------------+--------------+------------+
| Name            | District     | Population |
+-----------------+--------------+------------+
| Port Elizabeth  | Eastern Cape |     742319 |
| East London     | Eastern Cape |     211047 |
| Uitenhage       | Eastern Cape |     182120 |
| Mdantsane       | Eastern Cape |     172639 |
| Bloemfontein    | Free State   |     314341 |
| Welkom          | Free State   |     183296 |
| Botshabelo      | Free State   |     157971 |
| Cape Town       | Western Cape |    2452121 |
| Paarl           | Western Cape |     205768 |
| George          | Western Cape |     193818 |
+-----------------+--------------+------------+
```

After Query execution

```
+-----------------+--------------+------------+
| Name            | District     | Population |
+-----------------+--------------+------------+
| Port Elizabeth  | Eastern Cape |     743319 |
| East London     | Eastern Cape |     212047 |
| Uitenhage       | Eastern Cape |     183120 |
| Mdantsane       | Eastern Cape |     173639 |
| Bloemfontein    | Free State   |     316341 |
| Welkom          | Free State   |     185296 |
| Botshabelo      | Free State   |     159971 |
| Cape Town       | Western Cape |    2442121 |
| Paarl           | Western Cape |     195768 |
| George          | Western Cape |     183818 |
+-----------------+--------------+------------+
```

4.1.6 Country Independence

MySQL_6.txt

```
SELECT Name, IndepYear, #year(now())-IndepYear as ilen, Population, GovernmentForm,
    case
        when IndepYear is null then 'n/a '
        when year(now())-IndepYear < 10 then
            concat(if(Population>100000000,"Large ",""), 'New ', GovernmentForm)
        when year(now())-IndepYear between 11 and 49 then
            concat(if(Population>100000000,"Large ",""),'Modern ', GovernmentForm)
        when year(now())-IndepYear between 50 and 100 then
            concat(if(Population>100000000,"Large ",""),'Early ', GovernmentForm)
        when year(now())-IndepYear > 100 then
            concat(if(Population>100000000,"Large ",""),'Old ', GovernmentForm)
    end as 'Desc'
```

```
      FROM world.country;
```

```
+--------------------------------+-----------+-------------------------------------------+
| Name                           | IndepYear | Desc                                      |
+--------------------------------+-----------+-------------------------------------------+
| Aruba                          |      NULL | n/a                                       |
| Afghanistan                    |      1919 | Early Islamic Emirate                     |
| Angola                         |      1975 | Modern Republic                           |
| Anguilla                       |      NULL | n/a                                       |
| Albania                        |      1912 | Old Republic                              |
| Andorra                        |      1278 | Old Parliamentary Coprincipality          |
| Netherlands Antilles           |      NULL | n/a                                       |
| United Arab Emirates           |      1971 | Modern Emirate Federation                 |
| Argentina                      |      1816 | Old Federal Republic                      |
| Armenia                        |      1991 | Modern Republic                           |
| American Samoa                 |      NULL | n/a                                       |
| Antarctica                     |      NULL | n/a                                       |
| French Southern territories    |      NULL | n/a                                       |
| Antigua and Barbuda            |      1981 | Modern Constitutional Monarchy            |
| Australia                      |      1901 | Old Constitutional Monarchy, Federation   |
| Austria                        |      1918 | Old Federal Republic                      |
| Azerbaijan                     |      1991 | Modern Federal Republic                   |
| Burundi                        |      1962 | Early Republic                            |

| Venezuela                      |      1811 | Old Federal Republic                      |
| Virgin Islands, British        |      NULL | n/a                                       |
| Virgin Islands, U.S.           |      NULL | n/a                                       |
| Vietnam                        |      1945 | Early Socialistic Republic                |
| Vanuatu                        |      1980 | Modern Republic                           |
| Wallis and Futuna              |      NULL | n/a                                       |
| Samoa                          |      1962 | Early Parlementary Monarchy               |
| Yemen                          |      1918 | Old Republic                              |
| Yugoslavia                     |      1918 | Old Federal Republic                      |
| South Africa                   |      1910 | Old Republic                              |
| Zambia                         |      1964 | Early Republic                            |
| Zimbabwe                       |      1980 | Modern Republic                           |
+--------------------------------+-----------+-------------------------------------------+
239 rows in set (0.00 sec)
```

Normalisation.doc

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 enroll* 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

Discussion

The database design above is not good.

- The student details are duplicated for every subject taken
- Students cannot enroll without selecting subjects as the moduleID field is a required field.
- Subject details are duplicated for every student taking the same subject.
- There is not way to limit the subject for a given year

A better design for above scenario would be a database with at a table containing student information and a table for subject information. Since all modules are not offered each year there should probably be a table to indicate what is available every year as well. Then lastly a table to indicate subject taken by students referenced from the subject and students tables.

Student Table

The following minimum columns should be in the student table.

studentID*	StudentName	dob
1	Sean	2000-10-03
2	Bill	1990-04-23
3	Tom	1973-12-10
4	Mary	1991-04-12

studentID*	StudentName	dob
5	Joe	1982-06-29

Possible extensions to this table.

- Enrollment status
- Progress levels, like undergraduate etc.

Module Table

At the minimum the modules table should include these fields

moduleID*	moduleName
100	Applied Databases
101	Java Programming
102	Computer Architecture
103	*Unavailabe Subject*
104	Mobile Apps

Possible extensions to the table

- prerequisites to the subject
- graduate levels etc
- module credits

Available Subjects by year

idx*	year	subject
1	2019	100
2	2019	101
3	2019	102
4	2019	104

This table simply lists subjects available for a given year by inserting a year and subject id into the table. This information can the be used for the selection criteria query when subjects are selected by students for a given year.

Sujects selected by student

idx*	Student	Subject
1	1	100
2	2	100
3	3	101
4	3	104
6	4	102
7	5	100

idx*	Student	Subject
8	5	104

The student and subject fields will have foreign key constraints applied to ensure only valid students and subjects are entered in the table. Another constraint on entering data in the table might be to check the availability criteria of the subject chosen.

The last table simply list the selections by student ID against subject ID. It should probably have a flag column to indicate when a subject was completed. Another possible field to add would be credits obtained on completion.

Possible table extensions:

- Subject completed
- Credits obtained

MongoDB.txt

Import the database to collection docs.

```
mongoimport --db proj --collection docs --type json --file
C:\Users\%USERNAME%\Documents\52553\mongo.json
```

MongoDB.txt

```
        // 4.3.1 Average Engine Size
        db.docs.aggregate({$group:{_id:null,Average:{$avg:"$car.engineSize"}}})

        //4.3.2 Categorise County Populations
        db.docs.aggregate([
            {$bucket:{
                groupBy:"$pop",
                boundaries:[0,50000,100000,150000],
                default:"Other",
                output:{
                    "counties":{$push:"$name"}
                }
              }
            }
        ])

        // 4.3.3 Redefine County Populations
         db.docs.aggregate(
          [
            {
              $match:
                {
                  pop:{$exists:true}
                }
            },
            {
              $project:
                {
                  name:1,
                  pop:
                    {
                      $cond:{if:{$lte:["$pop",100000]},
                        then: "Small County", else:"Big County"}
                    }
                }
            }
          ]
         )
```

References:
MongoDB $buckets aggregation [1)]
MongoDB $cond aggregation [2)]

Python

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

Python.py

```
'''
This is a pythin based console application that connects to background
Database MySQL and MongoDB application to perform various display and
update functions from a menu driven console interface.

The application depends on a local MySQL and MongoDB database to be up
and running.

The application also requires some non standard pythoin libraries to be
installed. Check the dependancies from the console menu "c" or run the
check_dependansies() command from the console.

Run the application from a python or command console.

'''
debug = False
country_data_loaded = False
mongoclient=None
df=None
import pymysql
import pymongo
#import collections
from collections.abc import MutableMapping
from terminaltables import AsciiTable
from pkgutil import iter_modules
import keyboard
import os
import sys
import re
import pandas as pd

# Main function
def main():
    '''
    Display the menu and execute the choices returned from the user selection menu
    '''
    global mongoclient
    display_menu()

    while True:
        choice = input("Enter choice: ").strip()
        if (choice == "1"):
            view_15_cities()
            display_menu()
        elif (choice == "2"):
            view_cities_by_population()
            display_menu()
        elif (choice == "3"):
            add_new_city()
            display_menu()
        elif (choice == "4"):
            find_car_by_enginesize()
            display_menu()
        elif (choice == "5"):
            add_new_car()
            display_menu()
        elif (choice == "6"):
            view_countries_by_name()
            display_menu()
        elif (choice == "7"):
            view_countries_by_population()
            display_menu()
        elif (choice == "c"):
            check_dependansies()
            display_menu()
        elif (choice == "x"):
            # gracefully terminate the mongo client connection
            # mongoclient.close()
            break
        else:
            display_menu()

def view_15_cities():
    while True:
        '''
        Clear the screen between menu selections
        '''
        clear()
        #print('View 15 Cities')
        query = "select * from city limit 15;"
        print('\rrunning query, please wait ..... ',end='')
        result = mysql_query(query)
        #clear()
        print('\rView 15 Cities                   ')
        print_nice(result)
        wait_here()
        return True

def view_cities_by_population():
    '''
    This fiunction is called from the main console menu.

    Cities by population connects the the MySQL database and executes the query
    after appending the where clause returned by the 'add_where_clause' function.

    The returned query result then uses another function 'print_nice' to create
    an ascii table style output to the console of the query result returned and waits
    for the spacebar key beore proceeding back to the menu.
    '''
    clear()
    print('Cities by population\nCreate population filter')
    wc=add_where_clause()
    query = "SELECT * FROM world.city as c " + wc + " order by c.Population"
    print('\rRunning Cities by population query ...',end='')
    result = mysql_query(query)
    print('\rCities by population                  ')
    print_nice(result)
    wait_here()
    return True

def add_new_city():
    '''
    This function is called from the main console menu.

    add_new_city prompts for user data input and adds a new city the the mysql
    database.
    '''
    print('Add new city')
    citydata = prompt_city_data()
    mysql_add_city_data(citydata)
    wait_here()
    return True


def find_car_by_enginesize():
    '''
    This function is called from the main console menu.

    find_car_by_enginesize prompts the user for input and creates a query using the imput
    to show the enginesizes in an flattened ascii table returned by the mongodb query.

    The function uses the following internal function calls to complete and parse the query data.

        mongo_connect()
        mongo_to_list()
        print_nice()
        wait_here()

    '''
    global mongoclient
    Valid=False
    while Valid == False:
        size = input("Enter enginesize (eg 1.5 or * for all) : ").strip()
        if size == '*':
            Valid=True
        elif len(size)>=1:
            enginsize=float(size)
            if (enginsize > 0.8 and enginsize < 5.0):
```

```
                Valid = True
        if size == '*':
            query={'$and':[{'car':{'$exists':'true'}}]}
        else:
            query={'$and':[{'car':{'$exists':'true'}},{'car.engineSize':enginsize}]}
        print('\rProcessing query ....',end='')
        mongoclient=mongo_connect(mongoclient)
        print('\rCar by enginesize     ')
        #cars=mongo_find(mongoclient,'proj','docs',{"car":{"$exists":"true"}})
        cars=mongo_find(mongoclient,'proj','docs',query)
        if debug==True:
            for car in cars:
                print(car)
        car_list=mongo_to_list(cars)
        print_nice(car_list)
        wait_here()
        return True

    def add_new_car():
        '''
        This function is called from the main console menu.

        add_new_car creates a new enrt in the mongodb database from the user input provided
        in the console prompts. The prompt input is collected and returned by a sub-function.

        The followjng sub-functions are called from here:

            add_new_car_get_data()
            mongo_add_data()
            wait_here()

        The sub function details discussed in their own space.
        '''
        print('Add new Car')
        cardetails=add_new_car_get_data()
        id,reg,cc=cardetails
        id=float(id) # in line with exixsting variable types for cars??
        # print(id,reg,cardetails)
        db="proj"
        collection="docs"
        # newDoc = {"_id":7, "car":{"reg":"99-D-69674", "enginesize":1.0}}
        newDoc = {"_id":id, "car":{"reg":reg, "enginesize":cc}}
        mongodb_add_data(db,collection,newDoc)
        wait_here()
        return True

    def view_countries_by_name():
        '''
        This function is called from the main console menu

        This function queries the MySQl databases and return all the data in the
        world.country table to a pandas dataframe and then processes all subsequent
        request for data from the dataframe.

        To load an up to date copy for any queries related to the table the console
        application must be terminted and restarted or set the global variable
        country_data_loaded to False and the next call to any of the queries will
        reload the data from the database.

        This function depends on the following sub functions:
            country_data_to_df()
            reduce_df_to_header_list()
            AsciiTable() - external library
            wait_here()

        The table outputs the list of countries filtered by the country name or partial
        name entered at the user console inputs.

        '''
        print('Countries by name')
        if country_data_loaded == False:
            country_data_to_df()
        else:
            print('Data aready loaded')
        cname = input('Enter the full/partial country name: ').strip()
        case = input('Case sensitive? (True/False): ').strip().capitalize()
        if case.startswith('True'):
            case = True
        elif case.startswith('False'):
            case = False
        else:
            case = False
        datalist = reduce_df_to_header_list('Name', cname, 'str', case)
        table=AsciiTable(datalist)
        print(table.table)
        wait_here()
        return True

    def view_countries_by_population():
        '''
        This function is called from the main console menu

        This function queries the MySQl databases and return all the data in the
        world.country table to a pandas dataframe and then processes all subsequent
        request for data from the dataframe.

        To load an up to date copy for any queries related to the table the console
        application must be terminted and restarted or set the global variable
        country_data_loaded to False and the next call to any of the queries will
        reload the data from the database.

        This function depends on the following sub functions:
            country_data_to_df()
            reduce_df_to_header_list()
            AsciiTable() - external library
            wait_here()

        The table outputs the list of countries filtered by the country population criteria
        entered at the user console inputs.

        '''
        print('Countries by population')
        if country_data_loaded == False:
            country_data_to_df()
        else:
            print('Data aready loaded')
        pfilter=input('Enter a population filter (eg. <=1000): ').strip()
        if pfilter.strip().startswith('<') or pfilter.strip().startswith('>'):
            pfilter=pfilter
        elif pfilter.strip().startswith('='):
            if pfilter.strip().startswith('=='):
                pfilter=pfilter
            else:
                pfilter='='+pfilter
        elif pfilter.isalnum:
            pfilter='=='+pfilter
        datalist = reduce_df_to_header_list('Population', pfilter, 'val', False)
        table=AsciiTable(datalist)
        print(table.table)
        wait_here()
        return True

    def check_dependansies(mode='show_missing'):
        '''
        This function is an extra function called from the console menu.

        The purpose of the function is to verify that all dependant python libraries are installed
        and available that is required to run this application.

        mode options:
           show_missing: show only missisng items
          show_required: show a list of required modules

        This function is dependant on the followin submodules

            module_exists()

        '''
        module_list=['pymysql','pymongo', 'terminaltables', 'keyboard', 'pandas', 'collections', 'pkgutil']
        if mode == 'show_missing':
            some_missing=False
            print('Running Application dependancy checker\n')
            for module in  module_list:
                exist=module_exists(module)
                if exist == False:
                    some_missing=True
                    print('\tMissing: {}'.format(module))
            if some_missing==True:
```

```
            print('\n Please install missing components first')
        else:
            print('No missing modules')
    elif mode == 'show_required':
        print('Required modules:')
        for module in module_list:
            print('\t',module)
    wait_here()
    return True

def display_menu():
    '''
    This function generates the console menu for the console application
    and is called from the main() function.
    '''
    clear()
    print("World DB")
    print("--------")
    print("")
    print("MENU")
    print("=" * 4)
    print("1 - View 15 Cities")
    print("2 - View Cities by population")
    print("3 - Add New City")
    print("4 - Find car by enginesize")
    print("5 - Add New Car")
    print("6 - View Countries by name")
    print("7 - View Countries by population")
    print("c - Check dependancies")
    print("x - Exit")

def module_exists(module_name):
    '''
    module_exixts is a sub function iterating through a list of names passed to the function
    and simply returning True or false if able to determine of the module name is available
    to be called or loaded.

    parameters passed into the module is a python library name.
    '''
    return module_name in (name for loader, name, ispkg in iter_modules())

def reduce_df_to_header_list(columnName, filterstr, filtertype='str',caseSenstive=True):
    '''
    this sub-function references the globally decaled dataframe df and creates a subset from
    the complete dataset in the dataframe based on the filter criteria passed into the function.

    The function call accepts four parameters and the last two is optional.

        columnName:   - See list below
        filterstr:    - examples: 'Ire' for str types or '<1000' for val etc
        filtertype:   - 'str' or 'val'
        caseSenstive: - True or False

    Available column names are:
        (['Capital', 'Code', 'Continent', 'GNP', 'GovernmentForm', 'HeadOfState',
        'IndepYear', 'LifeExpectancy', 'LocalName', 'Name', 'Population',
        'Region', 'SurfaceArea']

        The list of names is generated calling the pandas command df.columns

        The rest of the code the creats a python list of values with a header row and
        rows of data ready to create user friendly asccii tables and returns this data
        to the calling function in a python list format.

    '''
    if filtertype.lower().__contains__('str'):
        filtered=df[df[columnName].str.contains(filterstr,case=caseSenstive)]
    elif filtertype.lower().__contains__('val'):
        #filtered = df[df[columnName]<1000]
        filtered=df[eval('df[columnName]'+filterstr)]
    # extract headings
    header=list(filtered)
    # extract rows
    rows=filtered.values.tolist()
    dat=[]
    dat.append(header)
    for row in rows:
        dat.append(row)
    return dat
```

```
def country_data_to_df():
    '''
    This sub-function is not called directly but rather called from
    view_counries_by_population() and view_countries_by_name().

    This routine fundamentally calls routines to connect to the MySQL
    database and load the country  data into a dataframe for subsequent
    refinement and interrogation and the dataframe is active and accessible
    globally for the duration of the python session.

    This function is dependant on:
        load_country_data()

    The function take the query result and converts it to a dataframe df that
    is accessible globally.

    '''
    global df
    global country_data_loaded
    print('\rLoading country data to memory...', end='')
    countries=load_country_data()
    df=pd.DataFrame(countries)
    country_data_loaded=True
    print('\rCountry data loaded to memory    ')

def load_country_data():
    '''
    This sub-function executes a mysql query on the database and returns the
    query result or en error to the calling routine.
    '''
    #load and store country data in memory for functions 6 and 7 calls
    try:
        country_data=mysql_query("select * from country")
    except Exception as e:
        print(e)
    else:
        return country_data


def mysql_add_city_data(citydata):
    '''
    This routine creates a database connection and inserts new city data
    passed to the function into the mysql word.city database.

    The data passed into the routine is a python list that requires four parameters
    passed into the routine:

        Name         - Name of the new city added
        CountryCode  - A valid country code, if not valid the entry will fail
        District     - The name of the district or county
        Population   - The population of the city added

    The function call on completion will return a success or failure message.

    '''
    conn = pymysql.connect( "localhost", "root", "root", "world",
                    cursorclass=pymysql.cursors.DictCursor)
    ins = "Insert INTO city (Name, CountryCode, District, Population) VALUE(%s, %s, %s, %s)"

    with conn:
        try:
            cursor = conn.cursor()
            cursor.execute(ins, (citydata[0], citydata[1], citydata[2], citydata[3]))
            conn.commit()
            print("Insert successful")
        except Exception as e:
            print("Insert failed! Invalid county entered", e)


def prompt_city_data():
    '''
    This sub-routine simply creates prompts for user input for data to create the new city
    with. It also stips white space and capatalise to ensure consistency and data integrity.
    When all the data is collected, its added to a sigle list variable and returned to the
    calling function

    '''
    # Name, CountryCode, District, Population, latitude, longitude
    clear()
    print('Enter the values in at the prompts adding a new City to the city database\n')
    Name = input('City Name: ').strip().capitalize()
    CountryCode = input('Country Code: ').strip().upper()
    District = input('District/County: ').strip().capitalize()
    Population = int(input('Population: ').strip())
    citydata=[Name, CountryCode, District, Population]
    if debug==True: print(citydata)
    return citydata

def mongodb_add_data(db,collection,newdoc):
    '''
    The routine adds data to the MongDB database passed into the function in newdoc.
    The function call expects three variable to be populated.

        db          - is the name of the MongoDB database to use
        collection  - collection is the name of the collection in the database
        newdoc      - newdoc is the datastring in the format specified below

    newDoc = {"_id":7, "car":{"reg":"99-D-69674", "enginesize":1.0}}

    The function call will return a success or failure to inser the data

    '''
    # connect if not already connected, otherwise skip and use current connection
    global mongoclient
    mongoclient=mongo_connect(mongoclient)

    db = mongoclient[db] #db = mongoclient["proj"]
    docs = db[collection]#docs = db["docs"]
    #newDoc = {"_id":7, "car":{"reg":"99-D-69674", "enginesize":1.0}}

    try:
        docs.insert_one(newdoc)
    except pymongo.errors.DuplicateKeyError:
        print('A duplicate key was entered, please try again.')
    except Exception as e:
        print(e)
    else:
        print('Successfully added the new car')


def add_new_car_get_data():
    '''
    This function creates the user prompts, collects and formats the data and assemble
    the results into a list and returns it to the calling function for adding a new car
    to the database.

    The function prompts the user for three values, a new id, the car reg and the enginesize.

    '''
    print('\nPlease enter details for new car to add\n')
    _id=input('_id: ').strip()
    carreg=input('car reg(eg:99-D-123): ').strip().upper()
    if carreg.find('-') < 0: #ife there is no dashes in the reg
        carreg = re.sub(r'([A-Za-z]+)',r'-\1-',carreg.upper())# add dashes to the reg
    enginesize=float(input('engine size(eg: 1.6): ').strip())
    #print('{} {} {}'.format(_id,carreg,enginesize))
    cardetails=[_id, carreg, enginesize]
    return cardetails


def flatten_dict(d, parent_key='', sep='_'):
    '''
    This function performs an intermediate step on the mongoDB data query result
    by flattening the json file structure returned by the mongoDB query to facilitare
    the printing of the query results in a user friendly tabular format.

    '''
    items = []
    for k, v in d.items():
        new_key = parent_key + sep + k if parent_key else k
        if isinstance(v, MutableMapping):
            items.extend(flatten_dict(v, new_key, sep=sep).items())
        else:
            items.append((new_key, v))
    return dict(items)

def mongo_to_list(mongo_cursor):
    '''
    iterate over the raw mongo cursor return and flatten the dictionary like format
    to a python list to facilitate user friendly tabular style prints
    '''
    to_list=[]
    for item in mongo_cursor:
        to_list.append(flatten_dict(item))
    return to_list

def mongo_connect(mongoclient):
    '''
    Conneect to the mongoclient of not already connected and return the connection reference
    '''
    if (not mongoclient):
        try:
            mongoclient = pymongo.MongoClient()
            mongoclient.admin.command('ismaster')
            if debug==True: print('client_connect: ',mongoclient)
        except Exception as e:
            print('Error', e)
    return mongoclient

def mongo_find(mongoclient,db,collection,query):
    '''
    find data in the mongodb using the parameters passed into the function and return
    the query data from the function call.

    Parameters passed in:
        mongoclient  - client info passed on from the db connect function call
        db           - the database to connect to
        collection   - the collection to query
        query        - mongoDB style query eg: {'car':{'$exists':'true'}}

    '''
    db = mongoclient[db]
    docs = db[collection]
    query = query
    query_result = docs.find(query)
    if debug == True:
        prdata=query_result.copy()
        for line in prdata:
            print(line)
    return query_result

def add_where_clause():
    '''
    This function takes user input and creates a where clause for a mysql query
    from the input prompts and returns the where clause to the calling function
    '''
    signs=['<','>','=']
    Valid=False
    while Valid == False:
        sign = input("Enter < > or = : ").strip()
        if sign in signs:
            Valid = True
    value = input("Enter population : ").strip()
    whereclause = 'where c.Population {} {}'.format(sign,value)
    return whereclause

def wait_here():
    '''
    This function creates a wait step anywhere in the application where required and
    display a messages that it is waiting untill the space bar is pressed.
    '''
    print("Press space to continue ...")
    keyboard.wait('space')

def clear():
    '''
    This clears the terminal output for linux or windows systems
    '''
    os.system( "cls" if os.name == "nt" else "clear")


def mysql_query(query):
    '''
    This function connects to the locally running mysql server assuming host and
    user credentials and execute the MySQL query passed into the function and return
    the query results in a python list structure to the calling function
    '''
    conn = pymysql.connect( "localhost", "root", "root", "world",
                    cursorclass=pymysql.cursors.DictCursor)
    with conn:
        cursor = conn.cursor()
        cursor.execute(query)
        results = cursor.fetchall()
    conn.close()
    return results

def print_nice(QueryData):
    '''
    This function takes the data returned from a mysql query and generates an ascii
    table from the data and output the result to the console window

    The input to the function call is the raw data returned from the sql query function
    mysql_query().

    '''
    #print(QueryData)
    if len(QueryData) == 0:
        print('No data returned!')
    else:
        heading=[]
        data=[]
        for txt in QueryData[0]:
            heading.append(txt)
        data.append(heading)
        for line in QueryData:
            vals=[]
            for idx,val in line.items():
                vals.append(val)
            data.append(vals)
        table=AsciiTable(data)
        print(table.table)

if __name__ == "__main__":
    '''
    The main function where everyting starts from end ends
    '''
    # execute only if run as a script
    #debug = True
    main()
```

Innovation.doc

The major innovation for this module is consolidation the entire project submission into a code friendly markup language driven document that is context sensitive to MySQL, mongoDB, JSON and python code syntax. The stylesheets behind the documenting system was customized to include JSON and MongoDB support and matching syntax color set in the style sheets behind the code using knowledge gained from the Web development module for this semester.

The document is generated in HTML, exported as PDF using stylesheets for export and converted into a word document for reference an portability. See the Two documents attached for the project.

Python code related innovations.

Some minor innovation were applied to the python code part of the project.

The first part is in the men code barely noticeable except if the queries are slow to execute.

The message will be displayed “Running query, please wait ....” and on completion of the query the line will be erase and replace with a new message, for example “View 15 cities”. This is achieved by applying two parameters to the print statement.

```
print('\rrunning query, please wait ..... ',end='')
result = mysql_query(query)
print('\rView 15 Cities                   ')
```

This ocde above causes the firts message to be displayed as long as the query rungs and raplaces that with the second message when the query completes.

The second innovation in the python code is the printing of all the query results in Ascii tabular format similar to the native output formats for MySQL. This output format is applied for both MySQL as well as mongoDB tables.

```
View 15 Cities
+----+----------------+-------------+---------------+------------+----------+-----------+
| ID | Name           | CountryCode | District      | Population | latitude | longitude |
+----+----------------+-------------+---------------+------------+----------+-----------+
| 1  | Kabul          | AFG         | Kabol         | 1780000    | None     | None      |
| 2  | Qandahar       | AFG         | Qandahar      | 237500     | None     | None      |
| 3  | Herat          | AFG         | Herat         | 186800     | None     | None      |
| 4  | Mazar-e-Sharif | AFG         | Balkh         | 127800     | None     | None      |
| 5  | Amsterdam      | NLD         | Noord-Holland | 731200     | None     | None      |
| 6  | Rotterdam      | NLD         | Zuid-Holland  | 593321     | None     | None      |
| 7  | Haag           | NLD         | Zuid-Holland  | 440900     | None     | None      |
| 8  | Utrecht        | NLD         | Utrecht       | 234323     | None     | None      |
| 9  | Eindhoven      | NLD         | Noord-Brabant | 201843     | None     | None      |
| 10 | Tilburg        | NLD         | Noord-Brabant | 193238     | None     | None      |
| 11 | Groningen      | NLD         | Groningen     | 172701     | None     | None      |
| 12 | Breda          | NLD         | Noord-Brabant | 160398     | None     | None      |
| 13 | Apeldoorn      | NLD         | Gelderland    | 153491     | None     | None      |
| 14 | Nijmegen       | NLD         | Gelderland    | 152463     | None     | None      |
| 15 | Enschede       | NLD         | Overijssel    | 149544     | None     | None      |
+----+----------------+-------------+---------------+------------+----------+-----------+
Press space to continue ...
```

```
Car by enginesize
+------+---------------+----------------+-------------------+
| _id  | car_reg       | car_engineSize | addresses         |
+------+---------------+----------------+-------------------+
| 1.0  | 191-G-123     | 1.5            | ['G', 'WH']       |
| 2.0  | 11-LM-988     | 1.3            | ['LM']            |
| 3.0  | 142-G-28      | 1.0            | ['MO', 'G', 'WH'] |
| 6.0  | 152-MO-134    | 1.5            | ['MO']            |
| 5.0  | 05-D-1234     | 1.4            | ['D', 'G']        |
| 7    | 99-D-69674    | 1.0            |                   |
| 9    | 09-WW-5475957 | 1.8            |                   |
+------+---------------+----------------+-------------------+
Press space to continue ...
```

Input data entere by users are optomised and checked where possible, caoatalised and for example when a car reg is entered in lower case without dashes they're capatailised and deshes added.

I also added function “c” to check for dependant modules required to run the code. So if any of the include libraries are not installed they will be higligthed as missing.

[1] https://docs.mongodb.com/manual/reference/operator/aggregation/bucket/

[2] https://docs.mongodb.com/manual/reference/operator/aggregation/cond/

Last update: **2019/05/11 11:25**