Lab Assignment 1 **CSE 180 – Winter 2025** **Due: 11:59pm Monday Jan 20**

1. Goal

The goal of the first assignment is to create a PostgreSQL data schema with 7 tables. That is all that is required in this assignment; don't do anything that's not required. The other Lab Assignments are much more difficult. In your Lab Sections, you may be given information about how to load data into a table and issue simple SQL queries, because that's fun, but loading data and issuing queries are **not** required in this assignment. (That will show up in the Lab2 assignment.)

2. Lab1 Description

2.1 Create PostgreSQL Schema Lab1

First, you will create a Lab1 schema to set apart the database tables created in this lab from tables you will create in future labs, as well as from tables (and other objects) in the default (public) schema. Note that the meaning of schema here is specific to PostgreSQL, and distinct from the general meaning of schema. See here for more details on PostgreSQL schemas. You create the Lab1 schema using the following command:

```
CREATE SCHEMA Lab1;
```

[PostgreSQL makes all identifiers lowercase, unless you put them in quotation marks, e.g.,, "Lab1". But in CSE 180, you don't have to bother using quotation marks for identifiers. We use capitals in assignments for readability, but it's okay (and equivalent) if you use lab1 as the schema name.]

Now that you have created the schema, you want to make Lab1 be the default schema when you use psql. If you do not set Lab1 as the default schema, then you will have to qualify your table names with the schema name (e.g., by writing Lab1.customer, rather than just customer). To set the default schema, you modify your search path as follows. (For more details, see here.)

```
ALTER ROLE cse180 SET SEARCH_PATH to Lab1;
```

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, and then are surprised that their tables aren't in the expected schema.) To see your current SEARCH_PATH, enter:

```
SHOW SEARCH_PATH;
```

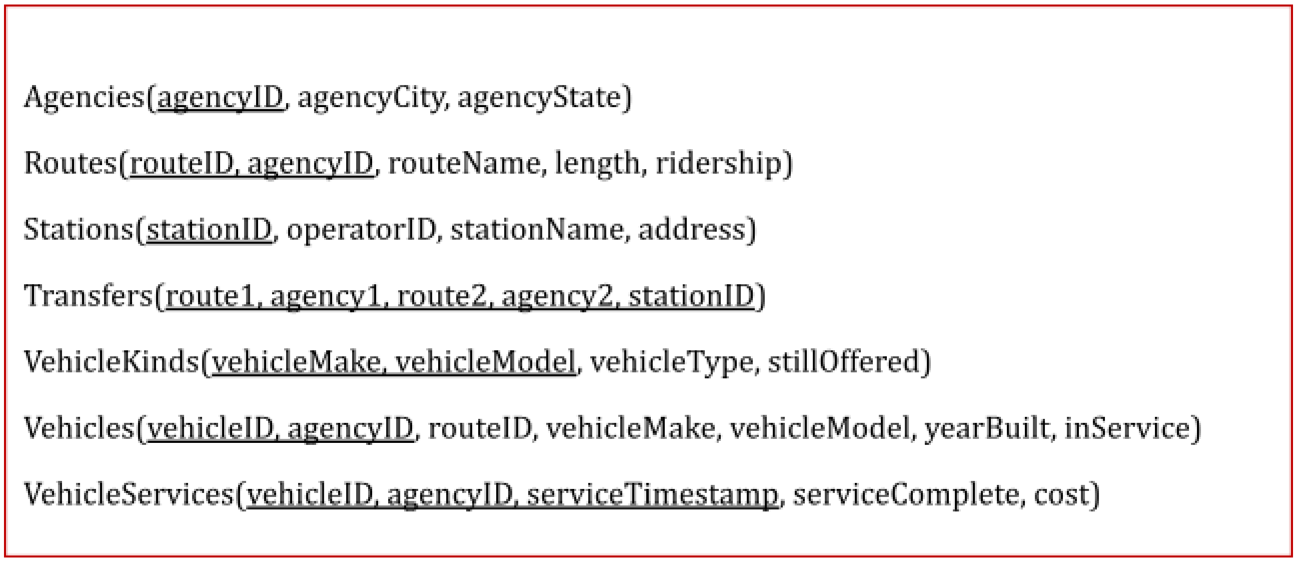
If you forget to do the ALTER ROLE for Lab1 to modify your search path then your schema will be the default schema, PUBLIC. Note that every student has their own

database, so your PUBLIC schema won't be the same as any other student's PUBLIC schema, and the same is true for any other schema, such as Lab1.

2.2 Tables

You'll be creating tables for a very simplified version of a database schema for a government body that oversees the operations of multiple Public Transportation agencies (which we'll call the Public Transport database). There are tables for Agencies, Routes, Stations, Transfers, VehicleKinds, Vehicles, and VehicleServices. The data types and Referential Integrity for the attributes in these 7 Public Transport tables are described in the next section. No, this schema doesn't provide everything that a real-world Public Transport database would include, but it's a decent start.

Important: To receive full credit, you must use the attribute names as given, and the attributes must be in the order given. Also, the data types and referential integrity must match the specifications given in the next section. Follow directions; do not do more than you're asked to do in this assignment.



Agencies(agencyID, agencyCity, agencyState)

Routes(routeID, agencyID, routeName, length, ridership)

Stations(stationID, operatorID, stationName, address)

Transfers(route1, agency1, route2, agency2, stationID)

VehicleKinds(vehicleMake, vehicleModel, vehicleType, stillOffered)

Vehicles(vehicleID, agencyID, routeID, vehicleMake, vehicleModel, yearBuilt, inService)

VehicleServices(vehicleID, agencyID, serviceTimestamp, serviceComplete, cost)

The underlined attribute (or attributes) identifies the Primary Key of each table. A table can only have one Primary Key, but that Primary Key may involve multiple attributes.

- An agency in Agencies describes a Public Transport agency. It gives the agency's ID and the city it's headquarters are in.
- A route in Routes specifies the route's identifier, the agency that operates the route, the route's name (usually a street or neighborhood the route serves), and how many people per day, on average, used this route during the 2019 fiscal year.
    - Any agencyID that appears in a route row must appear as the agencyID in the Agencies table.
- A station in Stations describes an official place where transit vehicles can board or alight passengers. It specifies the ID of the station, the agency that operates the station, and the station's name and address.
    - Any operatorID that appears in a station row must appear as the agencyID in the Agencies table.

- A transfer in Transfers describes a station where two routes come together and allow passengers to transfer between them. A transfer specifies the two routes that come together and the station where the transfer takes place.
    - Any (route1, agency1) that appears in a transfer row must appear as the (routeID, agencyID) in the Routes table.
    - Any (route2, agency2) that appears in a transfer row must appear as the (routeID, agencyID) in the Routes table.
    - Any stationID that appears in a transfer row must appear as the stationID in the Stations table.

- A kind of vehicle in VehicleKinds describes the specific vehicles that agencies may use. A kind of vehicle specifies the vehicle's make and model, that kind of vehicle's type (which can be 'MB', 'TB', 'LR', or 'HR', for motorbus, trolleybus, light rail, or heavy rail), and whether this kind of vehicle is still offered.

- A vehicle in Vehicles specifies the agency that operates it, the route the vehicle operates on, the kind of vehicle it is, when the vehicle was built, and whether the vehicle is currently operating.
    - Any `(vehicleMake, vehicleModel)` that appears in a vehicle row must appear as the `(vehicleMake, vehicleModel)` in the VehicleKinds table.

- A vehicle service in `VehicleServices` specifies the vehicle being serviced, when the service was performed, whether or not service has been completed, and how much the service cost.
    - Any `(vehicleID, agencyID)` that appears in a vehicle service row must appear as the `(vehicleID, agencyID)` in the `Vehicles` table.

In this assignment, you'll just have to create tables with the correct table names, attributes, data types, Primary Keys and Referential Integrity. "Must appear as" means that there's a Referential Integrity requirement. **Be sure not to forget Primary Keys and Referential Integrity when you do Lab1!**

2.2.1 Data types

Sometimes an attribute (such as state) appears in more than one table. Attributes that have the same attribute name might not have the same data type in all tables, but in our schema, they do.

- `agencyID`, `operatorID`, `agency1`, `agency2`, `routeID`, `route1`, and `route2` should use *character of variable length*, with maximum length 5.
- `agencyCity`, `routeName`, `stationName`, `vehicleMake`, and `vehicleModel` should be *character of variable length,* with maximum length 20.
- `address` should use *character of variable length*, with maximum length 60.
- `agencyState` and `vehicleType`, should use *character with fixed length 2*.
- `ridership`, `stationID`, `vehicleID`, and `yearBuilt` should use *integer*.
- `length` should use *numeric,* with at most 4 decimal digits to the left of the decimal point and 1 decimal digit after it.
- `cost` should use *numeric*, with at most 6 decimal digits to the left of the decimal point and 2 decimal digits after it.
- `serviceTimestamp` should use type *timestamp*.
- `stillOffered`, `inService`, and `serviceComplete` should use type *boolean*.

You must write a CREATE TABLE statement for each of the 7 tables in Section 2.2. Write the statements in the same order that the tables are listed above. **Use the data types, Primary Keys and Referential Integrity described above.** You will lose credit if you do anything beyond that, even if you think that it's sensible. Save your statements in the file create_lab1.sql

PostgreSQL maps all SQL identifiers (e.g., table names and attributes) to lowercase. That's okay in your CSE 180 assignments. You won't lose points for Lab1 because Persons appears in the database as persons, and Actors appears as actors. It is possible to specify specific case choices for an identifier by putting that identifier inside double-quote symbols, e.g., as "Agencies". But then every time you refer to that identifier, you'll have to use the double-quotes. "AGENCIES" is not the same identifier as "Agencies", and neither of those is the same as Agencies (written without double-quotes), which PostgreSQL maps to agencies. We will use capitalization for readability, but we won't bother using double-quotes in our own Lectures, Lab Assignments and Exams.

3. Testing

3.1 Using Docker

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
```

3.2 Testing your solution

While you're working on your solution, it is a good idea to drop all objects from the schema every time you run the create_lab1.sql script, so you can start fresh. Dropping each object in a schema may be tedious, and sometimes there may be a particular order in which objects must be dropped. (Why?) The following command, which you should put at the top of your create_lab1.sql script, will drop your Lab1 schema (and all the objects within it), and then create the (empty) schema again:

```
DROP SCHEMA Lab1 CASCADE;
CREATE SCHEMA Lab1;
```

The first statement will result in an error if the Lab1 Schema doesn't exist, but execution of your script will continue after that.

Before you submit your Lab1 solution, login to your database via psql and execute your create_lab1.sql script. As you'll learn in Lab Sections, the command to execute a script is: \i <filename> Verify that every table has been created by using the command: \d When you execute \d, the tables may be displayed in any order, not necessarily in the order in which you created them.

To verify that the attributes of each table are in the correct order, and that each attribute is assigned its correct data type use the following command: \d <table>.

We've supplied some load data that you can use to test your solution in the file load_lab1.sql. After you've created your tables, using the command : \i create_lab1.sql, you can load that data in psql by executing the command: \i load_lab1.sql. (Why will loading the data twice always result in errors?) If your solution fails on the load data, then it's likely that your solution has an error. But although testing can demonstrate that a program is buggy, testing cannot prove that a program is correct.

4. Submitting

1. Save your script as create_lab1.sql You may add informative comments to your scripts if you want. Put any other information for the Graders in a separate README file that you may submit.

2. Zip the file(s) to a single file with name Lab1_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 Lab1 should be named Lab1_1234567.zip

   If you have a README file (which is not required), you can use the Unix command:

   ```
   zip Lab1_1234567 create_lab1.sql README
   ```

   If you don't have a README file, to create the zip file you can use the Unix command:

   ```
   zip Lab1_1234567 create_lab1.sql
   ```

   (Of course, you should use **your own student ID**, not 1234567.) Submit a zip file, even if you only have one file.

Submit the zip file on Canvas under Assignment Lab1. Please be sure that you have access to Canvas for CSE 180. Registered students should automatically have access; students who are not registered in CSE 180 will not be able to submit solutions. You can replace your original solution, if you like, up to the Lab1 deadline. (Canvas will give the new file a slightly different name, but that's okay.) No students will be admitted to CSE 180 after the Lab1 due date.

Lab1 is due by 11:59pm on Monday, January 20. **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.