1 Preliminaries

In this lab, you will work with a Volleyball database schema similar to the schema that you used in Lab2. We've provided a lab3_create.sql script for you to use (which is similar to, but not quite the same as the create.sql in our Lab2 solution), so that everyone can start from the same place. Please remember to DROP and CREATE the Lab3 schema before running that script (as you did in previous labs), and also execute:

```
ALTER ROLE yourlogin SET SEARCH_PATH TO Lab3;
```

so that you'll always be using the Lab3 schema without having to mention it whenever you refer to a table.

You will need to log out and log back in to the server for this default schema change to take effect. (Students often forget to do this.)

We'll also provide a lab3_data_loading.sql script that will load data into your tables. (The timestamps in the Lab3 load data script do not include time zones.) You'll need to run that script before executing Lab3. The command to execute a script is: \i <filename>

In Lab3, you will be required to combine new data (as explained below) into one of the tables. You will need to add some new constraints to the database and do some unit testing to see that the constraints are followed. You will also create and query views, and create an index.

New goals for Lab3:

1. Perform SQL to "combine data" from two tables
2. Add foreign key constraints
3. Add general constraints
4. Write unit tests for constraints
5. Create and query views
6. Create an index

There are lots of parts in this assignment, but none of them should be difficult. Lab3 will be discussed during the Lab Sections before the due date, Sunday, May 16. The due date of Lab3 is 3 weeks after the due date of Lab2 because of the Midterm, and also because we didn't cover all of the Lab3 topics before the Midterm.

2. Description

2.1 Tables with Primary Keys for Lab3

The primary key for each table is underlined. There's one table that you haven't seen before.

Persons(personID, name, address, salary, canBePlayer, canBeCoach)

Teams(teamID, name, coachID, teamCity, teamColor, totalWins, totalLosses)

Players(playerID, teamID, joinDate, rating, isStarter)

Games(gameID, gameDate, homeTeam, visitorTeam, homePoints, visitorPoints)

GamePlayers(gameID, playerID, minutesPlayed)

PersonChanges(personID, name, address, salary)

In the lab3_create.sql file that we've provided under Resources→Lab3, the first 5 tables are similar to the tables were in our Lab2 solution, except that the NULL and UNIQUE constraints from Lab2 are **not** included. Also, lab3_create.sql does not have the Foreign Key Constraints on coachID in Teams, on gameID in GamePlayers, and on playerID in GamePlayers that were in our Lab2 solution.

In practice, primary keys, unique constraints and other constraints are almost always entered when tables are created, not added later. lab3_create.sql handles some constraints for you, but, you will be adding some additional constraints to these tables in Lab3, as described below.

Note also that there is an additional table, PersonChanges, in the lab3_create.sql file that has most (but not all) of the attributes that are in the Persons table. We'll say more about PersonChanges below.

Under Resources→Lab3, you'll also be given a load script named lab3_data_loading.sql that loads tuples into the tables of the schema. **You must run both lab3_create.sql and lab3_data_loading.sql before you run the parts of Lab3 that are described below.**

2.2 Combine Data

Write a file, *combine.sql* (which should have multiple SQL statements that are in a Serializable transaction) that will do the following. For each tuple in PersonChanges, there might already be a tuple in the Persons that has the same primary key (that is, the same value for personID). If there **isn't** a tuple in Persons with the same primary key, then this is a new person that should be inserted into Persons. If there already **is** a tuple in Persons with that primary key, then this is an update of information about that person. So here are the actions that you should take.

- If there **isn't** already a tuple in the Persons table which has that personID, then insert a tuple into the Persons table which has the personID, name, address and salary which are in that PersonChanges tuple. As part of your insert, the value of canBePlayer should be set to TRUE, and the value of canBeCoach should be set to NULL.

- If there already **is** a tuple in the Persons table which has that personID, then update the tuple in Persons that has that personID. Update name, address and salary for that existing Persons tuple based on the values in that PersonChanges tuple, and set both canBePlayer and canBeCoach to FALSE.

Your transaction may have multiple statements in it. The SQL constructs that we've already discussed in class are sufficient for you to do this part (which is one of the hardest parts of Lab3).

For a legal INSERT statement, the second output value indicates how many rows were inserted. (Please ignore the first output value, which is an internal PostgreSQL value.) For a legal UPDATE statement, the output indicates how many rows were updated.

2.3 Add Foreign Key Constraints

Important: Before running Sections 2.3, 2.4 and 2.5, recreate the Lab3 schema using the *lab3_create.sql* script, and load the data using the script *lab3_data_loading.sql*. That way, any database changes that you've done for Combine won't propagate to these other parts of Lab3.

Here's a description of the Foreign Keys that you need to add for this assignment. (Foreign Key Constraints are also referred to as Referential Integrity constraints. The lab3_create.sql file that we've provided for Lab3 includes the same Referential Integrity constraints that were in the Lab2 solution, but you're asked to use ALTER to add additional constraints to the Volleyball schema.

The load data that you're provided with should not cause any errors when you add these constraint. Just add the constraints listed below, exactly as described, even if you think that additional Referential Integrity constraints should exist. Note that (for example) when we say that every coach (coachID) in the Teams table must appear in the Persons table, that means that the coachID attribute of the Teams table is a Foreign Key referring to the Primary Key (personID) of the Persons table.

- Each coach (coachID) in the Teams table must appear in the Persons table as a personID. (Explanation of what that means appear in the above paragraph.) If a Persons tuple is deleted and there is a Teams tuples that refers to that employee, then the deletion should be rejected. Also, if the Primary Key of an Persons tuple is updated, and there is a Team whose coach corresponds to that person, then the update should also be rejected.

- Each game (gameID) that's in the GamePlayers table must appear in the Games table as a gameID. If a tuple in the Games table is deleted, then all GamePlayers tuples in which that game (gameID) appears should also be deleted. If the Primary Key (gameID) of a Games tuple is updated, then all

GamePlayers tuples which have that gameID should also be updated, getting the same new value for their gameID.

- Each player (playerID) that's in the GamePlayers table must appear in the Players table as a playerID. If a tuple in the Players table is deleted, then that deletion should be rejected if there are tuples in the GamePlayers table which match the deleted playerID. If the Primary Key (playerID) of a Players tuple is updated, then all GamePlayers tuples that have the corresponding playerID should also be updated, getting the same new value for their playerID.

Write commands to add foreign key constraints in the same order that the foreign keys are described above. Be sure to give names to each of the three foreign key constraints when you create it; you may choose any names that you like. Save your commands to the file *foreign.sql*

For legal ALTER TABLE statements, the output is ALTER TABLE.

2.4 Add General Constraints

General constraints for Lab3 are the following:

1.In GamePlayers, minutesPlayed must be greater than or equal to zero. Give a name to this constraint when you create it. We recommend that you use the name reasonable_salary, but you may use another name.

2. In Players, the value of rating must be either 'L', 'M', 'H' or NULL. Give a name to this constraint when you create it. We recommend that you use the name legal_rating, but you may use another name.

3. In Games, if homePoints is NULL, then visitorPoints must also be NULL. Give a name to this constraint when you create it. We recommend that you use the name null_twice, but you may use another name.

Write commands to add general constraints in the order the constraints are described above, and save your commands to the file *general.sq*l. (Note that UNKNOWN for a Check constraint is okay, but FALSE isn't.)

For legal ALTER TABLE statements, the output is ALTER TABLE.

2.5 Write Unit Tests

Unit tests are important for verifying that your constraints are working as you expect. We will require tests for just a few common cases, but there are many more unit tests that are possible.

For each of the 3 foreign key constraints specified in section 2.3, write one unit test:

- An INSERT command that violates the foreign key constraint (and elicits an error).

Also, for each of the 3 general constraints, write 2 unit tests:

- An UPDATE command that meets the constraint.
- An UPDATE command that violates the constraint (and elicits an error).

Save these 3 + 6 = 9 unit tests, in the file *unittests.sql*. The 3 foreign key unit tests should come first in order, then the 2 unit tests for the first general constraint, etc.

For legal INSERT and UPDATE statements that do not violate constraints, the output is as described in Section 2.2 for combine.sql. For INSERT and UPDATE statements that violate constraints, the output describes the constraint violation. For unittests.sql, some of your INSERT and UPDATE statements are supposed to violate constraints.

2.6 Working with Views

2.6.1 Create Two Views

a) totalWins is an attribute of the Teams table. But there is another way that you could compute the number of wins that a team has. We'll say that computedWins for a team is the number of games in which that team had more points than the other team.

Create a view called WinsDisagree in which the attributes are teamID, teamName, totalWins and computedWins. But a tuple should only appear in WinsDisagree if totalWins and computedWins don't have the same value. Your view should have no duplicates in it.

b) totalLosses is another attribute of the Teams table. But there is another way that you could compute the number of losses that a team has. We'll say that computedLosses for a team is the number of games in which that team had fewer points than the other team.

Create a view called LossesDisagree in which the attributes are teamID, teamName, totalLosses and computedLosses. But a tuple should only appear in LossesDisagree if totalLosses and computedLosses don't have the same value. Your view should have no duplicates in it.

Save the SQL statements that create the WinsDisagree view and the LosssesDisagree view in a file called *createviews.sql*

For a statement that successfully creates a view, the output is CREATE VIEW.

For WinsDisagree,, you can assume that each team has won at least one game, and for LossesDisagree you can assume that each team has lost at least one game. This matters because a team that has not won a game will never appear in the WinsDisagree view, and a team that has not lost a game will never appear in the LossesDisagree view! Later in the quarter, we'll discuss two ways of dealing with GROUP BY when there are zero tuples in a group and we'd like our result to show that.

2.6.2 Query Views

For this part of Lab3, you'll write a script called *queryviews.sql* that contains a query that uses both the WinsDisagree view and the LosssesDisagree view (and possibly some tables). In addition to that query, you'll must also include some comments in the queryviews.sql script; we'll describe those necessary comments below.

Write and run a SQL query over those views to answer the following "Messy Statistics" question. You may want to use some tables to write this query, but be sure to use both of the views.

> There may be some teams that have tuples in both the WinsDisagree view and the LossesDisagree view. We'll say that those teams have "Messy Statistics.
>
> Write a query that finds all the teams that have Messy Statistics. The output of your "Messy Statistics" query should be the teamID, the teamName, the difference totalWins minus computedWins, the difference totalLosses minus ComputedLosses, and the number of players on the team. The attributes in you result should appear as teamID, teamName, winDiff, lossDiff and numPlayers.

Important: Before running this query, recreate the Lab3 schema once again using the *lab3_create.sql* script, and load the data using the script *lab3_data_loading.sql*. That way, any changes that you've done for previous parts of Lab3 (e.g., Unit Test) won't affect the results of this query. Then write the results of the "Messy Statistics" query in a comment. *The format of that comment is not important; it just has to have all the right information in it.*

Next, write commands that delete just the tuples that have the following Primary Key values from the Games table (whose Primary Key is gameID):

- The tuple whose Primary Key is 10005
- The tuple whose Primary Key is 10001

Run the "Messy Statistics" query once again after those deletions. Write the output of the query in a second comment. Do you get a different answer?

You need to submit a script named *queryviews.sql* containing your query on the views. In that queryviews.sql file you must also include:

- the comment with the output of the query on the provided data before the deletions,
- the SQL statements that delete the tuples indicated above,
- and a second comment with the second output of the same query after the deletions.

You do not need to replicate the query twice in the *queryviews.sql* file (but you won't be penalized if you do).

Aren't you glad that you had the WinsDisagree and LosssesDisagree views? It probably was easier to write this query using those views than it would have been if you hadn't had those views.

You may use an additional view (or views) to write the "Messy Statistics" query, if you'd like. (That might be a good idea!) But if you do, then you'll have to include the statements creating those views inside the queryviews.sql file (not inside the createviews.sql file) before your query.

The output of a query that's legal SQL is the result of executing that query.

2.7 Create an Index

Indexes are data structures used by the database to improve query performance. Locating the tuples in the Games table for a particular homeTeam and visitorTeam might be slow if the database system has to search the entire Games table. To speed up that search, create an index named SearchGames over the homeTeam and visitorTeam columns (in that order) of the Games table. Save the command in the file *createindex.sql*.

Of course, you can run the same SQL statements whether or not this index exists; having indexes on a table might improve the performance of some SQL statements when that table is very large. This index might make it faster to determine which Games involve a particular homeTeam and visitorTeam, or help find all Games involving a particular homeTeam.

For this assignment, you need not do any searches that use the index, but if you're interested, you might want to do searches with and without the index, and look at query plans using EXPLAIN to see how queries are executed. Please refer to the documentation of PostgreSQL on EXPLAIN that's at https://www.postgresql.org/docs/12/sql-explain.html

3 Testing

Before you submit, login to your database via psql and execute the provided database creation and load scripts, and then test your seven scripts (combine.sql foreign.sql general.sql unittests.sql createviews.sql queryviews.sql createindex.sql). Note that there are two sections in this document (both labeled **Important**) where you are told to recreate the schema and reload the data before running that section, so that updates you performed earlier won't affect that section. Please be sure that you follow these directions, since your answers may be incorrect if you don't.

4 Submitting

1. Save your scripts indicated above as combine.sql foreign.sql general.sql unittests.sql createviews.sql queryviews.sql createindex.sql. You may add informative comments inside your scripts if you want (the server interprets lines that start with two hyphens as comment lines).
2. Zip the files to a single file with name Lab3_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 Lab3 should be named Lab3_1234567.zip To create the zip file you can use the Unix command:

   ```
   zip Lab3_1234567 combine.sql foreign.sql general.sql unittests.sql createviews.sql queryviews.sql createindex.sql
   ```

   (Of course, you use your own student ID, not 1234567.)
3. You should already know how to transfer the files from the UNIX timeshare to your local machine before submitting to Canvas.
4. Lab3 is due on Canvas by 11:59pm on Sunday, May 16. Late submissions will not be accepted, and there will be no make-up Lab assignments.