University of Toronto
CSC343, Summer 2019

Assignment 2

Due: July 15th, before 10:00pm

Learning Goals

By the end of this assignment you should be able to:

- interpret specifications accurately
- read and interpret a novel schema
- write complex queries in SQL
- design datasets to test a SQL query thoroughly
- quickly find and understand needed information in the postgreSQL documentation
- embed SQL in a high-level language using JDBC
- recognize limits to the expressive power of standard SQL

General Instructions

Please read this assignment thoroughly before you proceed. Failure to follow instructions can affect your grade.

We strongly encourage you to do all your development for this assignment on the CS Teaching Labs, either in the lab rooms or via a remote connection. Your code must run on these machines in order to earn credit.

Download the starter files from the course webpage: You can copy the files on the Teaching Labs by using the following command.

```
> cp ~csc343h/summer/pub/assignments/a2/data.zip .
```

You are allowed, and in fact encouraged, to work with a partner for this assignment. You must declare your team (whether it is a team of one or of two students) and hand in your work electronically using MarkUs.

Once you have submitted your files, be sure to check that you have submitted the correct version; new or missing files will not be accepted after the due date, unless your group has grace tokens remaining.

Schema

In this assignment, we will perform some analytics on real data from one hundred years of elections. The data comes from ParlGov[1], a database for political science. It contains information on political parties, elections and cabinets for most democracies that are part of the EU (European Union) or the OECD (Organization for Economic Co-operation and Development)

Download the schema and a sample dataset. Your code for this assignment must work on *any* database instance (including ones with empty tables) that satisfies the schema, so make sure you understand the schema well.

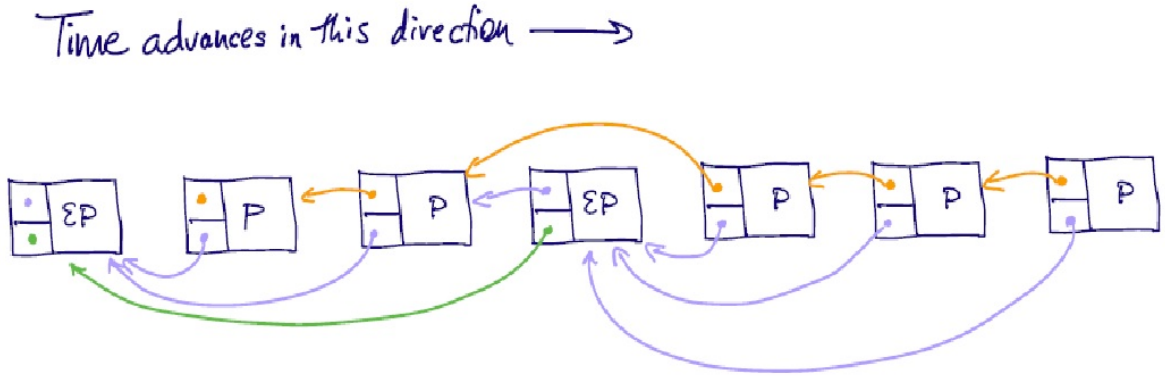
Warmup: Getting to know the schema

The schema for this assignment is one of the most complex we've ever used. (On the other hand, we have given you fewer queries than usual.) It is critical that you invest time early on getting to understand the structure and meaning of the tables. Here are a few things to keep in mind:

- The schema can record elections of two kinds: A **parliamentary election** is an election within a country to choose a national government. A **European parliament election** (or EP election) is an election, held across all European Union countries, to choose national representatives for the European parliament.
- If you are not familiar with any of the political terminology used in the schema, refer to the comments for an explanation.
- The same schema is being used to represent election data from many countries, with numerous variations in their style of their form of governance, so some of the terminology may be used in an unfamiliar way. Take it to be as defined in the schema, even if this doesn't quite match what you think a "cabinet" is in Canada, for instance.

Two tables embed structures within them that are particularly interesting.

- Each row of the `election` table records information about a single election. The row includes the ID of the previous Parliamentary election and the previous EP election (using attributes `previous_parliament_election_id` and `previous_parliament_election_id`). These two attributes essentially create two independent linked lists within the table. However, it's more complicated than that, because even a Parliamentary election has a reference to the previous EP election, and even an EP election has a reference to the previous Parliamentary election. This diagram may help you understand the structure embedded in the `election` table:



The orange arrows show the linked list of parliamentary elections going back through time, and the green arrow shows the linked list of EP elections going back through time. But we also store the references across election types that are also stored. When you look at the whole structure, you can see that it more than just two linked lists.

[1] http://www.parlgov.org

- The `election_result` table records political alliances that form between different parties in an election. To represent that a set of parties formed an alliance in an election, the database singles one party out (we'll call it the **head** of the alliance, and it is arbitrary which party is the head) and has all the others refer to it in their `alliance_id` attribute. (This is a little surprising, since `alliance_id` sounds like a unique identifier for the alliance rather than a reference to one of the parties in the alliance.) The other parties in the alliance refer to the head party by storing in `alliance_id` the `id` of the election result for the head party. The `alliance_id` value for the head party of the alliance is NULL. For example, if parties A, B, C, and D formed an alliance in election e1, then the table could include these rows:

id	election_id	party_id	alliance_id	seats	votes
id1	**e1**	**A**	**NULL**		
id2	**e1**	**B**	**id1**		
id3	**e1**	**C**	**id1**		
id4	**e1**	**D**	**id1**		

(Or the database could have singled out a different one of these parties to be the head.)

Part 1: SQL Statements

General requirements

In this section, you will write SQL statements to perform queries. Write your SQL statement(s) for each question in separate files. You will submit two files for each question i:

- In `qi.sql` file you will (a) define the table that is required for that query, (b) define the SQL query, and (c) INSERT INTO the table you defined.
- In `qi_order.sql`, you simply order the table you created in `qi.sql` based on ordering criteria defined in the question.

In total, you will submit 10 files: `q1.sql`, `q1_order.sql`, `q2.sql`, `q2_order.sql` ..., `q5.sql`,`q5_order.sql`

For example, suppose we ask you to find the sIDS of all students with a cpga greater than 3 and sort them by cpga. for the following table in q1 then you will take the following steps:

- In q1.sql:
  - `create table q1(sID integer primary key, cgpa integer);`
  - `insert into q1 select sID, cgpa from student where cgpa > 3;`
- And then in q1_order.sql:
  - `select * from q1 order by cgpa;`

You are encouraged to use views to make your queries more readable. However, each file should be entirely self-contained, and not depend on any other files; each will be run separately on a fresh database instance, and so (for example) any views you create in `q1.sql` will not be accessible in `q5.sql`. **Each of your files must begin with the line** `SET search_path TO parlgov;` Failure to do so will cause your query to raise an error, leading you to get a 0 for that question.

The output from your queries must exactly match the specifications in the question, including attribute names, order and type, and the order of the tuples.

We will be testing your code in the **CS Teaching Labs environment** using PostgreSQL. It is your responsibility to make sure your code runs in this environment before the deadline! **Code which works on your machine but not on the CS Teaching Labs will not receive credit**.

The queries

We will define the **winning party** to be the party that won the most votes. It is possible that several parties are tied for the most votes, in which case we say that there are multiple winning parties.

Write SQL queries for each of the following:

1. **Find Winners.** Find parties that have won more than 3 times the average number of winning elections of parties of the same country. Report the country name, party name and its party family name along with the total number of elections they have won. For each party included in the answer, report the id and year of the mostly recently won election.

Attribute	**Description**
countryName	Name of the country
partyName	Name of the party
partyFamily	Name of the family of a party
wonElections	Number of elections the party has won
mostRecentlyWonElectionId	The id of the election that was most recently won by this party
mostRecentlyWonElectionYear	The year of the election that was most recently won by this party
Order by	The name of the country ascending, then the number of won elections ascending, then the name of the party descending.
Everyone?	Include only countries and parties who meet the criteria of this question.
Duplicates?	Countries and party families can be included more than once with different party names.

2. **Do citizens participate more?** The number of eligible voters, votes votes, and valid votes have been recorded for each election. Analysts would like to know if citizens of countries participate enthusiastically in elections. The **participation ratio** of an election is the ratio of votes cast to the number of citizens who are eligible to vote (the "electorate"). Note the participation ratio is a value between zero and one. As part of this question, you will need to compute the participation ratio for each country, each year. If more than one election happens in a year in a country compute the average participation ratio across those elections.

   Write a query to return the countries that had at least one election over the last 15 years, 2001 to 2016 inclusive, and whose average election participation ratios during this period are *monotonically non-decreasing* (meaning that for Year Y and Year W, where at least one election has happened in each of them, if Y $<$ W, then average participation in Year Y is $\leq$ average participation in Year W). For such countries, report the name of the country and the average participation ratio per year for the last 15 years.

Attribute	**Description**
countryName	Name of the country
year	year
participationRatio	The average percentage ratio of citizens who cast vote in this year
Order by	The name of country descending then year descending
Everyone?	Include only countries that meet the criteria of this question.
Duplicates?	No rows if there are no elections for a country in the last 15 years.

3. **Committed Parties.**

   A committed party is the one that has been a member of all cabinets in their country over the past 20 years. For each country, report the name of committed parties, their party families and their "regulation of the economy" value.

Attribute	**Description**
countryName	Name of a country
partyName	Name of a committed party
partyFamily	Name of a committed party's family if exists, otherwise, null.
stateMarket	Regulation of the economy property of the party if exists, otherwise, null.
Order by	countryName ascending, then partyName ascending, then stateMarket descending
Everyone?	Include only countries with committed parties
Duplicates?	There can be no duplicates.

4. **Sequences of Cabinets.** For each country, report the cabinets formed over time.

Attribute	**Description**
countryName	name of a country
cabinetId	the id of a cabinet
startDate	start date of a cabinet
endDate	end date of a cabinet. For the most recent cabinet, report NULL for the endDate.
pmParty	name of the party that fills the position of prime minister. Report NULL if the information is not available.
Order by	country descending then start date asceding
Everyone?	Every country and cabinet should be included.
Duplicates?	No cabinet can be included more than once.

5. **Election Alliances** A political alliance, is an agreement for cooperation between different political parties often for purposes such as winning an election mutually. We assume each alliance is led by a party. Zero, one or more than one alliance might be formed in an election. In the *election_result* table, the row corresponding to the election result of a party that participates in an alliance links to the alliance leader party by recording the election result id of the leader in *alliance_id* attribute. The *alliance_id* of a leader is recorded as NULL. Report the pair of parties that have been allies with each other in at least 30% of elections that have happened in a country.

Attribute	**Description**
countryId	id of a country
alliedPartyId1	id of an allied party
alliedPartyId2	id of an allied party
Order by	countryId descending, then alliedPartyId1 descending, then alliedPartyId2 descending
Everyone?	Every allied pair that satisfy the condition.
Duplicates?	No pair can be included more than once. To avoid symmetric pairs ("pseudo-duplicates") solely include pairs that satisfy alliedPartyId1 <alliedPartyId2.

Part 2: Embedded SQL

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 2 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 `Assignment2.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 `Assignment2.java`, and complete the following methods.

1. `connectDB`: Connect to a database with the supplied credentials.
2. `disconnectDB`: Disconnect from the database.
3. `electionSequence`: A method that, given a country, returns the list of elections in that country, in descending order of years, and the cabinets that have formed after that election and before the next election of the same type.
4. `findSimilarPoliticians`: A method that, given a president, returns other presidents that have similar comments and descriptions in the database. See section Similar Politicians below for details.

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 Politicians

This method identifies similar politicians using the descriptions that are available about them in this parlgov database. In other words, two politicians 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 politians whose `description` attributes have similarity is above a given threshold.

Additional tips

Some of your SQL queries may be very long strings. You should write them on multiple lines for readability, and to keep your code within an 80-character line length. But you can't split a Java string over multiple lines. You'll need to break the string into pieces and use `+` to concatenate them together. Don't forget to put a blank at the end of each piece so that when they are concatenated you will have valid SQL. Example:

```
String sqlText =
   "select travelerId " +
   "from traveler t join Booking b on t.travelerId = b.travelerId " +
   "where city = 'Toronto'";
```

Please refer to the JDBC exercise from class for some common mistakes and the error messages they generate.