1. Preliminaries

Under Resources→Lab4 on Piazza, there are some files that are discussed in this document. Two of the files are lab4_create.sql script and lab4_load_data.sql. The lab4_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 *runHorseRacingApplication.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. *runHorseRacingApplication.py* will be the only file in your Python program.

Assuming that *runHorseRacingApplication.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 runHorseRacingApplication.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, both executing SQL statements that SELECT and UPDATE, and executing a Stored Function.

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

The *runHorseRacingApplication.py* that you've been given contains skeletons for three Python functions that interact with the database using psycopg2.

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

- *winCountForHorse:* The *winCountForHorse* Python function returns the number of times that a horse with a particular horseID finished first in a race (that is, had finishPosition 1). The arguments for *winCountForHorse* are the database connection and a horseID.

  It is not an error if there is no horse with that ID. If there are no races in which the horse finished first, then winCountForHorse should return 0.

- *updateRacetrackAddress:* address is an attribute of the RaceTracks table. Sometimes the address of a racetrack may change.

  Besides the database connection, the *updateRacetrackAddress* Python function has two arguments, a string argument oldAddress and another string argument, newAddress. For every racetrack in the Racetracks table (if any) whose address equals oldAddress, *updateRacetrackAddress* should change that racetrack's address to be newAddress.

  There might be no racetrack whose address equals oldAddress; that's not an error. There also might be one or more racetracks whose address equals oldAddress (since multiple racetracks might have the same address). *updateRacetrackAddress* should return the number of racetracks whose address was updated.

- *disqualifyHorseInRace*: Besides the database connection, this Python function has four other parameters:
  - theHorseID which is an integer,
  - theRacetrackID which is an integer,
  - theRaceDate which is a date,
  - theRaceNum, which is an integer.

  disqualifyHorseInRace invokes a Stored Function, *disqualifyHorseInRaceFunction*, that you will need to implement and store in the database according to the description in Section 5. The Stored Function *disqualifyHorseInRaceFunction* has all the same parameters as *disqualifyHorseInRace* (except for the database connection, which is not a parameter for the Stored Function), and it returns an integer.

  Section 5 tells you what "disqualifying a horse in a race" mean, and explains the integer value that *disqualifyHorseInRaceFunction* returns. The *disqualifyHorseInRace* Python function returns the same integer value that the *disqualifyHorseInRaceFunction* Stored Function returns.

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

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

The following psycopg2-related links appear in Lecture 10. 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 *disqualifyHorseInRaceFunction* that has parameters identifying **a particular horse** (theHorseID) and a particular race (theRacetrackID, theRaceDate, theRaceNum). The database connection is not a parameter for the Stored Function.

The HorseRaceResults table is supposed to have a tuple that gives the finishPosition for (theRacetrackID, theRaceDate, theRaceNum, theHorse). If there is no tuple for that horse in that race, then *disqualifyHorseInRaceFunction* should return the value -1.

If the finishPosition of that horse has the value NULL, then that horse has already been disqualified, and *disqualifyHorseInRaceFunction* should return the value -2.

Now that we know the finishPosition of that horse in that race, we're going to disqualify that horse. What does that mean? We're going to change that horse's finishPosition in that race to NULL, and we're going to improve the finishPosition of all the horses who finished after it, decreasing their finishPosition by 1. Here are some examples of how this works if that horse had finishPosition 4 in that race.

a) If there are 2 horses who had bigger finishPosition values 5 and 6 in that race, then the finishPosition for those horses in that race should be improved to 4 and 5.
b) If there were 3 horses who had bigger finishPosition values 5, 5 and 7 in that race, then the finishPosition for those horses in that race should be improved to 4, 4 and 6.
c) If there were no horses who had bigger finishPosition values in that race, then no horses should have their finishPosition improved in that race.

disqualifyHorseInRaceFunction should return the number of horses whose finishPosition has been improved. So the values it should return for the three examples above are a) 2, b) 3 and c) 0.

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

```
\i disqualifyHorseInRaceFunction.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 *disqualifyHorseInRace* 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 disqualifyHorseInRaceFunction.pgsql source file in the zip file of your Submission, together with your versions of the Python file runHorseRacingApplication.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 *runHorseRacingApplication.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 *runHorseRacingApplication.py* file that you submit in your Lab4 solution.

- Write two tests in *runHorseRacingApplication.py* of the Python function *winCountForHorse*.
    - The first test should be for horseID 526
    - The second test should be for horseID 555.

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

  Horse <the ID of the horse> won <the number of wins for that horse> races

  where you print the ID of the horse provided and the win count you computed.

- Write two tests in *runHorseRacingApplication.py* of the Python function *updateRacetrackAddress*.
    - The first test should be for oldAddress "Kellogg Rd 6301, Cincinnati, OH 45230" and newAddress "6301 Kellogg Road, Cincinnati, OH 45230".
    - The second test should be for oldAddress "Elmont, NY 11003" and newAddress "Belmont Park, NY 11003".

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

  Number of racetracks with whose address was changed from<oldAddress> to <newAddress> is <number of tuples updated>

  where you include out the oldAddress and newAddress provided, and the number of tuples that were Racetracks tuples that were updated.

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

- Also write three tests in *runHorseRacingApplication.py* of the Python function *disqualifyHorseInRace*.
    - The first test should be for horseID 552, racetrackID 3008, raceDate August 11, 2021 and raceNum 1.
    - The second test should be for horseID 553, racetrackID 3008, raceDate August 11, 2021 and raceNum 1.
    - The third test should be for horseID 575, racetrackID 3008, raceDate August 11, 2021 and raceNum 1.

  If *disqualifyHorseInRace* returns a negative value, then you should print out the values of its parameters, with a message explaining the error that occurred. You may choose the format yourself, as long as the error explanation is clear. You should continue executing further tests, even if *disqualifyHorseInRace* returns a negative value.

Otherwise, print out the parameters of *disqualifyHorseInRace* together with the returned value, which is the number of horses whose finishPosition was improved (which could be 0). You may choose the format yourself, as long as it's clear which values are which.

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?

Note: Since we've included the datetime() class in the Python skeleton for *runHorseRacingApplication.py*, writing datetime.date(2021, 8, 11) giving you a Python date object that corresponds to August 11, 2021, and that Python date object can be the argument for a PL/pgSQL stored function which expects a date parameter.

7. Submitting

1. Remember to add comments to your Python code so that the intent is clear.
2. Place the Python program runHorseRacingApplication.py and the stored procedure declaration code *disqualifyHorseInRaceFunction.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 runHorseRacingApplication.py disqualifyHorseInRaceFunction.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 createHorseRacingViews.sql, and include that file in your Lab4 zip file.
5. Lab4 is due on Canvas by 11:59pm on **Tuesday, May 31**. Late submissions will not be accepted, and there will be no make-up Lab assignments.