EECS3421, A Section, Fall 2019
Assignment 2
Interactive & Embedded SQL Queries

Due date: *Fri, Nov 8, 11:59pm*

Instructions

1. Read this assignment thoroughly before you proceed. Failure to follow instructions can affect your grade.
2. Download the database schema **a2.ddl** from the course website.
3. Download the file **a2.sql** from the course website.
4. Download the java skeleton file **Assignment2.java** from the course website.
5. Submit your work **electronically** using UNIX *submit*. Your submission must include the following files:
   - **a) a2.sql** — your queries for the interactive SQL part of the assignment (can include any view creation statement). If you define any **views** for a question, you must drop them after you have populated the answer table for that question.
   - **b) Assignment2.java** — your java code for the embedded SQL part of the assignment. Be careful to submit the .java file (**not** the .class file.). To get you started, we provide a skeleton of this file that you must download from the assignment webpage. Also, make sure that your java file **does not** include any `main()` function.
   - **c) team.txt** — a file that includes information about the team members (first name, last name, student ID, login, yorku email).

How to Submit

When you have completed the assignment, move or copy your files in a directory (e.g., assignment2), and use the following command to electronically submit your files:

```
% submit 3421A a2 a2.sql Assignment2.java team.txt
```

You can also submit the files individually after you complete each part of the assignment– simply execute the *submit* command and give the filename that you wish to submit. Make sure you name your files **exactly** as stated (including lower/upper case letters). Failure to do so will result in a mark of 0 being assigned. You may check the status of your submission using the command:

```
% submit -l 3421A a2
```

Interactive SQL Queries [50 marks]

In this section, you must edit the file **a2.sql** and add SQL statements that can **be run in *psql* on Prism machines.** Your SQL statements can create views and queries that will populate the result tables *query1, query2, …, query10* with tuples that satisfy the questions below. In order to ensure that everything is run in the correct order (by the markers or the automarker), you should add all your SQL statements in the file ***a2.sql*** that we have provided. This file can be read and executed using the *psql* command:

\i <FILENAME>

You can assume that the **a2.ddl** file has been read and executed in psql, before your **a2.sql** file is executed.

Follow these rules:

- The output of each query must be stored in a result table. We provide the definitions of these tables in the **a2.ddl** file (*query1, query2, …, query10*).
- For each of the queries below, your final statement should populate the respective answer table (*queryX*) with the correct tuples. It should look something like:

  "INSERT INTO queryX (SELECT … <complete your SQL query here> …)"

  where X is the correct index [1, …,10].
- In order to answer each of the questions, you are encouraged to create virtual **views** that can keep intermediate results and can be used to build your final INSERT INTO *queryX* statement. Do not create actual tables. Remember that you have to drop the views you have created, after each INSERT INTO *queryX* statement (i.e., after you have populated the result table).
- Your tables **must** match the output tables specified for each query. The attribute names **must** be **identical** to those specified in italics, and they must be in the specified order. Also, make sure to sort the results according to the attributes and ordering we specify in each question.
- We are not providing a sample database to test your answers, but you are encouraged to create one. We will test the validity of your queries against our own test database.
- All of your statements must run on PostgreSQL on the Prism machines, so be sure to populate your tables with test data and run all your statements on Prism prior to submission.

NOTE: Failure to do follow the instructions may cause your queries to fail when (automatically) tested, and you will lose marks.

Express the following queries in SQL.

1. [5 **marks**] Find player(s) that have been a champion in a tournament held in their country. Report the name of the player, country and tournament.
   Output Table: **query1**

Attributes:	*pname*	(player name)	[VARCHAR]
	cname	(country name)	[VARCHAR]
	tname	(tournament name)	[VARCHAR]
Order by:	*pname*	ASC	

2. [5 **marks**] Find the tournament that has the most seats in all courts (sum of capacity of its courts). Report the name of the tournament and the total capacity.
   Output Table: **query2**

Attributes:	*tname*	(tournament name)	[INTEGER]
	totalCapacity	(total capacity of its courts)	[VARCHAR]
order by:	*tname*	ASC	

3. [5 **marks**] For each player, among all the players he/she has played against, find the one with the highest globalRank. Report the id and name of the player(p1) and the opponent(p2).
   Output Table: **query3**

Attributes:	*p1id*	(player id)	[INTEGER]
	p1name	(player name)	[VARCHAR]
	p2id	(opponent player id)	[INTEGER]
	p2name	(opponent player name)	[VARCHAR]
Order by:	*p1name*	ASC	

4. [5 **marks**] Find the players that have been a champion in every tournament. Report the id and name of the player.
   Output Table: **query4**

Attributes:	*pid*	(player id)	[INTEGER]
	pname	(player name)	[VARCHAR]
order by:	*pname*	ASC	

5. [5 **marks**] Find the top-10 players with the highest average number of wins over the 4-year period of 2011-2014 (inclusive). (For example, player A won 25 games in 2017 and 24 in 2018, the average number of wins over these 2 years for A is 24.5) Notice: solve this one **based on the information in record table, not event table.**
   Output Table: **query5**

Attributes:	*pid*	(player id)	[INTEGER]
	pname	(player name)	[VARCHAR]
	avg*wins*	(player's average # of wins)	[REAL]
Order by:	avg*wins*	DESC	

6. [5 **marks**] Find the players for which their yearly number of wins is constantly increasing over the 4-year period of 2011-2014 (inclusive). Constantly increasing means from year to year there is a positive change (increase) in their number of wins (e.g., a player where 2011: 24, 2012: 25, 2013: 28, 2014: 30).
   Output Table: **query6**

Attributes:	*pid*	(player id)	[INTEGER]
	pname	(player name)	[VARCHAR]
Order by:	*pname*	ASC	

7. [5 **marks**] Find players that have been champions at least twice in a single year. Report the name of each player and the year(s) that they achieved this (If a player has achieved this twice, the result should include 2 tuples related to this player).
   Output Table: **query7**

Attributes:	*pname*	(player name)	[VARCHAR]
	year	(year)	[INTEGER]
Order by:	*pname*	DESC,	
	year	DESC	

8. [5 **marks**] Find all pairs of players that have played against each other and come from the same country. Report the names of the players and the name of the country. For example, <Federer, Wawrinka, Switzerland> should be in the result tuples because both players had played against each other in a match (event), and they both come from Switzerland. (***Notice***: In this example, you should insert **both** <Federer, Wawrinka, Switzerland> and < Wawrinka, Federer, Switzerland> into the table)
   Output Table: **query8**

Attributes:	*p1name*	(player name)	[VARCHAR]
	p2name	(opponent country name)	[VARCHAR]
	cname	(country name)	[VARCHAR]
Order by:	*cname*	ASC,	
	p1name	DESC	

9. [5 **marks**] Find the country (a citizen of which) has won the most tournaments (look for players coming from that country, i.e., player.cid == country.cid). Report the country name and total number of champions its player(s) won.
   Output Table: **query9**

Attributes:	*cname*	(country name)	[VARCHAR]
	champions	(number of champions)	[INTEGER]
Ordey by:	*cname*	DESC	

10. [5 **marks**] Find the player(s) that had more wins than losses in 2014 in all courts and their participation time was more than 200 minutes on average in all games (not only in 2014).
    Output Table: **query10**

Attributes:	*pname*	(player name)	[VARCHAR]
Ordey by:	*pname*	DESC	

Embedded SQL Queries [50 marks – 5 for each method]

For this part of the assignment, you will create the class **Assignment2.java** which will allow you to process queries using JDBC. We will use the standard tables provided in the **a2.ddl** for this assignment. If you feel you need an intermediate **view** to execute a query in a method, you must create it inside that method. You must also drop it before exiting that method.

Rules:

- Standard input and output must **not** be used. This will halt the "automarker" and you will probably end up with a zero.
- The *database*, *username*, and *password* must be passed as parameters, never "hard-coded".
- Be sure to close all unused statements and result sets.
- All return values will be *String*, *boolean* or *int* values.
- A successful action (Update, Delete) is when:
  - It doesn't throw an SQL exception, and
  - The number of rows to be updated or deleted is correct.

Class name	**Description**
Assignment2.java	Allows several interactions with a postgreSQL database.

Instance Variables (you may want to add more)

Type	**Description**
Connection	The database connection for this session.

Methods (you may want to add helper methods.)

Constructor	**Description**
Assignment2()	Identifies the postgreSQL driver using Class.forName method.

Method	**Description**
boolean connectDB(String URL, String username, String password)	Using the String input parameters which are the *URL*, *username*, and *password* respectively, establishes the Connection to be used for this session. Returns **true** if the connection was successful.
boolean disconnectDB()	Closes the connection. Returns **true** if the closure was successful.
boolean insertPlayer(int pid, String pname, int globalRank, int cid)	Inserts a row into the player table. *pid* is the id of the player*, pname* is the name of the player, *globalRank* is the global ranking of the player, cid is the id of the country of the player. You have to check if the player with id *pid* exists. Returns **true** if the insertion was successful, **false** otherwise.
int getChampions(int pid)	Returns the number of champions the player with id *pid* has won.
String getCourtnInfo(int courtid)	Returns a string with the information of a court with id *courtid*. The output is "courtid:courtname:capacity:tournamentname". Returns an empty string "" if the court does not exist.

Method	**Description**
boolean chgRecord(int pid, int year, int wins, int losses)	Changes the value of wins and losses of the player p*id* for the year *year* to the new values provided. Returns **true** if the change was successful, **false** otherwise.
boolean deleteMatchBetween(int p1id, p2id)	Deletes all events between two players. Returns **true** if the deletion was successful, **false** otherwise. You can assume that the events between two players to be deleted exists in the database.
String listPlayerRanking()	Returns a string that describes all player rankings ordered in a descending order from the highest ranking to the lowest one. The list of player rankings should follow the contiguous format described below, and contain the following attributes in the order shown: "p1name:p1rank**\n**p2name:p2rank**\n**... " where: • *piname* is name of the i-th player. • *pirank* is the ranking the i-th player. • **(should be ordered from the highest ranking to thr lowest one)** Returns an empty string "" if the country does not exist.
int FindTriCircle()	A tri circle is defined as: If player A has (at least once) won player B, player B has (at least once) won player C and player C has (at least once) won player A. This is called a tri circle. Return the number of tri circles in the database. If the 'ABC' example above was the only instance of a tri circle in the database, then your method should have returned the number 1.
boolean updateDB()	Create a table containing all the players which have won at least one tournament champion title. The name of the table should be *championPlayers* and the attributes should be: • *pid* INTEGER (player id) • *pname* VARCHAR (player name) • *nchampions* INTEGER (number of champions) Returns **true** if the database was successfully updated, **false** otherwise. Store the results in **ASC** order according to the player id (pid).