Project Description: Tournament Planner

In this project, you'll be writing a Python module that uses the PostgreSQL database to keep track of players and matches in a game tournament.

The game tournament will use the Swiss system for pairing up players in each round: players are not eliminated, and each player should be paired with another player with the same number of wins, or as close as possible.

This project has two parts: defining the database schema (SQL table definitions), and writing the code that will use it.

Code Templates

The templates for this project are in the tournament subdirectory of your VM's /vagrant directory. You'll find three files there: tournament.sql, tournament.py, and tournament_test.py.

The template file tournament.sql is where you will put the database schema, in the form of SQL create table commands. Give your tables names that make sense to you, and give the columns descriptive names. You'll also need to create the database itself; see below.

The template file tournament.py is where you will put the code of your module. In this file you'll see stubs of several functions. Each function has a docstring that says what it should do.

Finally, the file tournament_test.py contains unit tests that will test the functions you've written in tournament.py. You can run the tests from the command line, using the command python tournament_test.py.

Creating Your Database

Before you can run your code or create your tables, you'll need to use the create database command in psql to create the database. Use the name tournament for your database.

Then you can connect psql to your new database and create your tables from the statements you've written in tournament.sql. You can do this in either of two ways:

1. Paste each statement in to psql.

2. Use the command \i tournament.sql to import the whole file into psql at once.

Remember, if you get your database into a bad state you can always drop tables or the whole database to clear it out.

Design Notes

Rely on the unit tests as you write your code. If you implement the functions in the order they appear in the file, the test suite can give you incremental progress information.

The goal of the Swiss pairings system is to pair each player with an opponent who has won the same number of matches, or as close as possible.

You can assume that the number of players in a tournament is an even number. This means that no player will be left out of a round.

Your code and database only needs to support a single tournament at a time. All players who are in the database will participate in the tournament, and when you want to run a new tournament, all the game records from the previous tournament will need to be deleted. In one of the extra-credit options for this project, you can extend this program to support multiple tournaments.

Functions in tournament.py

registerPlayer(name)

Adds a player to the tournament by putting an entry in the database. The database should assign an ID number to the player. Different players may have the same names but will receive different ID numbers.

countPlayers()

Returns the number of currently registered players. This function should not use the Python len() function; it should have the database count the players.

deletePlayers()

Clear out all the player records from the database.

reportMatch(winner, loser)

Stores the outcome of a single match between two players in the database.

deleteMatches()

Clear out all the match records from the database.

playerStandings()

Returns a list of (id, name, wins, matches) for each player, sorted by the number of wins each player has.

swissPairings()

Given the existing set of registered players and the matches they have played, generates and returns a list of pairings according to the Swiss system. Each pairing is a tuple (id1, name1, id2, name2), giving the ID and name of the paired players. For instance, if there are eight registered players, this function should return four pairings. This function should use playerStandings to find the ranking of players.

Extra credit:

- Prevent rematches between players.

- Don’t assume an even number of players. If there is an odd number of players, assign one player a “bye” (skipped round). A bye counts as a free win. A player should not receive more than one bye in a tournament.
- Support games where a draw (tied game) is possible. This will require changing the arguments to reportMatch.
- When two players have the same number of wins, rank them according to OMW (Opponent Match Wins), the total number of wins by players they have played against.
- Support more than one tournament in the database, so matches do not have to be deleted between tournaments. This will require distinguishing between “a registered player” and “a player who has entered in tournament #123”, so it will require changes to the database schema.
- You may refer to outside resources to devise your pairing algorithm. Wizards of the Coast has prepared **simple instructions (http://www.wizards.com/dci/downloads/swiss_pairings.pdf)**, and more details can be found in **resources linked to in the reference section (http://en.wikipedia.org/wiki/Swiss-system_tournament#References)** of Wikipedia's article on Swiss tournaments.