Lab Assignment 2 CSE 180 – Winter 2025 Due: Tuesday, Feb 4, 11:59pm

1. Goal

The goal of the second assignment is to create a PostgreSQL data schema with 7 tables that are very similar to the tables that you created in Lab1. The tables have the same names, attributes, and data types as the tables of Lab1, and the same Primary Keys and Foreign Keys, but they have some additional UNIQUE constraints and restrictions on NULL that are described below.

After you create the data schema with the 7 tables, you must write five SQL statements that use those tables. We have also provided you with data that you can load into your tables so that you can test the results of your queries. Testing can prove that a query is wrong but not that it is right, so be careful. We will not give you the results of these queries on the load data; you should be able to figure out the results on that data yourselves. You can also test your queries on your own data. In the "real world", you have to make and check your own tests.

Lab2 is due in two weeks, so you will have an opportunity to discuss the assignment during the Discussion Section in the first week of the assignment and to discuss issues you have had in writing a solution to the assignment during the Discussion Section of the second week. Instructions for submitting the assignment appear at the end of this document.

2 Lab2 Description

2.1 Create PostgreSQL Schema Lab2

You will create a Lab2 schema to set apart the database tables created in this lab from the tables you will create in the future, as well as from tables (and other objects) in the default (public) schema. We'll refer to the database throughout the Lab Assignments, but your schemas will be called Lab1, Lab2, etc. In PostgreSQL, a database can have more than one schema; see here for more details on PostgreSQL schemas. You create the Lab2 schema as follows:

```
CREATE SCHEMA Lab2;
```

Now that you have created the schema, you want to set Lab2 to be your default schema when you use psql. If you do not set Lab2 as the default schema, then you will have to qualify your table names with the schema name (e.g., Lab2.Agencies). To set the default schema, you modify your search path. (For more details, see here.)

```
ALTER ROLE username SET SEARCH_PATH to Lab2;
```

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.)

You do not have to include the CREATE SCHEMA or ALTER ROLE statements in your solution. You can have them as comments in your sql file.

2.2 Create tables

Create tables in schema `Lab2` for the following entities: `Agencies`, `Routes`, `Stations`, `Transfers`, `VehicleKinds`, `Vehicles`, and `VehicleServices`.

The attributes for these 7 tables are the same as for the tables of Lab1. Moreover, the data types for the attributes in these tables are the same as the ones specified for the tables of Lab1, and the Primary Keys

and other Foreign Keys are also the same. However, the tables must have additional constraints, which are described in the next section.

2.2.1 Constraints

The following attributes cannot be NULL. All other attributes can be NULL … but remember that attributes in Primary Keys also cannot be NULL, even though you don't specify NOT NULL for them.

The following attributes cannot be NULL:

- In `Agencies`: `agencyCity`
- In `Routes`: `routeName`
- In `Stations`: `stationName`

Also, the following must be unique for the specified table. That is, there cannot be identical rows in that table that have exactly the same (non-NULL) values for all of those attributes (composite unique constraint).

- In `Agencies`: `agencyCity` and `agencyState` (composite unique constraint).
- In `Routes`: `routeName`.
- In `Stations`: `stationName`.
- In `Vehicles`: `vehicleID`.

For example:

- The first constraint puts a unique constraint on `agencyCity` and `agencyState` ensures that no two rows in `Agencies` can have the same city and state values unless one or both are NULL. This prevents duplicate entries for agencies based in the same location.
- A `NOT NULL` constraint on attributes like `vehicleID` in `Vehicles` ensures that every vehicle has a unique identifier. Without this constraint, data integrity issues could arise due to missing or duplicate entries.

You will write a CREATE TABLE command for each of the 7 tables that has these additional constraints. Save the commands in the file *create_lab2.sql*

SQL Queries

Below are English descriptions of the five SQL queries that you need to write for this assignment, which you will include in files queryX.sql, where X is the number of the query, e.g., your SQL statement for Query 1 will be in the file query1.sql, and so forth. Follow the directions as given. You will lose points if you give extra tuples or attributes in your results, if you give attributes with the wrong names or in the wrong order, or if you have missing or wrong results. You will also lose points if your queries are unnecessarily complex, even if they are correct. Grading is based on the correctness of queries on all data, not just the load data that we have provided.

Remember the Referential Integrity constraints from Lab1, which you should retain for Lab2. For example, if a station (identified by `stationID`) appears in a `Transfers` tuple, then there must be a corresponding tuple in `Stations` for that station (i.e., with that `stationID` value). Similarly, if a vehicle (identified by `vehicleID` and `agencyID`) appears in a `VehicleServices` tuple, then there must be a tuple in `Vehicles` for that vehicle. These constraints ensure that all references between tables are valid and enforce the integrity of the database.

Attributes should have their original names in the results of your queries, unless an alias is requested. And if a query asks that several attributes appear in the result, the first attribute mentioned should appear first, the second attribute mentioned should appear second, etc.

2.3 Query 1

A vehicle is considered "in service" if its `inService` attribute is TRUE. A vehicle is "out of service" if its `inService` attribute is FALSE. If `inService` is NULL, the service status is unknown.

Find all vehicles manufactured before 2010, that are out of service and have been used on a route that has a length greater than 10 miles. The attributes in your result should be the `vehicleID`, `yearBuilt`, and `routeID` of the vehicle, which should appear as `vehicleID`, `manufactureYear`, and `routeID`, respectively. Newer vehicles should appear before older vehicles in your result. If multiple vehicles have the same year of manufacture, order them by `vehicleID` in ascending order.

No duplicates should appear in your result.

2.4 Query 2

Find all the `vehicleID` values for vehicles whose `agencyID` operates at least one route with a `routeName` ending in the string ' Express' (case-sensitive, with a blank before 'Express').

Only include vehicles where the `yearBuilt` is not NULL. The attributes in your result should be `vehicleID` and `agencyID`. No duplicates should appear in your result.

2.5 Query 3

Find the `stationName` of the stations for which all of the following hold:

1. The `stationName` ends with "Station" (case-sensitive, with a blank before "Station").
2. There is at least one transfer at the station between two routes, where both routes are operated by the same agency and have a `length` greater than 10 miles.
3. There is no route operating at the station with an average `ridership` exceeding 2000.

No duplicates should appear in your result.

2.6 Query 4

Find the `agencyID` and `routeID` of routes for which there is more than one vehicle operating with the same non-NULL `yearBuilt`. The attributes in your result should be `agencyID`, `routeID`, and `yearBuilt`, which should appear in your result as `theAgency`, `theRoute`, and `theYear`.

No duplicates should appear in your result.

2.7 Query 5

Find all the routes that have at least one vehicle operating on them, and all the vehicles operating on the route belong to the same agency. The attributes in your result should be the `routeID` of the route and the `agencyID` of the agency to which all vehicles on the route belong.

No duplicates should appear in your result.

Testing

Since databases can be configured in many different ways, we'll be using the provided Docker container to ensure consistency in grading. While you may test on any configuration you wish, **the graders will be testing your code on the Docker container**.

You should already have Docker and PostgreSQL installed. If not, follow these instructions.

To test your code on the Docker container, you will first need to start the Docker daemon (see the installation doc), navigate to `cse180-compose.yml`, and run

```
docker compose -f cse180-compose.yml up
```

The Docker container should have started. Now, open a second terminal window and run

```
docker exec psql -h localhost -U cse180 cse180    or
docker exec -it container-psql psql -U cse180 -d cse180
```

Now you should have accessed the database on the Docker container. This is where you will do your testing. Now, open a third terminal window and navigate to your solution file. Run the following command to transfer files to the Docker container:

```
docker cp <file_name> container-psql:.
```

Your files should now be accessible from within the Docker container.

When you're done, you can exit the database using `\q` or `exit;`, and you can stop the Docker container by navigating back to `cse180-compose.yml` and running

```
docker compose -f cse180-compose.yml down
```

While your solution is still a work in progress, it is a good idea to drop all objects from the database every time you run the script, so you can start fresh. Of course, dropping each object may be tedious, and sometimes there may be a particular order in which objects must be dropped. The following commands (which you should put at the top of *create_lab2.sql*), will drop your Lab2 schema (and all objects within it), and then create the (empty) schema again:

```
DROP SCHEMA Lab2 CASCADE;
CREATE SCHEMA Lab2;
```

Before you submit, login to your database via psql and execute your script. As you've learned already, the command to execute a script is: \i <filename>.

We have shared a load script named *load_lab2.sql* that loads data into the 8 tables of the database. You can execute that script with the command:

\i load_lab2.sql

You can test your 5 queries using that data, but you will have to figure out on your own whether your query results are correct. We won't provide answers, and students should not share answers with other students. Also, your queries must be correct on any database instance, not just on the data that we provide. You may want to test your SQL statements on your own data as well.

Submitting

1. You can always get rid of duplicates by using DISTINCT in your SELECT. In CSE 180, we deduct points if students use DISTINCT and it wasn't necessary because even without DISTINCT, there couldn't be duplicates. We will also deduct if you were told not to eliminate duplicates, but you did.

2. Save your scripts for table creations and query statements as *create_lab2.sql* and *query1.sql* through *query5.sql* You may add informative comments inside your scripts if you want (the server interprets lines that start with two hyphens as comment lines).

3. Zip the file(s) to a single file with name Lab2_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 Lab2 should be named Lab2_1234567.zip

   To generate the zip file you can use the Unix command:

   zip Lab2_1234567 create_lab2.sql query1.sql query2.sql query3.sql query4.sql query5.sql

   (Of course, you use your own student ID, not 1234567.)

4. Lab2 is due by 11:59pm on Tuesday, February 4. **Late submissions will not be accepted; Canvas won't take them, nor will we.** Always check to make sure that your submission is on Canvas, and that you've submitted the correct file.