Portfolio 2 task: Data Entry and Data Statistics (Medium difficulty)

A database is more than just a storage area for data. We can also use the database query language to extract additional information of interest from the raw data. This task involves creating an SQL database containing data about cars with customers' ratings and developing some Python functions that extract additional information from the database not readily discernible from the raw data alone.

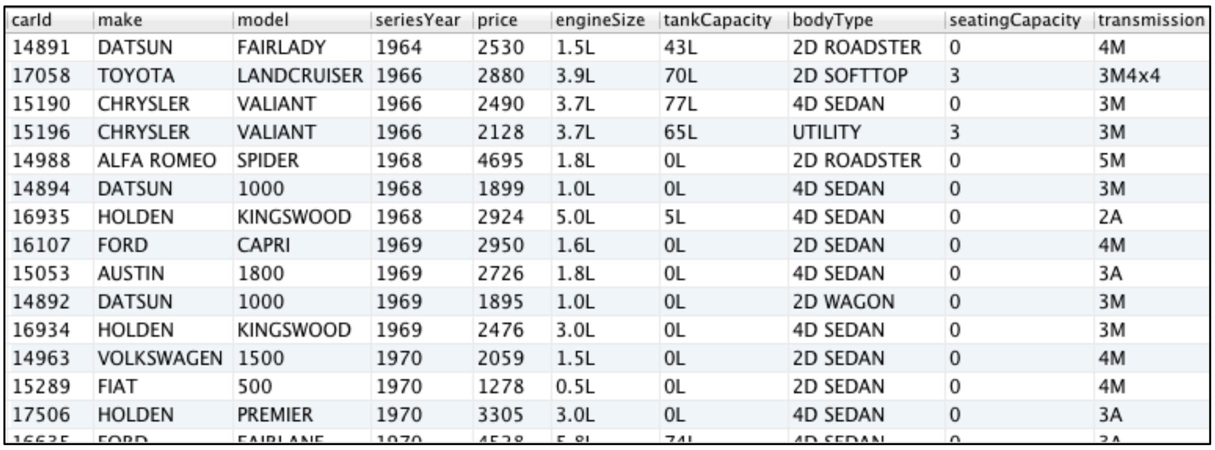
To get started we have some large text files containing (a) basic details about various models of cars and (b) the ratings given to specific cars by their owners. The data is currently stored in two text files, `car_details.txt` and `car_ratings.txt`. You are required to write some Python functions to generate certain statistics. To do so, you need to (1) create an SQL database, (2) write a Python function to enter the vehicle and rating data into the database, and (3) write some Python functions to extract and display the required statistical information.

Creating the Database

Before you begin this task you need to create a database called `car_reviews` containing two tables, `car_details` and `car_ratings`. (The data that will be entered into these tables is in two text files, `car_details.txt` and `car_ratings.txt`.) Each of these tables has many columns, so to save you the trouble of creating them all manually, we have provided an SQL script, `create_car_reviews_schema.sql`, which will set up the necessary database and table schemas. You should first run this script in the *MySQL Workbench* to set up the database needed for the remainder of the task.

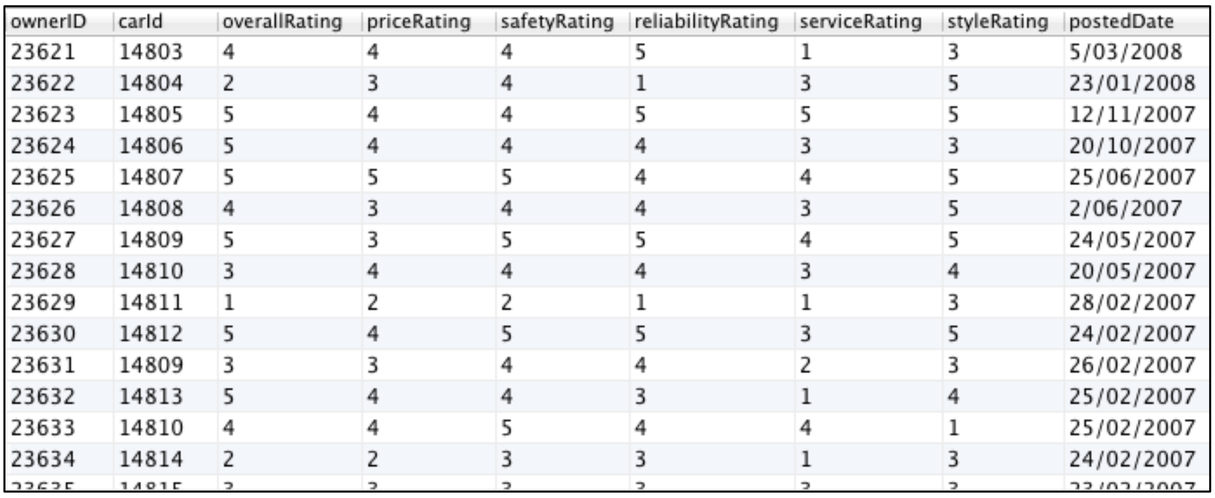
The Database's Contents

Running the script described above will create empty tables in your database server. However, when these are populated by your Python program they will contain a large amount of data as illustrated here. The `car_details` table will contain 3,119 rows like the following. Each describes basic features of different models of car. The primary key is `carId` by default.



carId	make	model	seriesYear	price	engineSize	tankCapacity	bodyType	seatingCapacity	transmission
14891	DATSUN	FAIRLADY	1964	2530	1.5L	43L	2D ROADSTER	0	4M
17058	TOYOTA	LANDCRUISER	1966	2880	3.9L	70L	2D SOFTTOP	3	3M4x4
15190	CHRYSLER	VALIANT	1966	2490	3.7L	77L	4D SEDAN	0	3M
15196	CHRYSLER	VALIANT	1966	2128	3.7L	65L	UTILITY	3	3M
14988	ALFA ROMEO	SPIDER	1968	4695	1.8L	0L	2D ROADSTER	0	5M
14894	DATSUN	1000	1968	1899	1.0L	0L	4D SEDAN	0	3M
16935	HOLDEN	KINGSWOOD	1968	2924	5.0L	5L	4D SEDAN	0	2A
16107	FORD	CAPRI	1969	2950	1.6L	0L	2D SEDAN	0	4M
15053	AUSTIN	1800	1969	2726	1.8L	0L	4D SEDAN	0	3A
14892	DATSUN	1000	1969	1895	1.0L	0L	2D WAGON	0	3M
16934	HOLDEN	KINGSWOOD	1969	2476	3.0L	0L	4D SEDAN	0	3M
14963	VOLKSWAGEN	1500	1970	2059	1.5L	0L	2D SEDAN	0	4M
15289	FIAT	500	1970	1278	0.5L	0L	2D SEDAN	0	4M
17506	HOLDEN	PREMIER	1970	3305	3.0L	0L	4D SEDAN	0	3A
16635	FORD	FAIRLANE	1970	4528	5.8L	74L	4D SEDAN	0	3A

Once populated the `car_ratings` table will contain 5,504 rows as follows. It contains the results of a survey of car owners about their particular car. The primary key is `ownerID` and `carId` is a foreign key (and is not necessarily unique in the `car_ratings` table's `carId` column). In particular, note that if you want information about the ratings given to particular makes or models of cars you will need to *join* data from both of these tables.



ownerID	carId	overallRating	priceRating	safetyRating	reliabilityRating	serviceRating	styleRating	postedDate
23621	14803	4	4	4	5	1	3	5/03/2008
23622	14804	2	3	4	1	3	5	23/01/2008
23623	14805	5	4	4	5	5	5	12/11/2007
23624	14806	5	4	4	4	3	3	20/10/2007
23625	14807	5	5	5	4	4	5	25/06/2007
23626	14808	4	3	4	4	3	5	2/06/2007
23627	14809	5	3	5	5	4	5	24/05/2007
23628	14810	3	4	4	4	3	4	20/05/2007
23629	14811	1	2	2	1	1	3	28/02/2007
23630	14812	5	4	5	5	3	5	24/02/2007
23631	14809	3	3	4	4	2	3	26/02/2007
23632	14813	5	4	4	3	1	4	25/02/2007
23633	14810	4	4	5	4	4	1	25/02/2007
23634	14814	2	2	3	3	1	3	24/02/2007
23635	14815	3	3	3	3	3	3	23/02/2007

Developing the Python Program

As usual we have supplied a Python 'question' template file for you to complete, containing unit tests that your functions must pass. You should rename this file, removing the "`_Q`" suffix, and complete your solutions in the place indicated.

In summary, you need to develop four Python functions to pass the unit tests. Function `populate_table` copies data from the provided text files into the SQL database tables. Functions `best_and_worst`, `most_expensive` and `average_ratings` then interrogate the database and print the results.

Specific Tasks

The specific things you must do to complete this portfolio task are as follows.

1. Create a database schema named `car_reviews` containing two tables called `car_details` and `car_ratings`, as explained above, to enter the data into. This can be done easily using the supplied SQL script and the *MySQL Workbench*.
2. Write a Python function called `populate_table`. The function takes as its parameter the name of the table, which is also the name of the input file containing raw data. For instance, for the table `car_details`, the corresponding text file is `car_details.txt`. The text file consists of tab-separated fields. (The tab character is represented as '`\t`' in Python strings.) The `populate_table` function must read the data from the named text file and insert it into the corresponding table using an appropriate SQL statement. After all rows have been entered, the function must return the number of rows in the table. (You could use an SQL "count" query to do this.) See the unit tests in question file `data_statistics_Q.py` for how the function is used.

   Importantly, while you are developing your solution, you will need to call this function many times and you don't want it to try putting the same data into the table more than once. Therefore, before inserting any data your `populate_table` should execute an SQL '`TRUNCATE`' statement to remove all data from the relevant table, otherwise you will get error messages from the SQL server about duplicate entries.
3. Write a Python function named `best_and_worst` that prints the range of "overall ratings" given to the cars with a specified make. For example, if the maximum overall rating and minimum overall rating given to all Volvos in the database are 5 and 2, respectively, your function should produce the following result when called.

   ```
   >>> best_and_worst('VOLVO')
   VOLVO (2-5)
   ```

   Note that to do this you need to combine information from both of the tables. Only the `car_details` table includes the makes of car by name and only the `car_ratings` table has the owners' overall ratings. Get the necessary SQL query correct using the *MySQL Workbench* first and then incorporate it into your Python program. (Hint: SQL's `max` and `min` functions will be helpful here.)

   If there are no details for the specified make of car in the database an appropriate error message must be printed. (Hint: A non-existent SQL row or field will be represented by the special Python value `None` when you try to fetch it.)

   See the unit tests in question file `data_statistics_Q.py` for further examples.

4. Write a Python function named `most_expensive` that takes a number and prints the make, model, price, and overall rating of the most expensive cars in the database, one car per line. (Here we are referring to specific cars, not the model of car in general.) The number determines how many lines to print. The output should be ordered in descending order by price first, then in ascending order by car make and then car model, and finally in descending order by overall rating. Moreover, the output should not contain duplicates, i.e., any two lines in the output can't be exactly the same.

   For example, if the top three most expensive cars are `FERRARI 575M` with price `$551,000` and overall rating 4, `FERRARI 599` with price `$551,000` and overall rating 5, and `MERCEDES-BENZ E55` with price `$225,600` and overall rating 5, the output of your function should be as follows.

```
>>> most_expensive(3)
FERRARI 575M: $551000, Overall rating 4
FERRARI 599: $551000, Overall rating 5
MERCEDES-BENZ E55: $225600, Overall rating 5
```

   See the unit tests in question file `data_statistics_Q.py` for further examples.

5. Write a Python function named `average_ratings` that takes a make of car and prints the average "overall rating" of each model of that make, one model per line. The average overall rating should be rounded to one decimal place. For example, there are 11 models of Honda in the database, with average ratings for each model as shown below.

```
>>> average_ratings('HONDA')
LEGEND 5.0
MDX 4.7
INTEGRA 4.6
PRELUDE 4.4
ODYSSEY 4.4
JAZZ 4.3
ACCORD 4.3
CIVIC 4.1
CITY 4.0
CRV (4x4) 4.0
HRV (4x4) 4.0
```

   (Hint: You will need to "group" the results by model so that the average rating is calculated for each model separately. See the SQL `GROUP` option in the `SELECT` statement.) Again, if there are no details for the specified make of car in the database an appropriate error message must be printed.

   See the unit tests in question file `data_statistics_Q.py` for further examples.

Development hints

- Before you can begin this task you must ensure that you have access to MySQL software and the *MySQL Workbench* so that you can create the database. You must also have access to an appropriate MySQL-Python connector module so that you can call MySQL functions from Python code. This will be either MySQLdb (for Windows users) or the MySQL Connector (for Mac users). See the Week 7 and 8 workshops and the *INB104 Software Handbook* for more detail on installing this software on your own machine or for accessing it in the S-Block labs.

- You need to write one Python program which contains (at least) 4 functions (i.e., `populate_table`, `best_and_worst`, `most_expensive`, and `average_ratings`) to enter data into and retrieve data from the database. The three data retrieval functions are independent of each other, so can be completed in any order. However, you need to complete the `populate_table` function first to fill the tables with data.
- As noted above, you should use the SQL '`TRUNCATE`' statement in your `populate_table` task to remove all old data from the relevant table before you try to enter any new data. This ensures that when you repeatedly call this function during program development you don't get 'duplicate entry' errors.
- Don't try to do everything at once. You need to write Python code and SQL queries, so do these one at a time. After you have created and populated the database, work out what SQL queries you need using the *MySQL Workbench*. Once you have these correct, then you can incorporate them into your Python program.

Deliverables

The deliverables for this task are as follows.

1. A complete Python program `data_statistics.py` that achieves the desired functionality. This should be developed from the 'question' template file `data_statistics_Q.py` but with the '`_Q`' suffix removed. It must pass the unit tests in the original `data_statistics_Q.py` file. Your program must contain the original set of unit tests **without modification** and must be ready to run.
2. A completed "statement" at the beginning of the Python file to confirm that this is your own work and to identify your programming pair. (We can't award you marks if we don't know who you are!) Also you must indicate the percentage contribution made to the whole portfolio by both students. We do *not* expect a precise 50/50 split; an accurate and honest assessment is appreciated. This percentage will not affect your marks except in cases of *extreme* imbalances or disagreements.

Apart from working correctly your program code must be well-presented and easy to understand, thanks to (sparse) commenting that explains the *purpose* of significant code segments and *helpful* choices of variable and function names.

You do *not* need to submit any SQL database scripts or dumps. We will use our own SQL database to test your software.