Lab Assignment 3   CSE 180 – Winter 2023   Due: 11:59pm Tuesday February 28

1   Preliminaries

In this lab, you will work with a Highway Info 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, February 28. The due date of Lab3 is 3 weeks (plus a couple of days) 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.

Highways(highwayNum, length, speedLimit)

Exits(highwayNum, exitNum, description, mileMarker, exitCity, exitState, isExitOpen)

Interchanges(highwayNum1, exitNum1, highwayNum2, exitNum2)

Cameras(cameraID, highwayNum, mileMarker, isCameraWorking)

Owners(ownerState, ownerLicenseID, name, address, startDate, expirationDate)

Vehicles(vehicleState, vehicleLicensePlate, ownerState, ownerLicenseID, year, color)

Photos(cameraID, vehicleState, vehicleLicensePlate, photoTimestamp)

PhotoChanges(cameraID, vehicleLicensePlate, photoTimestamp)

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; the NULL and UNIQUE constraints from Lab2 are **not** included. Moreover, create_lab3.sql is missing the Foreign Key Constraints on Vehicles (which says that a vehicle's owner must be an owner in Owners), and is missing both of the Foreign Key Constraints on Photos 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. create_lab3.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, PhotoChanges, in the create_lab3.sql file that has some (but not all) of the attributes that are in the Photos table. We'll say more about PhotoChanges below.

Under Resources→Lab3, we've provided a load script named lab3_load_data that loads tuples into the tables of the schema. **You must 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.sq*l (which should have multiple SQL statements that are in a <u>Serializable transaction</u>) that will do the following. For each tuple in PhotoChanges, there might already be a tuple in the Photos table that has the same Primary Key (that is, the same value for both cameraID and photoTimestamp). If there **isn't** a tuple in Photos with the same Primary Key, then this is a new member who should be inserted into Photos. If there already **is** a tuple in Photos with that Primary Key, then this is an update of information about that photo. So here are the effects that your transaction should have:

- If there <u>isn't already a tuple in the Photos table </u>which has that cameraID and photoTimestamp, then you should insert a tuple into the Photos table corresponding to all the attribute values in that PhotoChanges tuple. Your insert should also vehicleState to NULL.

- If there <u>already is a tuple in the Photos table</u> which has that cameraID and photoTimestamp, then update the tuple in Photos that has that cameraID and photoTimestamp. Update vehicleLicensePlate for that existing Photos tuple to equal the corresponding attribute value in the PhotoChanges tuple. Also, change the value of vehicleState to 'CA'.

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

<u>**Important**</u>: 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. <u>Just add the constraints listed below, exactly as described</u>, even if you think that additional Referential Integrity constraints should exist. Note that (for example) when we say that every camera (cameraID) in the Photos table must appear in the Cameras table, that means that the cameraID attribute of the Photos table is a Foreign Key referring to the Primary Key of the Cameras table (which also is cameraID).

- Each owner (ownerState, ownerLicenseID) that appears in the Vehicles table as the owner of a vehicles must appear in the Owner table as a Primary Key (ownerState, ownerLicenseID). If a tuple in the Owners table is deleted, and there are Vehicles tuples that correspond to that owner, then that Owners tuple deletion should <u>be rejected </u>. If the Primary Key (ownerState, ownerLicenseID) of an Owners tuple is updated, then all vehicles in the Vehicles tables corresponding to that owner should also be updated to the new (ownerState, ownerLicenseID) value .

- Each vehicle (vehicleState, vehicleLicensePlate) that appears in the Photos table as the vehicle in the photograph must also appear in the Vehicles table as a Primary Key (vehicleState, vehicleLicensePlate). If a tuple in the Vehicles table is deleted, then all Photos tuples corresponding to that vehicle should also be deleted. If the Primary Key (vehicleState, vehicleLicensePlate) of a Vehicles tuple is updated, then all tuples in Photos that correspond to that vehicle should have both their vehicleState and vehicleLicensePlate values set to NULL.

- Each camera (cameraID) that appears in the Photos table as the camera which took the photograph must also appear in the Cameras table as a Primary Key (cameraID). If a tuple in the Cameras table is deleted, then all Photos tuples that that camera should have their cameraID value set to NULL. If the Primary Key (cameraID) of a Cameras tuple is updated, and there are photos that correspond to that camera, then the update of the Cameras tuple should be <u>rejected</u>.

Write commands to add foreign key constraints in the same order that the foreign keys are described above. Your foreign key constraints <u>should all have names</u>, 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 Exits, mileMarker must be greater than or equal to zero. This constraint should be named mileMarkerNonNegative.

2. In Interchanges, highwayNum1 must be less than highwayNum2. Please give a name to this constraint when you create it. This constraint should be named interchangeHighways.

3. In Owners, if startDate is NULL then expirationDate must also be NULL. This constraint should be named startNullExpirationNull.

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 <u>one</u> 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 <u>2</u> 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 <u>in the order specified above</u> 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

You might expect that when one exit has a greater number than another exit on the same highway, that its mileMarker would also be greater. But we don't have a constraint that guarantees that constraint will hold.

We'll say that an exit is "misdirected" if there is a second exit on the same highway which has a greater exitNumber but whose mileMarker is less than its mileMarker That second exit proves that the first exit is misdirected. There could be multiple second exits on the same highway which have a greater exitNumber but whose mileMarker is less than its mileMarker. For example, exit 3 on a highway might have mileMarker 52, while exit 6 has mileMarker 41 and exit 8 has mileMarker 50, so both exit 6 and exit 8 on that same highway prove that exit 3 is misdirected.

Create a view called **MisdirectedExitView** which for each exit gives the number of exits on the same highway which prove that it is misdirected The attributes in your view should appear as highwayNum, exitNum, mileMarker and misdirectedProverCount. But only include an exit in the MisdirectedExitView view if its misdirectedProverCount value 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 "MisdirectedHighwayQuery" query. (You don't actually name the query.) MisdirectedHighwayQuery uses **MisdirectedExitView** and (possibly) some other tables. In addition to writing the MisdirectedHighwayQuery query, you must also include some comments in the queryview.sql script; we'll describe those necessary comments below.

An exit is open if isExitOpen is TRUE for that exit. For each highway <u>whose highwayNum appears in at least one exit in MisdirectedExitView</u>, we want to count the number of open exits which appear for that highway in MisdirectedExitView.

Write and run a SQL query which finds the number of open exits for each highwayNum that appears in MisdirectedExitView. The attributes in your result should be highwayNum, length and openMisdirectedExitCount, where length is the length of the highway, and openMisdirectedExitCount is the number of open exits for that highway in <u>MisdirectedExitView.</u>

Then write the results of the "MisdirectedHighwayQuery" 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 Exits table:

- The Exits tuple whose Primary Key is (17, 34).

- The Exits tuple whose Primary Key is (280, 95).

Run the "MisdirectedHighwayQuery" 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 views. In that file you must also include:

- a comment with the output of the query on the load 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 *queryview.sql* file (but you won't be penalized if you do).

It probably was a lot easier to write this query using the view than it would have been if you hadn't had it!

2.7 Create an Index

Indexes are data structures used by the database to improve query performance. Locating the tuples in the Vehicles table for a particular vehicleState, color and year might be slow if the database system has to search the entire Vehicles table (if the number of Vehicles was very large). To speed up that search, create an index named VehicleQuest over the vehicleState, color and year columns (in that order) of the Vehicles 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 the vehicles from a particular state which have a particular color, or determine if there any vehicles from a particular state that have a particular color and a particular 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/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, February 28. Late submissions will not be accepted, and there will be no make-up Lab assignments.