Chapter 4 - Update Anomalies and Normalization

This homework requires to have the "world" example Mysql database installed.
To get and install the sample database: Setting Up the world Database

SQL Queries

Note: All SQLs are developed with MYSQL, compatibilities with other RDBMS is not guaranteed.

Implement SQL queries and get the result sets of the following requirements.

A query that displays the city and country names, and populations from cities that have at least 200,000 people and less than 250,000 people.

```
-- Result set row count: 353
SELECT
    city.name, country.name AS countryname, city.Population
FROM city
JOIN country ON city.countrycode = country.code 
```

```
WHERE city.population >= 200000
  AND city.population < 250000;
```

A query that shows the countries where 8 or more languages are spoken, and how many languages are spoken in that country.

```
-- Result set row count: 30
SELECT
    c.Code, c.Name, COUNT(cl.`Language`)
FROM country c
JOIN countrylanguage cl ON c.Code = cl.CountryCode
GROUP BY c.Code
HAVING COUNT(cl.`Language`) >= 8;
```

A query that displays the population, names and countries (name and ID) of the largest city in each country

```
-- Result set row count: 232
SSELECT
    cn.Code, cn.Name,
    c.Name AS cityname,
    c.ID AS cityid, MAX(c.Population) as population
FROM country cn
JOIN city c ON cn.Code = c.CountryCode
GROUP BY cn.code;
```

A query that displays the names of cities and their countries where the capital city in the largest of all cities listed for that country

```
-- Result set row count: 184
SELECT
    c.ID, c.Name,
    cn.Name AS countryname, c.Population
FROM city c
JOIN country cn ON (cn.Code = c.CountryCode AND cn.Capital = c.ID)
JOIN (
    SELECT c1.ID AS cityid, MAX(c1.Population)
    FROM country cn1
    JOIN city c1 ON cn1.Code = c1.CountryCode
    GROUP BY cn1.code) largestcities ON largestcities.cityid = c.ID;
```

Programming

Imagine that you had two datafile cities.csv and countries.csv (text files, where each row is a row in the database, and each column is separated by commas) that represent the database world that we've been working with so far. An example piece of cities.csv looks like:

```
1,Kabul,AFG,Kabol,1780000
```

```
2,Qanadahar,AFG,Qandahar,237500
...
```

Without using any SQL, write a program (in any language) that answers #4 above. It will involve using multiple loops and variables (likely arrays/associative arrays) as well.

Python Implementation

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

"""[Homework 5]
This homework will require you to use the World Database.

This python module/program simulates an RDB SQL query which is an equivalent to
the 4th query in this homework.

[The 4th query]
Write a query that displays the names of cities and their countries where
the capital city in the largest of all cities listed for that country.

SELECT c.ID, c.Name, cn.Name AS countryname, c.Population
FROM city c
JOIN country cn ON (cn.Code = c.CountryCode AND cn.Capital = c.ID)
JOIN (
    SELECT countrycode, ID AS cityid, MAX(Population)
    FROM city
    GROUP BY countrycode) largestcities ON largestcities.cityid = c.ID;

[Python simulation]
Imagine that you had two data file cities.csv and countries.csv that represent
the database world that we?ve been working with so far.
An example piece of cities.csv looks like:
1,Kabul,AFG,Kabol,1780000
2,Qanadahar,AFG,Qandahar,237500
...

Without using any SQL, write a program (in any language) that answers #4 above.
It will involve using multiple loops and variables (likely arrays/associative
arrays) as well.

[Note]
Both cities.csv and countries.csv must be a full dump from their database
tables.

This program results the same number of cities as Q4 does: 184
"""

import csv
import os.path
import io
```

```
import itertools
import traceback


_city_csv = "cities.csv"
_country_csv = "countries.csv"
_csv_dialect = 'excel'


def _convertToNumber(d):
    """Convert dictionary string values into integer if the values are formed
    with digits.
    """
    return dict((k, v if not v.isdigit() else int(v)) for (k, v) in d.items())


def _read_csv(csv_file):
    """Read the given csv file and store the result sets into a list of
    dictionary."""
    if csv_file == None or csv_file.strip() == "":
        raise ValueError("Missing file path")
    if not os.path.exists(csv_file) and not os.path.isfile(csv_file):
        raise IOError("File does not exist: %s" % csv_file)

    result_set = None
    with open(csv_file, "rb") as csvfile:
        reader = csv.DictReader(csvfile, dialect=_csv_dialect)
        result_set = [_convertToNumber(r) for r in reader]
    # test
    # print re
    return result_set


def _groupby(result_set, groupby_key, sort_key=None, sort_desc=False):
    """Slice the given list of dict into sublist groups by the given key.
    If the sub groups should be sorted, specify sort_key.
    """
    grouped = []
    if not result_set or len(result_set) == 0 or not groupby_key or len(groupby_key.strip()) == 0:
        return grouped
    for column, group in itertools.groupby(result_set, lambda d: d[groupby_key]):
        g = list(group)
        if sort_key and len(sort_key.strip()) != 0:
            g = sorted(g, key=lambda k: k[sort_key], reverse=sort_desc)
        grouped.append(g)
    return grouped


def _group_max(result_set, groupby_key, sort_key):
    """Get a list of dict records which have the largest value of sort_key in
     groups.
```

```
     This method does the SQL part:

    SELECT *, max(sort_key)
    FROM table
    GROUP BY groupby_key
    """
    if not result_set or len(result_set) == 0:
        return []
    return [x[0] for x in _groupby(result_set, groupby_key, sort_key, True)]


def _join(src, to_join, on_src_key, on_to_join_key):
    """Get a sublist of src which is in an intersection of 2 lists of dictionary
    on one key.
    This method simulates a SQL JOIN, such as:

    SELECT t.*
    FROM table t
    JOIN table1 t1 ON t.key = t1.key
    """
    on_values = set(d[on_to_join_key] for d in to_join)
    return [d for d in src if d[on_src_key] in on_values]


def _select_capitals_with_max_population(cities, countries):
    """Get a list of cities which both are capital and have the most population
    in the country"""
    groupby_key = "CountryCode"
    sort_key = "Population"
    capital_key = "Capital"
    city_id = "ID"
    # cities with max(population)
    grouped_cities_max_population = _group_max(cities, groupby_key, sort_key)
    # captital cities
    capital_cities = _join(cities, countries, city_id, capital_key)
    return _join(capital_cities, grouped_cities_max_population, city_id, city_id)


# Main entrance
if __name__ == '__main__':
    try:
        cities = _read_csv("cities.csv")
        countries = _read_csv("countries.csv")
        if not cities or len(cities) == 0:
            raise RuntimeError("No data read from city file")
        if not countries or len(countries) == 0:
            raise RuntimeError("No data read from country file")
        capitals_max_population =\
            _select_capitals_with_max_population(cities, countries)
        for row in capitals_max_population:
            print row
```

```
        print "Cities: %d\n" % len(capitals_max_population)
    except Exception as e:
        print "Runtime error happened: %s" % e
        traceback.print_exc()
        exit(1)
    exit(0)
```

E4.5 from the book (Page 121)

AIRPORT KLX TABLE

Date	AirlineID	Airline Name	TerminalID	NumberOfGates	NumberOfDepartingFlights

Insertion anomaly

Cannot insert a new terminal, say (D:15), into the table without an airline infomation.

Deletion anomaly

Cannot delete airline SWA from the table without deleting terminal C's information.

Modification anomaly

Changing terminal B's NumberOfGates will make change to 3 records.

Full key functional dependancies, partial key functional dependancies, transitive functional dependancies

```
Full key functional dependancies: Date, AirlineID --> NumberOfDepartingFlights
Partial key functional dependancies: AirlineID --> AirlineName, TerminalID, NumberOfGates
Transitive functional dependancies: TerminalID --> NumberOfGates
```

2NF Result

TRAFFIC TABLE

Date	AirlineID	NumberOfDepartingFlights

AIRLINE TABLE

AirlineID	Airline Name	TerminalID	NumberOfGates

3NF Result

TRAFFIC TABLE

Date	AirlineID	NumberOfDepartingFlights

AIRLINE TABLE

TerminalID	AirlineID	Airline Name

TERMINAL TABLE

TerminalID	NumberOfGates

TerminalID	NumberOfGates

Using 3NF result, describe how anomalies are eliminated

```
Insertion: No anomaly because insertion a new terminal won't require full information of traffic and airline information.
Modification: No anomaly because update upon terminal B's NumberOfGates affect only 1 record in TERMINAL table.
Deletion: No anomaly because deleting terminal C in TERMINAL table will not affect AIRLINE table.
```

Tools and Reference

Reference

- SQL JOINS: LEFT JOIN vs. LEFT OUTER JOIN

Tools

- HediSQL, A lightweight and handy tool for SQL development: HeidiSQL - MySQL, MSSQL and PostgreSQL made easy

---- Jason, 10/12/2016