1 Preliminaries

In this lab, you will work with a Newspaper database schema similar to the schema that you used in Lab2. We've provided a create_lab3.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 have provided a load_lab3.sql script soon 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, Tuesday, November 21. The due date of Lab3 is 3 weeks after the due date of Lab2 because students wanted to spend time studying for Midterm2, and also because we didn't cover all of the Lab3 topics earlier.

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.

SubscriptionKinds(<u>subscriptionMode, subscriptionInterval</u>, rate, stillOffered)

Editions(<u>editionDate</u>, numArticles, numPages)

Subscribers(<u>subscriberPhone</u>, subscriberName, subscriberAddress)

Subscriptions(<u>subscriberPhone, subscriptionStartDate</u>, subscriptionMode, subscriptionInterval, paymentReceived)

Holds(<u>subscriberPhone, subscriptionStartDate, holdStartDate</u>, holdEndDate)

Articles(<u>editionDate, articleNum</u>, articleAuthor, articlePage)

ReadArticles(<u>subscriberPhone, editionDate, articleNum</u>, readInterval)

NewReadArticles(<u>subscriberPhone, editionDate, articleNum</u>, readInterval)

In the create_lab3.sql file that we've provided under Resources→Lab3, the first 7 tables are similar to the tables were in our Lab1 solution, but the NULL and UNIQUE constraints from Lab2 are **not** included. Moreover, create_lab3.sql is missing is missing both of the Foreign Key Constraints on Subscriptions that were in our Lab2 solution, and also doesn't have the Foreign Key Constraint on Holds (which says that a subscription for which there is a hold must be a subscription in Subscriptions). (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. create_lab3.sql handles some constraints for you, but you will be adding some <u>additional</u> constraints to these tables in Lab3, as described below.

Note also that there is an <u>additional table</u>, NewReadArticles, in the create_lab3.sql file that has the same attributes (and Primary Key and Foreign Key constraint) as the ReadArticles table. We'll say more about NewReadArticles below.

Under Resources→Lab3, we've provided a load script named lab3_load_data that loads tuples into the tables of the schema. **You <u>must</u> run both create_lab3.sql and lab3_load_data.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 NewReadArticles, there might already be a tuple in the ReadArticles table that has the same Primary Key (that is, the same value for subscriberPhone, editionDate, and articleNum). If there **isn't** a tuple in ReadArticles with the same Primary Key, then this is the first time that this article has been read, and a new tuple who should be inserted into ReadArticles. If there already **is** a tuple in ReadArticles with that Primary Key, then this article has been read for additional minutes. So here are the effects that your transaction should have:

- If there isn't already a tuple in the ReadArticles table which has those subscriberPhone, editionDate, and articleNum values, then you should insert a tuple into the ReadArticles table corresponding to all the attribute values in that NewReadArticles tuple.

- If there already is a tuple in the ReadArticles table which has those subscriberPhone, editionDate, and articleNum values, then you should update the tuple in ReadArticles that has those subscriberPhone, editionDate, and articleNum values, adding the readInterval in the NewReadArticles tuple to the readInterval in the existing ReadArticles tuples.

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 *create_lab3.sql* script, and load the data using the script *load_lab3.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 create_lab3.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 Lab3 schema.

The load data that you've been given should not cause any errors when you add these constraints. 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 subscriber in the Subscriptions table must appear as a subscriber in the Subscriber table, that means that the subscriberPhone attribute of the Subscriptions table is a Foreign Key referring to the Primary Key of the Subscribers table (which is also subscriberPhone).

- Any subscriber that's in a Subscriptions row must appear as a subscriber in the Subscribers table. If a tuple in the Subscribers table is deleted, and there are Subscriptions tuples which correspond to that subscriber, then that Subscribers tuple deletion should be rejected. If the Primary Key (subscriberPhone) of a Subscribers tuple is updated, then that update should also be applied to all subscriptions that correspond to that subscriber.

- Each subscription kind (subscriptionMode, subscriptionInterval)that's in a Subscriptions row must appear as a (subscriptionMode, subscriptionInterval) in the SubscriptionKinds table. If a tuple in the SubscriptionKinds table is deleted, then subscriptionMode and subscriptionInterval for all subscriptions which correspond to that subscription kind should be set to NULL. If the Primary Key (subscriptionMode, subscriptionInterval) of a SubscriptionKinds tuple is updated, then the same update should be applied to the (subscriptionMode, subscriptionInterval) attributes of the subscriptions which correspond to that subscription kind.

- Each (subscriberPhone, subscriptionStartDate) that appears in the Holds table must appear in the Subscriptions table as a Primary Key (subscriberPhone, subscriptionStartDate). If a tuple in the Subscriptions table is deleted, and there are Holds tuples that correspond to that subscriber, then that Subscribers tuple deletion should be rejected . If the Primary Key (subscriberPhone, subscriptionStartDate) of a Subscriptions tuple is updated, and there are Holds tuples that correspond to that Subscriptions, then that Subscriptions tuple update should also 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 all 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 SubscriptionKinds, rate must be greater than zero. This constraint should be named positiveRate.

2. In Holds, subscriptionStartDate must be less than or equal to holdStartDate, and holdStartDate must be less than or equal to holdEndDate. This constraint should be named okayDatesForHolds.

3. In Subscribers, if subscriberName is NULL then, subscriberAddress must also be NULL. This constraint should be named ifNameNullThenAddressNull.

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 the values TRUE and UNKNOWN are okay 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 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). 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 3 + 6 = 9 unit tests in the order specified above in the file *unittests.sql*.

2.6 Working with a View

Important: Before starting Section 2.6, recreate the Lab3 schema once again using the *create_lab3.sql* script, and load the data using the script *load_lab3.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.

2.6.1 Create a view

We'll say that an edition is a "2021 edition" if its editionDate value has year 2021. If you have a value that's a date (such as editionDate), then EXTRACT(YEAR FROM editionDate) gives its year. articleAuthor is an attribute in Articles, specifying the name of the author of the article.

Some authors are prolific, meaning that they wrote many articles, which appeared in many different editions. We'll say that an author is "prolific in 2021" if that author wrote at least 3 articles, and the articles they wrote appeared in at least 2 different 2021 editions.

Create a view called **ProlificIn2021View** which for each author gives the number of articles they wrote, and the number of different 2021 editions in which their articles appeared. The attributes in your view should be called articleAuthor, articleCount2021, and differentEditionCount2021. But only include an author in the **ProlificIn2021View** view if their articleCount2021 value is at least 3, and their differentEditionCount2021 is at least 2.

Save the script for creating that view in a file called *createview.sql*

2.6.2 Query a View

For this part of Lab3, you'll write a script called *queryview.sql* that contains a query which we'll informally refer to as the "Better2022Authors" query. (You don't actually name that query, or create a view for it.) Better2022Authors uses **ProlificIn2021View** and (possibly) some other tables. In addition to writing the Better2022Authors query, you must also include some comments in your queryview.sql script; we'll describe those necessary comments below.

We'll say that an author is "improving in 2022" if they wrote more articles in 2022 than they did in 2021. (You should know how to figure out how many articles an author wrote in 2022.) Write and run a SQL query which finds just the authors who appear in the **ProlificIn2021View,** and who also had more articles in 2022 than in 2021. (Don't to check on their number of 2022 editions.) The attributes in your result should be authorName, articleCount2021, and articleCount2022, where articleCount2021is the articleCount2021 value for that author that's in **ProlificIn2021View**, and articleCount2022 is the number of articles which that author wrote in 2022.

Then write the results of the "Better2022Authors" 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 only the tuples that have the following Primary Keys from the Articles table:

- The Articles tuple whose Primary Key is ('2021-06-13', 10).
- The Articles tuple whose Primary Key is ('2021-06-13', 1).

Run the "Better2022Authors" 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 *queryview.sql* containing your query on the **ProlificIn2021View** view. In that file you should include:

- your Better2022Authors query,
- a comment with the output of the query on the load data before the deletions,
- the SQL statements that delete the tuples indicated above,
- your Better2022 Authors query (a second time),
- and a second comment with the second output of the same query after the deletions.

It probably was a lot easier to write the Better2022Authors query using the **ProlificIn2021View** view than it would have been if you didn't have that view!

P.S. Yes, you may create additional views, if you want. If you do, be sure to include them in your createview.sql script. The queryview.sql script should not include any view creation.

2.7 Create an Index

Indexes are data structures used by the database to improve query performance. Locating the tuples in the Articles table for a particular articleAuthor and editionDate might be slow if the database system has to search the entire Articles table (if the number of Articles was very large). To speed up that search, create an index named ArticleSearch over the articleAuthor and editionDate columns (in that order) of the Articles 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 find all the articles that were written by a particular author, or to determine how many articles were by a particular author appeared before a particular editionDate.

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/14/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 createview.sql queryview.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 createview.sql queryview.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 createview.sql queryview.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 Tuesday, November 21. Late submissions will not be accepted, and there will be no make-up Lab assignments.