CSCC43 Summer 2015
Assignment 2
Interactive & Embedded SQL Queries

Due date: 3rd July at 11:59 pm

Instructions

Read this assignment thoroughly before you proceed. Failure to follow instructions can affect your grade.
Download the database schema **a2.ddl** from the assignment webpage.
Download the file **a2.sql** from the assignment webpage.
Download the java skeleton file **Assignment2.java** from the assignment webpage.
Submit your work **electronically** using *MarkUs*. Your submission must include the following files:

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.

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.

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 the CDF machine.** 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.

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 CDF machines, so be sure to populate your tables with test data and run all your statements on CDF 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.

[5 **marks**] Find player(s) that have been a champion in a tournament that was not held in their team city. Report the name of the player, city and tournament.

Output Table: **query1**			
Attributes:	*pname*	(player name)	[VARCHAR]
	cname	(city name)	[VARCHAR]
	tname	(tournament name)	[VARCHAR]
Order by:	*pname*	ASC	

[5 **marks**] Find the team that have never won any tournaments held in their home city.

Output Table: **query2**			
Attributes:	*tname*	(team name)	[VARCHAR]
order by:	*tname*	ASC	

[5 **marks**] Find the id and name of the players who have played against the team of the player with the highest globalRank (globalRank=1).

Output Table: **query3**			
Attributes:	*pid*	(player id)	[INTEGER]
	pname	(player name)	[VARCHAR]
Order by:	*pname*	ASC	

[5 **marks**] Find the teams that have been a champion in every tournament. Report the id, name and the city of the team.

Output Table: **query4**			
Attributes:	*tid*	(team id)	[INTEGER]
	tname	(team name)	[VARCHAR]
	city	(city name)	[VARCHAR]
order by:	*tname*	ASC	

[5 **marks**] Find the top-10 teams with the highest winning average over the 5-year period of 2011-2015(inclusive). (For example, team A won 25 games in 2011 and 24 in 2012, the winning average over these 2 years for A is 24.5)

Output Table: **query5**			
Attributes:	*tid*	(team id)	[INTEGER]
	tname	(team name)	[VARCHAR]
	avg*wins*	(team's average HDI)	[REAL]
Order by:	avg*wins*	DESC	

[5 **marks**] Find the teams for which their winning times is not improving over the 4-year period of 2012-2015(inclusive). Not improving means that from year to year there is not always increasing change in team's wins.

Output Table: **query6**

Attributes:	*tid*	(team id)	[INTEGER]
	tname	(team name)	[VARCHAR]
	city	(city name)	[VARCHAR]
Order by:	*tname*	ASC	

[5 **marks**] Find defenseman players that have played against champion teams at least once in a single year. Report the name of each player, the year(s) and the teams that they have played against.

Output Table: **query7**

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

[5 **marks**] Find all pairs of players that have played against each other at least twice. Report the names of the players and the number of times they have played against each other. For example, <A1, A2, 4> should be in the result because the two players had played four matches (event). (***Notice***: In this example, you should insert **both** <A1, A2, 4> and < A2, A1, 4> into the table)

Output Table: **query8**

Attributes:	*p1name*	(player name)	[VARCHAR]
	p2name	(opponent player name)	[VARCHAR]
	num	(number of matches)	[INTEGER]
Order by:	*p1name*	DESC	

[5 **marks**] Find the cities that team(s) who belongs to it (team.cid == city.cid) have won the minimum number of tournaments. Report the city name and total number of tournaments these team(s) won.

Output Table: **query9**

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

[5 **marks**] Find the team(s) that had more wins than losses in 2015 in all rinks and their participation time was more than 200 minutes on average in all games (not only in 2015).

Output Table: **query10**

Attributes:	*tname*	(team name)	[VARCHAR]
Ordey by:	*tname* 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". We will use your connectDB() method defined below to connect to the database with our own credentials.

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, establish 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 insertTeam(int eid, String ename, int cid)	Inserts a row into the team table. *eid* is the id of the team*, ename* is the name of the team, *cid* is the id of the city of the team. You have to check if the team with id *eid* exists.

	Returns **true** if the insertion was successful, **false** otherwise.
Int getChampions(int eid)	Returns the number of times the team with id *eid* has been a champion.
String getRinkInfo(int rid)	Returns a string with the information of a rink with id *rid*. The output is “rinkid:rinkname:capacity:tournamentname” . Returns an empty string “” if the court does not exist.
Boolean chgRecord(int pid, int rank)	Changes the global rank of the player p*id*. Returns **true** if the change was successful, **false** otherwise.
Boolean deleteMatchBetween(int e1id, e2id)	Deletes all the events between two teams between years of 2011 to 2015. Returns **true** if the deletion was successful, **false** otherwise. You can assume that the events between two teams to be deleted exists in the database.
String listPlayerRanking()	Returns a string that describe the player ranking as following format: “p1name:p1globalRank\np2name:p2globalRank... ” where: *piname* is name of the i-th player. *piglobalRank* is the ranking the i-th player.
Int FindTriCircle()	A tri circle is defined as: If team A has (at least once) won team B, player B has (at least once) won team C and team C has (at least once) won team A. This is called a tri circle. Return the number of tri circles in the database. In the ‘ABC’ example above (If there are no others), return 1.
boolean updateDB()	Create a table containing all the players whose teams have won all the tournaments in their home city. The name of the table should be *allTimeHomeWinners* and the attributes should be: *pid* INTEGER (player id)

	tid INTEGER (team id) *pname* VARCHAR (player name) *city* VARCHAR (city name s) Returns **true** if the database was successfully updated, **false** otherwise. Store the results in **ASC** order according to the player id (pid).