CSC343 Winter 2018
Assignment 3
JDBC & Database Design

Due: Thursday 29 March at 5:00 pm

1 JDBC

The first part of the assignment is about embedded SQL. We use the same database as A2 for all questions. Imagine a political science analytics app that is used by researchers and analysts. The app has a graphical user-interface, written in Java, but ultimately it has to connect to the database where the core data is stored. Some of the features will be implemented by Java methods that are merely a wrapper around a SQL query, allowing input to come from gestures the user makes on the app, like button clicks, and output to go to the screen via the graphical user-interface. Other app features will include computation that can't be done, or can't be done conveniently, in SQL.

For part one of this assignment, you will not build a user-interface, but will write several methods that the app would need. It would need many more, but we'll restrict ourselves to just enough to give you practise with JDBC and to demonstrate the need to get Java involved, not only because it can provide a nicer user-interface than postgreSQL, but because of the expressive power of Java.

General requirements

- You may not use standard input or output. Doing so even once will result in the autotester terminating, causing you to receive a **zero** for this part.
- You will be writing a method called `connectDb()` to connect to the database. When it calls the `getConnection()` method, it must use the database URL, username, and password that were passed as parameters to `connectDb()`; these values must not be "hard-coded" in the method. Our autotester will use the `connectDb()` and `disconnectDB()` methods to connect to the database with our own credentials.
- You should **not** call `connectDb()` and `disconnectDB()` in the other methods we ask you to implement; you can assume that they will be called before and after, respectively, any other method calls.
- All of your code must be written in `Assignment3.java`. This is the only file you may submit for this part.
- You may not change the interface for any of the methods we've asked you to implement. However, you are welcome to write helper methods to maintain good code quality.
- As you saw in lecture, to run your code, you will need to include the JDBC driver in your class path. You may wish to review the related JDBC Exercise posted on the course website.
- JDBCSubmission is an abstract class that is provided to you. You should not make any changes in this file and you do not need to submit this file.

Your task

Open the starter code in `Assignment3.java`, and complete the following methods.

1. `connectDB`: Connect to a database with the supplied credentials.
2. `disconnectDB`: Disconnect from the database.
3. `presidentSequence`: A method that, given a country, returns the list of Presidents in that country, in descending order of date of occupying the office, and the name of the party to which the president belonged.
4. `findSimilarParties`: A method that, given a party, returns other parties that have similar descriptions in the database.

You will have to decide how much the database will do and how much you'll do in Java. At one extreme, you could use the database for very little other than storage: for each table, you could write a simple query to dump its contents into a data structure in Java and then do all the real work in Java. This is a bad idea. The DBMS was designed to be extremely good at operating on tables! You should use SQL to do as much as it can do for you, and part of your mark for Part 2 will be based on whether or not you do so.

We don't want you to spend a lot of time learning Java for this assignment, so feel free to ask lots of Java-specific questions as they come up.

Similar parties

This method identifies similar parties using the descriptions that are available about them in this parlgov database. In other words, two parties are potentially similar if the textual information available about them is similar enough.

"Jaccard similarity" provides a simple but effective similarity score (between 0 and 1) for two given sets of strings. It is defined as the size of the intersection divided by the size of the union of two sets. For instance, the Jaccard similarity of `S1 = {Ontario, Toronto}` and `S2 = {Alberta, Ontario, Manitoba}` is 0.25. The helper method `similarity` computes the Jaccard similarity of two sets of strings. Your job is to use the `similarity` method to find the parties whose `description` attributes have similarity above a given threshold.

2 Database Design

The second part of the assignment is about Database Design

2.1 Introduction

In class, we are in the middle of learning about functional dependencies and how they are used to design relational schemas in a principled fashion. After that, we will learn how to use Entity-Relationship diagrams to model a domain and come up with a draft schema which can be normalized according to those principles. By the end of term you will be ready to put all of this together, but in the meanwhile, it is instructive to go through the process of designing a schema informally.

2.1.1 About Car Rentals

A car rental service [1] provides cars (without a chauffeur service) on rent to customers for a period of time. Cars are kept at rental stations which are situated at fixed locations.

[1] See for example: `https://www.avis.ca/`

2.1.2 System Functional Requirements

Customers provide their name, age and email to register into the system. Once registered, a customer can make reservations for a from/to date for a car from a particular location. A reservation status can be confirmed (before journey), ongoing (during journey), completed (after journey completion) or cancelled. Each reservation involves exactly one car. Every car corresponds to a single rental station (The car has to be picked up and dropped off at this particular station only.) It is assumed that the customer making the reservation will be driving the car. However, a customer can choose to add more customers for a reservation if multiple people will be driving the car.

Every rental station has a code, name, location and the associated cars. Location consists of the area code, city and descriptive street address. All cars have a licence plate number, model number, model name, type and capacity information. For each model number, there can be multiple number of cars, all having the same name, type and capacity.

A customer can change a confirmed reservation (only once), in which case the system considers the existing reservation to be cancelled and a new reservation is confirmed. The system preserves the old reservation details associated with this new reservation.

No pricing information needs to be tackled in the design.

Define a schema

Your first task is to construct a relational schema for our domain, expressed in DDL. Write your schema in a file called `schema.ddl`.

As you know, there are many possible schemas that satisfy these properties. We aren't following a formal design process for this part, so instead follow as many of these general principles as you can when choosing among options:

- If a constraint given above in the domain description can be expressed without assertions or triggers, it should be enforced by your relational schema.
- Avoid redundancy.
- Avoid designing your schema in such a way that there are attributes that can be null.
- Wherever an attribute *cannot* be null (according to the domain description), add a `NOT NULL` constraint.

You may find there is tension between some of these principles. Where that occurs, use your judgment to make a tradeoff.

To facilitate repeated importing of the schema as you correct and revise it, begin your DDL file with our standard three lines:

```
drop schema if exists carschema cascade;
create schema carschema;
set search_path to carschema;
```

Document your choices

At the top of your DDL file, include a comment that answers these questions:

1. What constraints from the domain could not be enforced?
2. What constraints that could have been enforced were not enforced? Why not?

Instance and Queries

Once you have defined your schema, create a file called `data.sql` that inserts data into your database that represents the small dataset defined informally in file `Car-data.txt`. You may find it instructive to consider this data as you are working on the design.

Then write queries to do the following:

1. Q1: Find the top 2 customers with the highest reservation cancellation ratio.
2. Q2: Find the top 2 customers who rent cars with driver(s) most frequently.
3. Q3: Find the most frequently rented car model in Toronto, where the reservation started and was fully completed in the year 2017.
4. Q4: Find the list of all customers younger than 30 years old who changed at least 2 reservations in the past 18 months. Note: You're required to return a list of customer IDs.

We will not be autotesting your queries, so you have latitude regarding details like attribute types and output format. Make good choices.

Write your queries in files called `q1.sql` through `q4.sql`. Download file `runner.txt`, which has commands to import each query one at a time. Once all your queries are working, start postgreSQL, import `runner.txt`, and cut and paste your entire interaction with the postgreSQL shell into a plain text file called `demo.txt`.

3 Deliverables

3.1 Part A

This part will be submitted through markus

1. connectDB 2
2. disconnectDB 2
3. presidentSequence 10
4. findSimilarparties 10
5. comments, coding style 6

3.2 Part B

Hand in plain text files `schema.ddl`, `data.sql`, `q1.sql` through `q4.sql`, and `demo.txt`. These must be plain text files, and you must include the demo file, or you will get zero for this part of the assignment. If you are unsure about this, please talk to an instructor during office hours.

1. schema.ddl 25
2. data.sql 10
3. queries 5 each
4. demo.txt 15

4 Submission Instructions

Will be submitted/marked through markus