1 Preliminaries

In this lab, you will work with a Nile Book Club 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 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've also provided a `lab3_data_loading.sql` script that will load data into your tables. You'll need to run that before executing Lab3. The command to execute a script is: \i <filename>

You will be required to combine new data (as explained below) into one of the tables. You will also 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 during the weeks before Sunday, November 18. (You have an extra week to do this Lab because of the Midterm, which was on Wednesday, October 31.) But note that Monday, November 12 is a holiday, Veterans Day, so there won't be a class or Lab Section on that day.

2. Description

2.1 Tables with Primary Keys for Lab3

The primary key for each table is underlined.

Authors(authorID, authorName, address, numBooksWritten, mostRecentPubDate)

Books(bookID, authorID, bookName, publisherID, pubDate, price, category, lastOrderDate, totalOrdered)

Publishers(publisherID, publisherName, address)

Members(memberID, memberName, joinDate, renewalDate, isCurrentMember)

Orders(memberID, bookID, orderDate, quantity)

Reviews(reviewerID, bookID, reviewDate, reviewStars)

NewMemberships(memberID, memberName, renewalDate)

In the lab3_create.sql file that we've provided under Resources→Lab3, the first 6 tables are the same as they were in our Lab2 solution, including NULL and UNIQUE constraints. Note that there is an additional table, NewMemberships that has some of the same attributes as Members. As the table name suggests, each of its tuples records a Nile Book Club membership, which may be either for a new member, or a renewal of an existing member. We'll say more about NewMemberships below.

As in previous assignments, to avoid mentioning the schema every time you refer to the tables, you can make **Lab3** the default schema in the search path by issuing the following command:

ALTER ROLE your_user_id SET SEARCH_PATH TO Lab3;

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

Under Resources→Lab3, you've also been 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 in it in a Serializable transaction) that will do the following. For each "new member" tuple in NewMemberships, there might already be a tuple in Members that has the same primary key, memberID. If there **isn't** a tuple with that memberID, then this is a new member of Niles Books. If there already **is** a tuple with that memberID, then this is a renewal of member's membership. So here are the actions that you should take.

a) If there **isn't** already a tuple in Members that has the same primary key, then insert a tuple into the Members table corresponding to that NewMemberships tuple. Use memberID and memberName and renewalDate, as provided in the NewMemberships tuple. Set joinDate to be CURRENT_DATE, and isCurrentMember to be TRUE. (See https://www.postgresql.org/docs/10/static/functions-datetime.html#FUNCTIONS-DATETIME-CURRENT for information on CURRENT_DATE.)

b) If there already **is** a tuple in Members that has the same primary key, then update Members based on that NewMemberships tuple. Don't change memberID or joinDate for that member, but update MemberName and renewalDate based on the values of those attributes in the NewMemberships tuple. (The member may have changed their name.) Also, set isCurrentMember to be TRUE.

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). A helpful hint is provided in the initial Lab3 announcement posted on Piazza.

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. The default for referential integrity should be used in all cases. The data that you're provided with should not cause any errors. (There are many other referential integrity constraints that would exist for this schema—you might want to think about those--but please just add the constraints listed below.)

a) The authorID field in Books should reference the authorID primary key Authors.

b) The bookID field in Orders should reference the bookID primary key in Books.

c) The reviewerID field in Reviews should reference the memberID primary key in Members.

Write commands to add foreign key constraints in the same order that the foreign keys are described above. Save your commands to the file *foreign.sql*

2.4 Add General Constraints

General constraints for Lab3 are:

1. In Orders, quantity must be positive. Please give a name to this positive quantity constraint when you create it. We recommend that you use the name positive_quantity, but you may use another name. The other general constraints don't need names.

2. In Books, lastOrderDate must be greater than or equal to pubDate.

3. In Members, if joinDate is NULL then isCurrentMember must also be NULL.

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 write just a few for the common cases, but there are many more possible tests we could write.

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

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

Also, for each of the 3 general constraints, write <u>2</u> 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, <u>in the order given above</u>, in the file *unittests.sql*.

2.6 Working with views

2.6.1 Create two views

Let's call a book that has at least one review a Reviewed Book. Create a view named GreatPublishers A publisher who is in Publishers is in the GreatPublishers view if it published at least 2 Reviewed Books, and every review of a book that it published has at least 3 stars. The attributes of GreatPublishers should be publisherID and the number of Reviewed Books that the publisher published. In your result, the second attribute should be called numReviewedPublished.

Create a second view named BadBookTotals. A book in Books has a bad book total if that book's totalOrdered (which is an attribute of Books) is not equal to the sum of the quantity values for the Orders of that book. (quantity is an attribute of Orders.) The attributes of BadBookTotals should be bookID, totalOrdered and badQuantitySum for that book; badQuantitySum should be the sum of the quantity values for the Orders of that book.

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

2.6.2 Query views

Write a query over the GreatPublishers and BadBookTotals views to answer the following "Great Publishers with Bad Book Totals" question. (You may also have to use some tables to do this, but be sure to use these views.)

> Some publishers who are GreatPublishers published books that have BadBookTotals. For each publisher that has at least one book that appears in BadBookTotals, output that publisher's publisherID, the number of books they published (numReviewedPublished from GreatPublishers), and the number of books with BadBookTotals that they published. The third attribute in your result should be called numBad.

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 that query in a comment.

Next, write commands that delete just the tuples with the following primary keys from the Orders table:

(8844, 'jgzhwq')
(2161, 'rrrrrr')

Run the "Great Publishers with Bad Book Totals" 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 two 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 will not be penalized if you do).

2.7 Create an index

Indexes are data structures used by the database to improve query performance. Locating all the Orders of a particular book by a particular member may be slow if the database system has to search the entire Orders table. To speed up that search, create an index named LookUpOrders over the bookID and memberID columns (in that order) of the Orders table. Save the command in the file *createindex.sql*.

Note that you can run the same SQL statements whether or not this index exists; having indexes just changes the performance of SQL statements.

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/10/static/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 wouldn'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 18. Late submissions will not be accepted, and there will be no make-up Lab assignments.