1 Preliminaries

In this lab, you will work with a Conference Management 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.)

Soon, we'll also provide a lab3_data_loading.sql script that will load data into your tables. 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 a view, 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 a view
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, November 14. The due date of Lab3 is 3 weeks after the due date of Lab2 because students want to study for 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. The attributes are the same ones from Lab2, but there's one table that you haven't seen before.

Persons(personID, lastName, firstName, email, affiliation, isStudent)

Conferences(conferenceName, year, conferenceDate, regularAttendeeCost, studentAttendeeCost, submissionDueDate, reviewDueDate, importance)

Submissions(conferenceName, year, submissionID, numPages, submitDate, wasAccepted, submissionTitle, dateAccepted, datePublished)

Authors(authorID, conferenceName, year, submissionID, authorPosition)

Reviewers(reviewerID, conferenceName, year, reliability)

Reviews(reviewerID, conferenceName, year, submissionID, reviewDate, rating)

Attendees(attendeeID, conferenceName, year)

SubmissionChanges(conferenceName, year, submissionID, dateAccepted)

In the lab3_create.sql file that we've provided under Resources→Lab3, the first 7 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 reviewer in Reviews and on (conferenceName, year, submissionID) in Reviews that were in our Lab2 solution. (You'll create new variations of those constraints in Lab3.)

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, SubmissionChanges, in the lab3_create.sql file that has most (but not all) of the attributes that are in the Conferences table. As with Submissions, any (conferenceName, year) that's in a SubmissionChanges row must appear as a (conferenceName, year) in the Conferences table. In other words, (conferenceName, year) in SubmissionChanges is a Foreign Key referencing the Primary Key of Conferences, as you can see by looking at lab3_create.sql. We'll say more about SubmissionChanges 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.sq*l (which should have multiple SQL statements that are in a Serializable transaction) that will do the following. For each tuple in SubmissionChanges, there might already be a tuple in the Submissions table that has the same primary key (that is, the same value for conferenceName, year, submissionID). If there **isn't** a tuple in Submissions with the same primary key, then this is a new submission that should be inserted into Submissions. If there already **is** a tuple in Submissions with that primary key, then this is an update of information about that submission. So here are the actions that you should take.

- If there **isn't** already a tuple in the Submissions table which has that conferenceName, year and submissionID, then insert a tuple into the Submissions table corresponding to that SubmissionChanges tuple. Set submitDate to today's date, use the dateAccepted provided in the SubmissionChanges tuple for dateAccepted, set wasAccepted to TRUE, and set all the other attributes in Submissions to NULL. [There's an Appendix at the end of this Lab3 file that tells you how to set submitDate to today's date; it's easy.]

- If there already **is** a tuple in the Submissions table which has that conferenceName, year and submissionID, then update the tuple in Submissions that has that conferenceName and year. Update dateAccepted for that existing Submissions tuple based on the value of dateAccepted in the SubmissionChanges tuple, change wasAccepted to TRUE, and change submitDate to NULL. Don't modify any of the other attributes in that existing Submissions tuple.

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).

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 only some of the Referential Integrity constraints that were in the Lab2 solution, but you're asked to use ALTER to add additional constraints to the Conference Management 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 reviewer (reviewerID) in the Reviews table must appear in the Reviewers table, that means that the reviewerID attribute of the Reviews table is a Foreign Key referring to the Primary Key of the Reviewers table. (There already is a constraint that every reviewerID must correspond to a personID in the Persons table; we haven't changed that. But reviewerID in a Reviewers tuple could be updated to be a different personID.)

- Each reviewer (reviewerID, conferenceName, year) in the Reviews table must appear in the Reviewers table as a Primary Key, (reviewerID, conferenceName, year). (Explanation of what that means appear in the above paragraph.) If a Reviewers tuple is deleted, then all Reviews tuples that correspond to that reviewer should have all of their Foreign Key attributes set to NULL. Also, if the Primary Key of a Reviewers tuple is updated (by changing reviewerID to a different person, since a reviewer must be a person), then all reviews in Reviews for that reviewerID should also be updated to that different reviewerID.

- Each submission (conferenceName, year, submissionID) that's appears in the Reviews table must also appear in the Submissions table as the Primary Key, (conferenceName, year, submissionID), of a submission. If a tuple in the Submissions table is deleted, then all Reviews tuples that have the corresponding (conferenceName, year, submissionID) should also be deleted. If the Primary Key (conferenceName, year, submissionID) of a Submissions tuple is updated (e.g., by changing its submissionID), and there are Reviews tuples that correspond to that (conferenceName, year, submissionID), then the update of the Submissions tuple should be rejected.

Write commands to add foreign key constraints in the same order that the foreign keys are described above. Your foreign key constraints should have names, but you may choose any names that you like. Save your commands to the file *foreign.sql*

2.4 Add General Constraints

General constraints for Lab3 are:

1. In Authors, authorPosition must be positive. Please give a name to this constraint when you create it. We recommend that you use the name positive_authorPosition, but you may use a different name if you'd like.

2. In Conferences, studentAttendeeCost must be less than regularAttendeeCost. Please give a name to this constraint when you create it. We recommend that you use the name students_pay_less, but you may use a different name if you'd like.

3. In Submissions, if wasAccepted is NULL then dateAccepted must also be NULL. We recommend that you use the name consistent_acceptance, but you may use a different name if you'd like.

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.

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 2 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). You must violate that specific foreign key constraint, not any other constraint.

Also, for each of the 3 general constraints, write 2 unit tests, with 2 tests for the first general constraint, followed by 2 tests for the second general constraint, followed by 2 tests for the third general constraint.

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

Save these 2 + 6 = 8 unit tests, in the order specified above, in the file *unittests.sql*.

2.6 Working with Views

2.6.1 Create two views

We're going to make it easier for you to solve a harder version of Query 5 from Lab2 by creating two views.

Create a view called **HuntingtonReviewerConferences** that has three attributes, conferenceName, year and hCount. This view should have a tuple for each conference for which there is at least one reviewer whose last name is Huntington. (Of course, there may be many reviewers for a conference whose last name is Huntington.) The hCount attribute should be the number of reviewers for that conference whose last name is Huntington. Your view should have no duplicates in it.

Create another view called **LongConferenceSubmissionsCount** that has three attributes, conferenceName, year and longCount. For each conference that has at least one submission that is longer than 7 pages, you should count the number of submissions to that conference that are more than 7 pages long. That count should appear in the longCount attribute. Your view should have no duplicates in it.

Save the script for creating both of these views in a file called *createviews.sql*

2.6.2 Query a View

For this part of Lab3, you'll write a script called *queryviews.sql* that contains a query that uses **HuntingtonReviewerConferences**, **LongConferenceSubmissionsCount**, and (possibly) some tables. In addition to that query, you 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 "HarderThanQuery5" question. You may want to use some tables to write this query, but be sure to use the views.

> There may be conferences that appear in both views, HuntingtonReviewerConferences and LongConferenceSubmissionsCount. Both of those views have conferenceName and year attributes, which is the Primary Key of the Conferences table.
>
> Some conferences may appear in <u>both</u> views. Some of those conferences will have the same year. The output of your "HarderThanQuery5" query should be year and the number of conferences that appear in <u>both</u> views for that year. But only include a tuple in your result if:
>
> a) all the Huntington conferences for that year have exactly one Huntington reviewer, and
> b) the total number of long conference submissions for that year is at least 4.

No duplicates should appear in your result.

<u>Important</u>: 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 "HarderThanQuery5" 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 Keys from the Reviewers table:

- The Reviewers tuple whose Primary Key is (3488, 'Santa Cruz Law Review', 2021).
- The Reviewers tuple whose Primary Key is (3396, 'American Conference of Law', 2020).

Run the "HarderThanQuery5" 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 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 two views? It probably was a lot easier to write this query using those views than it would have been if you hadn't had them!

2.7 Create an Index

Indexes are data structures used by the database to improve query performance. Locating the tuples in the Reviews table for a particular conferenceName and year might be slow if the database system has to search the entire Reviews table (if the number of reviews was very large). To speed up that search, create an index named SearchForConferenceReviews over the year and conferenceName columns (in that order) of the Reviews 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 just changes the performance of SQL statements. But this index could make it faster to determine if there are any reviews for a particular year, or find all the reviews for a particular (conferenceName, year).

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, November 14. Late submissions will not be accepted, and there will be no make-up Lab assignments.

5 Appendix: Today's date

The combine.sql section of Lab3 requires that you make the submitDate value of a tuple be today's date. That's easy to do in SQL. You can use CURRENT_DATE as a constant in SQL statements the same way that you use any other date constant (although CURRENT_DATE is actually a function). But CURRENT_DATE should not be preceded by the keyword DATE.

For example, if you want to find all the Activities that took place today, you could write:

```
SELECT *
FROM Activities
WHERE day = CURRENT_DATE;
```

Similarly, you can use CURRENT_DATE in modification statements (INSERT, DELETE, UPDATE), just as you would use any other DATE constant.

If you're interested in learning more about Time and Date Functions, see Section 9.9.4 of the PostgreSQL documentation.