1. Preliminaries

Under Resources→Lab4 on Piazza, there are some files that are discussed in this document. Two of the files are create.sql script and lab4_load_data.sql. The create.sql script creates all tables within the schema Lab4; otherwise, it is the same as the schema in our create.sql solution to Lab2. All the constraints that were in our Lab2 solution are included, but Lab3's new General constraints and revised Foreign Key constraints are not in this schema.

lab4_load_data.sql loads data into those tables, just as similar files did for previous Lab Assignments. Alter your search path so that you can work with the tables without qualifying them with the schema name:

```
ALTER ROLE <username> SET SEARCH_PATH TO Lab4;
```

You must log out and log back in for this to take effect. To verify your search path, use:

```
SHOW SEARCH_PATH;
```

Note: It is important that you do not change the names of the tables. Otherwise, your application may not pass our tests, and you will not get any points for this assignment.

2. Instructions for database access from Python

An important files under Resources→Lab4 is *runVolleyballApplication.py*. That file is not executable as is. You will have to complete it to make it compilable, including writing three Python functions that are described in Section 4 of this document. You will also have to write a Stored Function that is used by one of those Python functions; that Stored Function is described in Section 5 of this document. As announced at the beginning of the quarter, we assume that CSE 182 students are familiar with Python 3. *runVolleyballApplication.py* will be the only file in your Python program.

Assuming that *runVolleyballApplication.py* is in your current directory, you can execute it (assuming that it is correct Python) with the following command (where the ">" character represents the Unix prompt):

> *python3 runVolleyballApplication.py <your_userid> <your_password>*

where <your_userid> and <your_password> are replaced by your userid and your password for our PostgreSQL database.

[Do not put your userid or your password in your program code, and do not include the < and > symbols, just the userid and password. We will run your program as ourselves, not as you.]

How and where you develop your program is up to you, but we will run (and grade) your program using these commands on unix.ucsc.edu. So you must ensure that your program works in that environment. Don't try to change your grade by telling us that your program failed in our environment, but worked in your own environment; if you do, we'll point you to this Lab4 comment.

3. Goal

The fourth lab project puts the database you have created to practical use. You will implement part of an application front-end to the database. As good programming practice, your functions should catch erroneous parameters (and impossible situations), such as a value of maxFired that's not positive in *fireSomePlayers*. For Lab4, if a parameter supplied to one of your functions is invalid, you should print an appropriate error message, and call sys.exit(-1). In practice, application code should be more robust than that, continuing execution despite erroneous parameters. But for Lab4 we're keeping things simple and just exiting if there are errors, since your runVolleyballApplication.py code is supposed to be executing your functions with valid parameters on a valid database instance.

4. Description of the three Python functions in the runVolleyballApplication.py file that interact with the database

The *runVolleyballApplication.py* that you've been given contains skeletons for three Python functions that interact with the database using psycop2.

These three Python functions are described below. The first argument for all of these Python function is your connection to the database.

- *teamPlayerCount:* The *teamPlayerCount* Python function returns the number of players who are the team with a specified name. The arguments for *teamPlayerCount* are the database connection and the name of the team.

  It is not an error if there is no team with that name in the Teams table. It is also not an error if there is a team with that name, but there are no players on that team. In both of these circumstances, *teamPlayerCount* should return 0.

- *updateTeamCity:* teamCity is an attribute of the Teams table. Sometimes a team moves to a different city.

  Besides the database connection, the *updateTeamCity* Python function has two arguments, an integer argument theTeamID and a string argument, newTeamCity. For every team in the Teams table (if any) whose teamID equals theTeamID, *updateTeamCity* should change that team's teamCity to be newTeamCity.

  There might be no team whose teamID equals theTeamID (that's not an error), and there also might be one team whose teamID equals theTeamID, since teamID is the Primary Key of the Teams table. *updateTeamCity* should return the number of teams whose teamCity was updated.

- *fireSomePlayers*: Besides the database connection, this Python function has one integer parameters, maxFired. A value of maxFired that's not positive is invalid, and you should call sys.exit(-1) from fireSomePlayers if an invalid maxFired value has been provided.

  fireSomePlayers invokes a Stored Function, *fireSomePlayersFunction*, that you will need to implement and store in the database according to the description in Section 5. The Stored Function *fireSomePlayersFunction* should just have one parameter, maxFired, so don't the database connection is not a parameter for this Stored Function.

  Section 5 explains what "firing" means, and also tells you which Players should be fired, The *fireSomePlayers* Python function should return the same integer result that the *fireSomePlayersFunction* Stored Function returns.

  The *fireSomePlayers* function must only invoke the Stored Function *fireSomePlayersFunction*, which does all of the work for this part of the assignment; *fireSomePlayers* should not do the work itself.

Each of these three functions is annotated in the runVolleyballApplication.py file we've given you, with comments providing a description of what it is supposed to do (repeating the above descriptions). Your task is to implement functions that match those descriptions.

The following psycopg2-related links appear in Lecture 11. All are helpful; none are perfect introductory tutorials.

- Python PostgreSQL Tutorial Using psycopg2 on the PYnative site, showing some SELECT, INSERT, DELETE and UPDATE statement. This one uses Python 3, which is why it's the first link on the list.
- psycopg2 Tutorial on the PostgreSQL site. Unfortunately, code is written in Python 2, not Python 3, but it explains use of psycopg2 pretty well.
- psychopg2 usage examples, on the psycopg site. Once again, code is written in Python 2, not Python 3, but it explains use of psycopg2.

5. Stored Function

As Section 4 mentioned, you should write a Stored Function (not a Stored Procedure) called *fireSomePlayersFunction* that has one integer parameters, maxFired. For each fired player, *fireSomePlayersFunction* will change the teamID for that player's tuple in the Players table to NULL. This Stored Function should count the number of the players who are fired, and it should return that fired count. The fired count returned will never be more than maxFired. (It might equal maxFired, but it might also be less than maxFired.)

Here's how to determine which players should be fired. rating is an attribute in the Players table. minutesPlayed is an attribute in the GamePlayers table. We'll say that a player is "at risk" if the following three things all hold for that player:

- the player's team isn't NULL (so they're currently on a team), and
  the player's status is 'L' (low player status) and
- the player's total minutes played (adding up their minutesPlayed in GamePlayers) is more than 60.

Some of the "at risk" players will be fired … but some won't be fired. How do you decide which "at risk" players will be fired? salary is an attribute in the Persons table. You'll look at the "at risk" players, and fire the maxFired players who have the highest salaries.

Examples showing how *fireSomePlayersFunction* should work:

- If there are 10 "at risk" players and maxFired is 6, you'll fire the 6 "at risk" players who have the highest salary. If the players with the sixth and seventh highest salary have exactly the same salary, it's okay to fire either one (but only fire one). The fired count value returned will be 6.
- If there are 10 "at risk" players and maxFired is 10, you'll fire all 10 "at risk" players, and the fired count value returned will be 10.
- But if there are 10 "at risk" players and maxFired is 12, you'll fire all 10 "at risk" players, and the fired count value returned will be 10, not 12.

Write the code to create the Stored Function, and save it to a text file named *fireSomePlayersFunction.pgsql*. To create the Stored Function *fireSomePlayersFunction*, issue the psql command:

```
\i fireSomePlayersFunction.pgsql
```

at the server prompt. If the creation goes through successfully, then the server should respond with the message "CREATE FUNCTION". You will need to call the Stored Function from the *fireSomePlayers* function in your Python program, as described in the previous section, so you'll need to create the Stored Function before you run your program. You should include the fireSomePlayersFunction.pgsql source file in the zip file of your submission, together with your versions of the Python file runVolleyballApplication.py that was described in Section 4. See Section 7 for detailed submission instructions.

A guide for defining Stored Functions for PostgreSQL can be found here on the PostgreSQL site. PostgreSQL Stored Functions have some significant syntactic differences from the PSM stored procedures/functions that were described in Lecture. We've posted a "StoredFunction_Info _for_PostgreSQL" file on Piazza under Resources→General Information that should help you write

PostgreSQL Stored Functions in PL/pgSQL (not in PSM). For Lab4, you should write a Stored Function that has only IN parameters; that's legal in both PSM and PostgreSQL.

6. Testing

Within main for *runVolleyballApplication.py*, you should write several tests of the Python functions described in Section 4. You might also want to write your own tests, but only the following tests should be included in the *runVolleyballApplication.py* file that you submit in your Lab4 solution.

- Write two tests in *runVolleyballApplication.py* of the Python function *teamPlayerCount*.
  - The first test should be for name 'Birds'.
  - The second test should be for name 'Skyscrapers'.

  For each test of *teamPlayerCount*, the format of your output should be:

  Count of players on team <name of the team> is <the player count>

  where you print the name of the team provided and the player count you computed.

- Write two tests in *runVolleyballApplication.py* of the Python function *updateTeamCity*.
  - The first test should be for theTeamID 202 and newTeamCity 'Las Vegas'
  - The second test should be for theTeamID 208 and newTeamCity 'Toronto'

  For each test of *updateTeamCity*, print out its result (which is the number of Teams tuples that were updated) using the following format:

  Number of teams with teamID <theTeamID> who moved to city <newTeamCity> is <number of tuples updated>

  where you print out the teamID provided, the newTeamCity provided, and the number of tuples that were updated (which will always be 0 or 1).

  [The output for each invocation of updateTeamCity should appear on a single line, not split into two lines.]

- Also write two tests in *runVolleyballApplication.py* of the Python function *fireSomePlayers*.
  - The first test should have maxFired value 2.
  - The second test should have maxFired value 3.

  For each test of *fireSomePlayers*, print its result (number of players fired) using the following format:

  Number of player fired in fireSomePlayers when maxFired equals <maxFired> is <number of players fired>

  where you print the value of the maxFired parameter, and the number of players who were fired.

  [The output for each invocation of fireSomePlayers should appear on a single line, not split into two lines.]

You must run all of these function tests in order, starting with the database provided by our create and load scripts. Some of these function change the database, so using the load data that we've

provided and executing the functions in order is required. Reload the original load data before you start, but you do not have to reload the data multiple times in Lab4.

Hmmm, do any of these tests affect each other? What do you think?

7. Submitting

1. Remember to add comments to your Python code so that the intent is clear.
2. Place the Python program runVolleyballApplication.py and the stored procedure declaration code *fireSomePlayersFunction.pgsql* in your working directory at unix.ucsc.edu.
3. Zip the files to a single file with name Lab4_XXXXXXX.zip where XXXXXXX is your 7-digit student ID, for example, if a student's ID is 1234567, then the file that this student submits for Lab4 should be named Lab4_1234567.zip. To create the zip file, you can use the Unix command:

   zip Lab4_1234567 runVolleyballApplication.py fireSomePlayersFunction.pgsql

   Please do not include any other files in your zip file, except perhaps for an optional README file, if you want to include additional information about your Lab4 submission.
4. Some students might want to use views to do Lab4. That's not required, but it is permitted. If you do use views, you must put the statements creating those views in a file called createVolleyballViews.sql, and include that file in your Lab4 zip file.
5. Lab4 is due on Canvas by 11:59pm on **Sunday, May 30**. Late submissions will not be accepted, and there will be no make-up Lab assignments.