DATABASES MINI-PROJECT SPECIFICATIONS
CS2855

Dr. Iddo Tzameret
Department of Computer Science
Royal Holloway, University of London
DUE DATE: **8 January 2018, 15:00**

LEARNING OUTCOMES

This mini-project covers some of the notions covered in class and lab tutorials. The assessed learning outcomes are a correct usage and understanding of SQL via the JDBC library.

SUBMISSION INSTRUCTIONS

You must build your java project with **a single class** file according to the specification described in what follows. **Please read very carefuly the requirements, they are not extremely difficult, but following the details is very important.**

Please submit the program by **8 January 2018, 15:00 (UK time)**. Submissions are done through the cs submission server, as usual. Here are the submission instructions:

> You will submit your work electronically by means of the submission script *submitCoursework* which can be found on teaching. You may resubmit your script any number of times, though only the last submission will be kept. The submission will occur on teaching and the protocol is:
>
> *submitCoursework <your directory>*
>
> For example, assuming the directory with your java class file is *dirname* then you would submit by typing in the following command from teaching:
> \> *submitCoursework dirname*
> and then follow the instructions accordingly. The files you submit cannot be overwritten by anyone else, and they cannot be read by any other student. You can, however, overwrite your submission as often as you like, by resubmitting, though only the last version submitted will be kept. Submission after the deadline will be accepted but it will automatically be recorded as being late and is subject to College Regulations on late submissions. Please note that all your submissions will be graded anonymously.

MARKING CRITERIA

This coursework is assessed and mandatory and is worth 20% of your total final grade for this course. Some marks may be given if the approach was correct even if the final answer is not complete.

This is a programming assignment, thus **if your program does not compile you have the risk of getting 0 marks. The program will be tested on linux.cim server (formerly known as the Teaching server) (and on the Eclipse IDE). So make sure the program compiles and runs correctly on this server.**

The program will be tested **against test files**. The test files will have the **same format** as discussed below in the Specifications section. So make sure your program is oblivious to the *content* of the file (but of course, no to the format of the file), and will result in a correct answer to any possible test file conforming to the specified format.The fact that your program gives the correct results on the input files supplied on Moodle (these correct results are shown below as "example outputs"), does not necessarily imply that it will give the correct results on other files; so make sure that the logic of the queries is correct.

Correct functionality of the program as tested via a run of the program against the test files as input will have a clear correspondence with the final mark of the mini-project. But the way the database was implemented within the project will also have a weight on the marking (namely, the functionality and output of the program should be dealt with by querying a database; obviously, everything here can be done even without using postgreSQL, but the latter choice would be regarded as an unreasonable implementation).

PROGRAM SPECIFICATION

You are to build a program that uses a database for storing, manipulating and querying information about worldwide most popular URLs.

- Perhaps the most important part in successfully completing the project is **to read very carefully the specificaitions** (possibly as you progress with the program), since this will save you precious time figuring out some technical details by yourself.
- You should use the Java language, and the program should run on the linux.cim server, connecting to the *postgreSQL* server as was done in the lab sessions (it will be tested on linux.cim, so make sure it compiles on linux.cim using Eclipse!).
- You should not use any other auxiliary file in your program. Only the files *topURLs* and *mapping* should be used.
- The program should be oblivious to the content of the two files (but not to their format).
- **Files**: the java class file, the *topURLs* file and the *mapping* file, should be **in the same directory!** The program should read the file using **a relative path**, otherwise we will not be able to check the project! You can download these (example) files from Moodle.
- The program should have a simple **text based user interface**. There should be no graphical user interface; in fact, there should not be a user interface at all, namely, *avoid* defining menus and choices.
- **JDBC**: See instructions on Moodle (Lab 7) on how to use the JDBC library, and how to connect to a postgreSQL server from a Java program. The model solution for Lab 7 is a good template to start the project from (it contains methods to drop tables [you can similarly drop views], sending SQL queries, and reading files into tables).
- **Better use of JDBC**:
  - I have uploaded other example files to Moodle (Lab 7) showing how to use JDBC better than what is shown in the model solution for lab 7. Specifically, this includes how to avoid memory allocation leaks (using anonymous classes).
  - **Prepared statements**: Another important aspect that is absent from the example files are ***prepared statements*** for SQL. Such statements protect against *SQL injections*, and are thus advisable from the security perspective. You might want to use prepared statements in your code (though I'm not going to penalise students who don't). Prepared statements are quite easy to use; see for example: https://docs.oracle.com/javase/tutorial/jdbc/basics/prepared.html

Required functionality of the program:

Initialising the connection to the postgres server

1. **Important**: The program should take as **an argument** the username and password of the user, **and connect to the correct database based on the username passed to the program as an argument.** So please **DON'T** just write your own username, own database name and password inside the program code. The reason is that we need to test it on *our own* database.
2. The connection to the postgres server is then done in the same way shown in lab (see Lab 7 on Moodle, using appropriatly the arguments passed to the program).

Reading/writing tables in the databases

1. The program should check if the tables (and possibly views) it creates already exist in the database test, and **only if they do,** drops them before their creation.

For your convenience, here is an example showing how to check for the existence of tables in the databse:

```
// connection is of type Connection (in JDBC)
DatabaseMetaData dbm = connection.getMetaData();

// check if table is there
ResultSet tables = dbm.getTables(null, null, “table name”, null);
if (tables.next())
       // Table exists
}
else {
       // Table does not exist
}
```

As for checking for the existence of **views** in the databse, it is very similar and I leave it for you to find out.

Information about URLs and top level domains

For the sake of this mini-project a URL is an alphanumeric string (including the period "." symbol), like google.com and google.co.uk. The string is divided into substrings via dots. The first substring in google.com is "google" and the second substring is "com". Accordingly, the first substring of google.co.uk is "google", the second is "co" and the third is "uk".

The ***top level domain***, in our context, will always be the given URL excluding the first substring (and excluding the dots). E.g., the top level domain in google.com is "com". And the top level domain in "google.co.uk" is "co.uk" (and *not* "uk").

You may assume that every URL contains **at most two dots** (and thus at most three substrings). E.g., *google.co.uk* is legit, but *this.has.more.dots* is not legit.

Information about mapping *parts* of top level domains to their descriptions

The file mapping contains, in each row, the rightmost substring of the top level domain and its description, **separted by a tab**. E.g.,

```
com        commercial organization
uk         United Kingdom (United Kingdom of Great Britain and Northern Ireland)
```

Important notice: Note that although uk is mapped to its description in the mapping file above, the top-level domain for google.co.uk is (in our context) co.uk (and *not* uk).

Reading the topURLs file

topURLs file format

The file topURLs file is a file that has the following format:

- Each line of topURLs file consists of an integer and a website address **separated by a tab**. A line
  ```
  n      <address>
  ```
  means that <address> is the worlds's n-th most popular website.
- The number of lines in the file will not exceed 10000.
- Note that the file **might not be ordered**! Namely, more popular websites might appear after less popular ones.

2. The program now should read the file topURLs and insert the information it reads to the database, in an appropriate way that would suite the requirements below (namely, it should insert it to tables in the way best suitable to a database in which the queries below can be completed with relative ease).

Popularity Queries

The program should now output some statistical information about the data it reads from the file. More precisely, it should do the following (in the **same** order they are listed):

3. List the 10 most popular URLs in descending order (decending with respect to popularity).

   Example of output:

   ```
   ################# 1st Query ################
   1      google        com
   2      facebook      com
   …

   10     live          org
   ```

4. List the 10 distinct most popular top level domains in descending order (with respect to popularity).

Example of output (you might wish to replace "null" with blank):

```
################# 2nd Query ################
com    null
org    null
co     in
…
co     uk
net    null
```

5. List the 10 distinct most popular **descriptions** of the rightmost parts of the toplevel domains in descending order (with respect to popularity).

   Example:

```
################## 3nd Query ###############
  Commercial organizations
  Non-profit organizations
  India (Republic of)
  China (People Republic of)
  Japan
  Russia (Russian Federation)
  Germany (Federal Republic of)
  United Kingdom (United Kingdom of Great Britain and Northern Ireland)
  Network
  France (French Republic)
```

6. List the top 10 distinct most popular *left*-most substrings *appearing in more than one line* in the topURLs file. E.g., the occurrence of both "google.com" and "google.co.uk" in the file, means that the leftmost substring "google" appears in more than one line.

   Example:

```
################## 4th Query ###############
       google
       amazon
       yahoo
       …
       mail
```

Notes:

- When you are required ***to list*** a collection of URLs, it is meant to output them to the standard output, where each URL is written ***in a new line***.
- A URL may be printed as above, i.e., without *dots*, and in which parts are sperated by tabs or spaces, an empty string can be replaced by "null" if you find it more convenient.
- After the program lists each of the clauses above, it should output section separators like "`########### ith Query ########`" to make the output easy to read.
- If you are requested to list 10 domains, websites, etc., and there are less than 10 such elements, the program should simply print as many as there are.
- Make sure that basic integrity constrainsts like **primary keys** and **foreign keys** are specified in the relation schemas you define (so that, e.g., users cannot delete a record containing information referred to (implicitly) by another table).

Happy Coding