CSC343 Winter 2023
Assignment #2: Embedded SQL
Due Thursday March 9 by 5:00pm

In this assignment you will be working with a SQL schema for a database to manage waste collection for the City of Toronto, similar to the schema for Assignment 1. Do **not** assume any of the constraints from A1 apply here, unless they are explictly enforced in this handout or in our **waste_wrangler_schema.sql** file.

We change our point of view from Assignment 1, where we recorded only events in the past. In Assignment 2 we also schedule events in the future, for example trips that are to take place in the future.

Learning Goals

This assignment aims to help you learn to:

- Read a SQL schema
- Apply the techniques from class for writing queries
- Devise techniques using Python's psycopg2 module to augment basic SQL techniques

Submission Instructions

You will submit a Python file **a2.py**. You may work on the assignment on your own, or in a group of 2, and submit a single assignment for the entire group on MarkUs. Partners do not have to be in the same section, but must be in CSC343 at St. George this semester. You must establish your group well before the due date. Instructions on how to form a group can be found here. **No changes may be made to groups after Monday, March 6.** This means you should have created your group or declared that you will work alone in MarkUs by then!

You declare a group by submitting an empty, or partial, file, and this should be done well before the due date. You may always replace such a file with a better version, until the due date.

Check that you have submitted the correct version of your file by downloading it from MarkUs; new files will not be accepted after the due date. If you are working with a partner, only one of you needs to upload the file for your group.

SQL schema

You can copy our schema and some sample data to the folder on `dbsrv1.teach.cs.toronto.edu` that you are working in as follows:

```
cp ~csc343h/winter/pub/waste_wrangler_schema.sql .
cp ~csc343h/winter/pub/waste_wrangler_data.sql .
cp ~csc343h/winter/pub/qualifications.txt .
```

We provide you with some sample data, but you are responsible for populating the tables with an instance that more fully tests your **a2.py** solution.

Methods

In **a2.py** you will find headers for the methods described below. Your task is to fill in **only the body** of these definitions so that they correctly satisfy their specifications. If you instantiate a cursor within a method, be sure to close it within the body of your method.

We have provided **connect** and **disconnect** for you. Please **do not** call these within the methods you write. You may assume that they have been called before your method (connect), and after your method (disconnect). You'll find an example in **a2.py**'s main block.

Please follow good programming practices, including using **private** helper methods where appropriate. Be sure your helper methods have docstrings.

schedule_trip: Schedule a truck and two employees to a given route at a given date and time to pick up an unknown volume of waste. Both employees and the truck must not be scheduled for a different trip from 30 minutes before the expected start of the trip until 30 minutes after the expected end of this trip. The end of the trip can be computed by assuming that the truck travels an average of 5 kph. At least one of the employees must be able to drive the truck. Prefer employees with earlier hire dates and break ties by ascending eID.

schedule_trips: Schedule the given truck for trips on the given date using the following approach:

1. Find routes with no trips scheduled on the given date that have a waste type that the given truck is able to carry. Schedule trips on these routes in ascending order of their IDs.
2. Starting from 8 a.m., find the earliest available pair of drivers of whom at least one can drive the given truck and both are available for the day. Break ties by choosing lower `eIDs`.
3. Continue scheduling trips, being sure to leave 30 minutes between trips using the assumption that the truck will travel an average of 5 kph. The last trip must have ended by 4:00pm.

update_technicians: A training centre sends a text file with the following format:

```
<firstName> <surName>
<truckType>
```

... where `<firstName>`, `<surName>` and `<truckType>` don't contain any whitespace characters, but occasionally `<firstName>` is preceded is preceded by a title (e.g. `Mr., Ms, Prof.`), followed by white space. These titles should not be kept in our database. There is no blank line at the end of the file.

The meaning is that the named employee is now qualified to work on `truckType` as a technician.

Use the given **qualifications.txt** file to update the database to reflect the new qualifications.

workmate_sphere: Good and bad habits get passed from worker to worker when they work together. For employee **e1**, any employee who has been on a trip with **e1** is part of **e1**'s workmate sphere. Recursively, any employee who has been on a trip with an employee in **e1**'s workmate sphere is also in **e1**'s workmate sphere.

schedule_maintenance: For each truck with most recent maintenance over 90 days ago before the given date, schedule maintenance with a technician qualified to work on that truck. Choose the first day after the given date when there is a qualified technician available (not scheduled to maintain another truck

that day) and the truck is not scheduled for a trip/maintenance. If there is more than one technician available on a given day, choose the one with the lowest `eID`.

reroute_waste: The given facility had an emergency shutdown. Reroute the trips scheduled on the given date to this Facility to another facility that takes the same type of waste. Don't worry about too many trips arriving at the same time, each facility has ample receiving facility.

Assume this happens before any of the trips have reached the given facility.

Marking

We will be automarking your submissions. Even if you don't completely solve all methods, be sure they run without error and in a reasonable amount of time.

- Do **not** share your code outside your MarkUs group in any form: verbal, written, or electronic. Don't look at any other being's code, except our starter code. Academic offense investigations are messy, painful for both students and instructors, and drag on for ages.
- Do **not** use another platform to share code, even with your partner. If your code migrates from, say, github to any student not in your group, we will hold you responsible for any resulting academic offense. You can easily share code with your partner using ssh, or downloading the current version from MarkUs. **Be careful to make sure you are not sharing code in a publicly visible repo!**
- We will test your code on **dbsrv1.teach.cs.toronto.edu**. Your code must run on our machine, although you may develop it elsewhere. Be sure that you are comfortable using ssh or another utility to move files to and from **dbsrv1.teach.cs.toronto.edu**.
- We have include some **assert** statements at the end of `a2.py` in the `test_preliminary` function. These are meant to assure you that you have started off on the right track, and you will earn 20% of the assignment grade for passing these.
- Be sure to use clear comments documenting your code, and modular design.
- Your methods will be marked independently, so you do not need to complete every one to be able to earn marks (or part marks) on the others. **It is important to make sure your file runs without error though, so that we can import it into our autotester.**