Tournament Results: Getting Started

PDF Download

In Project 2, you will be writing a Python module that uses the PostgreSQL database to keep track of players and matches in a game tournament.

Project 2 was designed to teach you how to create and use databases through the use of database schemas and how to manipulate the data inside the database. This project has two parts: defining the database schema (SQL table definitions), and writing code that will use it to track a Swiss tournament.

Getting Started

1. You will complete this project within the Vagrant virtual machine we've provided and configured for you. If you would like to review that before moving on refer to the course materials for help with installing Vagrant and Virtual Box, and previously recorded office hours where we'll show you how to use Vagrant.
2. Next clone the fullstack-nanodegree-vm repository to your local machine.
3. Now, lets explore the starter code for this project provided within the VM: `cd` into `/vagrant/tournament` where you will see there are 3 files you have to work with on this project:
    - `tournament.sql` is where you will put the database schema, in the form of SQL create table commands
    - `tournament.py` is where you will put the code for your tournament module.
    - `tournament_test.py` contains test functions that will test the functions you've written in intournament.py
4. To run the Vagrant VM, in the terminal use the command `vagrant up` followed by `vagrant ssh.` Remember, once you have executed the `vagrant ssh` command, you will want to execute `cd /vagrant` to change directory to the sync folders.
5. The Vagrant VM provided in the fullstack repo already has PostgreSQL server installed, as well as the psql command line interface (CLI).
6. The very first time we start working on this project, no database will exist - so first, we'll need to create the SQL database for our tournament project. We can do this using `psql` or in `tournament.sql.`
7. `tournament.sql` is where we'll create our database schema and views; we also have the option of creating the database and tables in this file.
8. With the psql command line interface (cli), you can run any SQL statement using the tables in the connected database. Make sure to end SQL statements with a semicolon, which is not always required from Python.

9. To build and access the database we run `psql` followed by `\i tournament.sql`

tournament.sql

First off, we need to create the database and connect to it using the commands:

```
CREATE DATABASE tournament;
\c tournament;
```

We follow that up with the tables that we will be working with, namely Players and Matches. We can create them with the command:

```
CREATE TABLE [table name](....);
```

Our Players table can have a name and id set up and our matches table can have say p1 and p2 for the 2 players and a winner column which references the id in players.

Once we have our database and tables set up, the next steps moving forward would be to incorporate views and set up 3 queries in our .sql file for:

- Finding the number of matches each player has played.
- The number of wins for each player.
- The player standings.

Setting up your database:
(this is assuming your tournament.sql file has the Create database, \c tournament and Create tables commands in it, you can do this explicitly via the terminal as well)
For example:

CREATE DATABASE dbname;

\c dbname;

CREATE TABLE tablename (
…….
);

```
vagrant@vagrant-ubuntu-trusty-32:/vagrant/tournament$ psql
vagrant=> \i tournament.sql
You are now connected to database "tournament" as user "vagrant".
tournament=>
```

tournament.py and tournament_test.py

Rely on the unit tests in `tournament_test.py` as you write your code. Writing your tournament module should be done in conjunction with testing using the `tournament_test.py` file; each function has a corresponding test function. If you implement the functions in the order they appear, the test suite can help you incrementally test your code as you write each function.

Recall the sequence of SQL query commands needed to manipulate database records:

```
conn = connect()
c = conn.cursor()
c.execute("your query;")
conn.commit()
conn.close()
```

The various functions in `tournament.py` and their corresponding test functions in `tournament_test.py` are:

`tournament.py` **function**	`tournament_test.py` **test function**
`connect` Meant to connect to the database. Already set up for you.	
`deleteMatches` Remove all the matches records from the database.	`testDeleteMatches`
`deletePlayers` Remove all the player records from the database.	`testDelete`
`countPlayers` Returns the number of players currently registered	`testCount`
`registerPlayer --` Adds a player to the tournament database.	`testRegister`, `testRegisterCountDelete`
`playerStandings --` Returns a list of the players and their win records, sorted by wins. You can use the player standings table created in your .sql file for reference.	`testStandingsBeforeMatches`
`reportMatch` This is to simply populate the matches table and record the winner and loser as (winner,loser) in the insert statement.	`testReportMatches`
`swissPairings` Returns a list of pairs of players for the next round of a match. Here all we are doing is the pairing of alternate players from the player standings table, zipping them up and appending them to a list with values: (id1, name1, id2, name2)	`testPairings`

Running your project!

Once you have your .sql and .py files set up, it's a good idea to test them out against the testing file provided to you (tournament_test.py). To run the series of tests defined in this test suite, run the program from the command line `>> python tournament_test`.

```
if __name__ == '__main__':
  testDeleteMatches()
  testDelete()
  testCount()
  testRegister()
  testRegisterCountDelete()
  testStandingsBeforeMatches()
  testReportMatches()
  testPairings()
  print "Success!  All tests pass!"
```

And you should be able to see the following output once all your tests have passed:

```
vagrant@vagrant-ubuntu-trusty-32:/vagrant/tournament$ python
tournament_test.py
1. Old matches can be deleted.
2. Player records can be deleted.
3. After deleting, countPlayers() returns zero.
4. After registering a player, countPlayers() returns 1.
5. Players can be registered and deleted.
6. Newly registered players appear in the standings with no matches.
7. After a match, players have updated standings.
8. After one match, players with one win are paired.
Success!  All tests pass!
vagrant@vagrant-ubuntu-trusty-32:/vagrant/tournament$
```

To Submit

Once you have finished your project, go to this link here. If you have a Github account (which we recommend), connect with Github to get started. If you do not have a Github account, follow the instructions here for Mac OS X 10.0 or later, here for Windows 7, 8, or 8.1, or here for anything else. These links will help you create a Github account to submit your project.

If you run into any trouble, send us an e-mail at fullstack-project@udacity.com, and we will be more than happy to help you.

Example of a 16 Player Swiss Tournament:

First round pairing is by random draw. For example, with 16 players they would be matched into 8 random pairs for the first round. For now, assume all games have a winner, and there are no draws.

After the first round, there will be a group of 8 players with a score of 1 (win), and a group of 8 players with a score of 0 (loss). For the 2nd round, players in each scoring group will be paired against each other – 1’s versus 1’s and 0’s versus 0’s.

After round 2, there will be three scoring groups:
4 players who have won both games and have 2 points
8 players who have won a game and lost a game and have 1 point
4 players who have lost both games and have no points.

Again, for round 3, players are paired with players in their scoring group. After the third round, the typical scoring groups will be:
2 players who have won 3 games (3 points)
6 players with 2 wins (2 points)
6 players with 1 win (1 point)
2 players with no wins (0 points)

For the fourth (and in this case final) round, the process repeats, and players are matched with others in their scoring group. Note that there are only 2 players who have won all of their games so far – they will be matched against each other for the "championship" game. After the final round, we’ll have something that looks like this:
1 player with 4 points – the winner!
4 players with 3 points – tied for second place
6 players with 2 points
4 players with 1 point
1 player with 0 points

The Swiss system produces a clear winner in just a few rounds, no-one is eliminated and almost everyone wins at least one game, but there are many ties to deal with.