Lab Sheet 2

More SELECT-FROM-WHERE and Some Data Manipulation Queries

In this lab we will further explore the SELECT-FROM-WHERE queries but using a different database. We will also experiment with SQLs data manipulation capabilities.

SQLite supports a few useful commands that we can execute just as if they were ordinary queries (i.e. by right clicking and selecting "Run Query"). Two useful ones are:

- `.tables` which lists the tables in the current database
- `.scheme tablename` which displays a summary of the columns
- `.help topic` displays a one-line summary of topic (e.g. 'backup' or 'restore').

SELECT Queries with Another DB

We will use a fresh database named `countries.sqlite` that contains some basic statistical data [1] about the countries of the world. This single-table database has a table named `countries` that has named columns, named `name`, `region`, `area`, `population` and `gdp`. The region indicates the region of the world in which a country is located (Europe, Africa, South Asia etc.). The area and population are self explanatory. The GDP is a measure of economic activity in the country; for our purposes its just a number and we need not dwell on how it's defined.

SQL incorporates a simple pattern-recognising capabiity. Whereas a SELECT query with `WHERE region = 'Africa'` will identify the countries in Africa, a query with `WHERE region LIKE 'A%'` will identify those whoe region name begins with the letter 'A' and `WHERE region LIKE '%a'` will identify those that end with the letter 'a'. (Here the % symbol stands for "zero or more characters"

when comparing strings. There is also the '_' (underscore) that stands for a single character.)

Compose complete SQL queries for each of the following.

1. List the names of all the countries in Europe in decreasing order of population.
2. List all various regions alpabetically. Each region should appear once in result.
3. List the names of all the countries, alpahetically by region and within each region by country names i.e. all the African countries first in order from Angola to Zimbabwe, then the Americas and so on.
4. List the names of all countries whose GDP is listed as NULL.
5. List the names of all countries whose region contains the word 'Asia'.
6. List the names of all countries whose name contains the 'South' or 'North'.

[1] The data is somewhat out-of-date having been compiled a number of years ago, but since we are only using it as target practice it will suffice.

7. List the names of all countries whose names ends with 'stan'.
8. List the names of all countries whose names contains one or more of the letters 'x', 'y', 'z'.
9. List the names of all countries whose per capita GDP is at least 25000. The per capita GDP is obtained by dividing the GDP by the population. Ignore countries wiht NULL gdp.

Data Manipulation Queries

This section involves experimenting with SQLite's data manipulation capabilities: `INSERT`, `UPDATE` and `DELETE`. As these will modify the underlying database, we need to be careful. Before experimenting with `students.sqlite`, make a copy named `copy_students.sqlite` and use the copy not the original. If all else fails, remember you can always start over by downloading a fresh copy of `students.sqlite` from the Canvas page and starting again.

10. Modify Aoife's points to add 50 points.
11. Modify Fionn's date of birth and hometown to be first of the millennium and Tahiti, respectively.
12. Add in a new row for a student named Grianne Goggin. Invent suitable values for the various fields.
13. Add a new row for a student named Hugh Hegarty. Populate the row initally with only the id number, first and last names.
14. Complete the Hugh Hegarty row by providing values for the remaining columns.
15. Add in four new rows/students in a single command. Invent appropriate names, homwtwons etc. but give then all points values below 300.
16. Delete all the rows with less than 300 points.
17. Trying to insert a new row with an id number that is the same as one of the existing rows.
18. Try to insert a new row that is identical to an existing row except with a different id number.
19. Experiment with SQLite's save and restore capabilities: (a) Execute the `.save main backupdb.sqlite` to store a snapshot of the database in file `.backupdb.sqlite`; (b) Make a change to the database e.g. seleting Aoife's row; (c) Execute the `.restore main backupdb.sqlite` to restore the database to its original state from the snapshot; (d) Verify that Aoife's row has been re-instated.