CSC343 Winter 2014
Assignment 2
Interactive & Embedded SQL Queries

Due date Updated: *Friday 7 Mar at 10:00pm*

Instructions

1. Read this assignment thoroughly before you proceed. Failure to follow instructions can affect your grade.
2. The database schema you are using is defined in **A2.ddl** which must be downloaded from the assignment webpage.
3. You must hand in your work **electronically** using the UNIX *submit* command. You must submit the following 5 files:
   - **a) team.txt** information about your team (whether it is a team of one or two students).
   - **b) a2drop** statements for dropping all tables and views of your assignment.
   - **c) a2tables** statements for creating result tables.
   - d) **a2sql** your queries for the interactive SQL part of the assignment (can include any view creation statement).
   - **e) 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.

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 within that directory:

% submit -c csc343h -a A2 team.txt a2drop a2tables a2sql Assignment2.java

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. You may submit your solutions as many times as you wish prior to the submission deadline (you might need to use the ***–f*** flag of the submit command). 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.

Once you have submitted, be sure to check that you have submitted the correct version of each file; new or missing files will not be accepted after the due date. You may check the status of your submission using the command

% submit -l -N A2 csc343h

where -l is a hyphen followed by the letter 'ell'.

Interactive SQL Queries [45 marks]

In this section, you must create views and queries **to be run in *psql* on the CDF machine.** In order to ensure that everything is run in the correct order (by the markers), you must create three files containing your statements (***a2drop***, ***a2tables***, ***a2sql***), which can be read into *psql* and executed using the *psql* command:

\i <FILENAME>

The files must be as follows:

1. **a2drop**: statements to drop any and all views and tables that you create.
2. **a2tables:** DDL statements to create the tables which will hold the query results (Query1 through Query7), as well as any intermediate views you may require for this assignment.
3. **a2sql**: all queries that populate the QueryX table (e.g., Query1, Query2, etc.) with results that satisfy the questions below.

Express the following queries in SQL. Follow these rules:

- The output of each query must be stored in a new table. You must create the table definition for the tables that are used to store the results of your queries (e.g., Query1, Query2, etc.). These definitions should be saved in **a2tables**.
- You are encouraged to create **views** in order to have intermediate relations in hand that could help build your final SQL statement, but not actual tables.
- The final statement to insert your query results should look something like:
  "INSERT INTO Query1 (SELECT … <complete your SQL query here> …)"
- 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.
- 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 machine, so be sure to populate your tables with test data and run all your statements on CDF prior to submission.

NOTE: Failure to do this may cause your query to fail when (automatically) tested, and you will lose marks.

1. [**5 marks**] Find the player(s) that has the best ratio of value-to-goals and print his first name, last name and the name of the national team into which he belongs. Smallest ratio is the best since it denotes that a player has scored many goals and his value is not big enough making him the ideal player for purchasing him during the summer transfers period.
   Output Table: **Query1**
   Attributes: *fname* (first name) [VARCHAR(20)]
   lname (last name) [VARCHAR(20)]
   country (country name) [VARCHAR(20)]

2. [**5 marks**] Find the top-3 names of the football clubs that have the most players participating into World Cup 2014. *Note*: Explore the *LIMIT* clause of PostgreSQL to answer this query.
   Output Table: **Query2**
   Attributes: *name* (football club name) [VARCHAR(20)]
   num (number of players) [INTEGER]
   order by: *num* DESC, *name* ASC

3. [**5 marks**] Find out whether the first match of Brazil is also the opening match (i.e., first match) of World Cup 2014 competition.
   Output Table: **Query3**
   Attributes: *isOpeningGame* (boolean attribute) [BOOLEAN]

4. [**5 marks**] Report the MID(s) of the matches for which at least two tickets were bought on the date of the match. Present the results in a descending order.
   Output Table: **Query4**
   Attributes: *mid* (the match id) [INTEGER]

5. [**5 marks**] Find the national team(s) that has played matches in all the stadiums of World Cup 2014.
   Output Table: **Query5**
   Attributes: *name* (the name of the national team) [VARCHAR(20)]

6. [**10 marks**] Find the pid, the first and the last name(s) of the football player(s) that played in all the matches of their national team and their participation time was more than 75 minutes on average.
   Output Table: **Query6**

Attributes:	*pid*	(player id) [INTEGER]
	fname	(first name) [VARCHAR(20)]
	lname	(last name) [VARCHAR(20)]
	minutes	(AVG minutes played) [NUMERIC]
Order by:	*minutes*	DESC

7. [**10 marks**] Retrieve the name(s) and coach(es) of national team(s) that have the lowest budget (i.e., the total sum of values of the players in a national team) and happen to have a player that is the top scorer of World Cup 2014.
   Output Table: **Query7**
   Attributes: *name* (name of the national team) [VARCHAR(20)]
   coach (name of the coach) [VARCHAR(20)]
   budget (total team budget) [INTEGER]

Embedded SQL Queries [55 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 in 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.
- When rows of data are returned as a String, they must be in the following contiguous format:
  - Columns are separated with a colon ":". There is no colon after the last column of a row.
  - Rows are separated with a pound-sign "#". There is no pound-sign after the last row.
  - Leading and trailing spaces are eliminated; e.g., a result set that includes 4 rows of the columns firstname, lastname might look like:

    "fnameA:lnameA#fnamaB:lnameB#fnameC:lnameC#fnameD:lnameD"

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 insertCountry(int cid, String name, String coach)	Inserts a row into the country table. *name* is the name of the country (national team) and *coach* is the coach of the newly inserted national team. You have to check if the country with id *cid* exists. Returns **true** if the insertion was successful, **false** otherwise.

Method	**Description**
int getPlayersCount(int cid)	Returns the number of players registered in table 'player' that play in country with id *cid*. Returns **-1** if an error occurs.
String getPlayerInfo(int pid)	Returns a string with the information of a player with id *pid*. The output is "fname:lname:position:goals". Returns an empty string "" if the player does not exist.
boolean chgStadiumLocation(int sid, String newCity)	Changes the city name of the stadium s*id* to the city name supplied (newCity). Returns **true** if the change was successful, **false** otherwise.
boolean deleteCountry(int cid)	Deletes the national team identified by *cid*. Returns **true** if the deletion was successful, **false** otherwise. You can assume that the national team to be deleted exists in the database.
String listPlayers(String fcName)	Returns a string with all the players that play in a football club with name *fcName* and participate in the World Cup 2014 as well. The list of players should follow the contiguous format described above, and contain the following attributes in the order shown: "fname1:lname1:position1:goals1:cname1#fname2:lname2:position2:goals2:cname2#... " where: • *fname* is the first name of the player. • *lname* is the last name of the player. • *position* is the position into which the player plays in the team. • *goals* is the number of goals that the player has scored in World Cup 2014. • *cname* is the name of the country with which this player plays in World Cup 2014. Returns an empty string "" if the football club does not exist.
boolean updateValues(String cname, int incrV)	Increases the values of all the players playing in the national team with cname by the value of argument *incrV* (i.e. value + incrV). Returns **true** if the update was successful, **false** otherwise.
String query7()	Execute Query 7 described in the Interactive SQL section above. Instead of inserting the results in a table, return them as a String in the same format as is specified for the output table for the query. Be sure to follow the pre-stated String rules involving colons and pound-signs for your return String. Do not use the views that you created in the SQL part. If you need views, create them when you execute function query7().
boolean updateDB()	Create a table containing all the players who have played exactly 90 minutes in at least one of the match(es) they have participated. The name of the table should be *valuablePlayers* and the attributes should be: *pid* INTEGER (player id) *lname* VARCHAR(20) (last name) Returns **true** if the database was successfully updated, **false** otherwise. Do not report duplicates (i.e., each player should be reported only once). Store the results in ASC order according to the player id (pid).