University of Toronto
CSC343, Winter 2018

Assignment 2

Due: March 9, 5pm

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

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.

To obtain some example data and the schema we will use for this assignment, you can use the command line on teach.cs.toronto.edu to download the files. For example:

```
> wget http://www.teach.cs.toronto.edu/~csc343h/winter/assignments/a2/data.zip

> wget http://www.teach.cs.toronto.edu/~csc343h/winter/assignments/a2/answerTableDefinitions.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.

We will provide test cases for the queries (meaning example input database along with expected answers). You may use these to debug your queries, but of course your queries must be correct on all possible instances (databases), not just the test cases. You may run a query in Markus over a test case, but to ensure we do not crash the server or deprive other courses of resources, you may do so at most eight times in a day.

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.

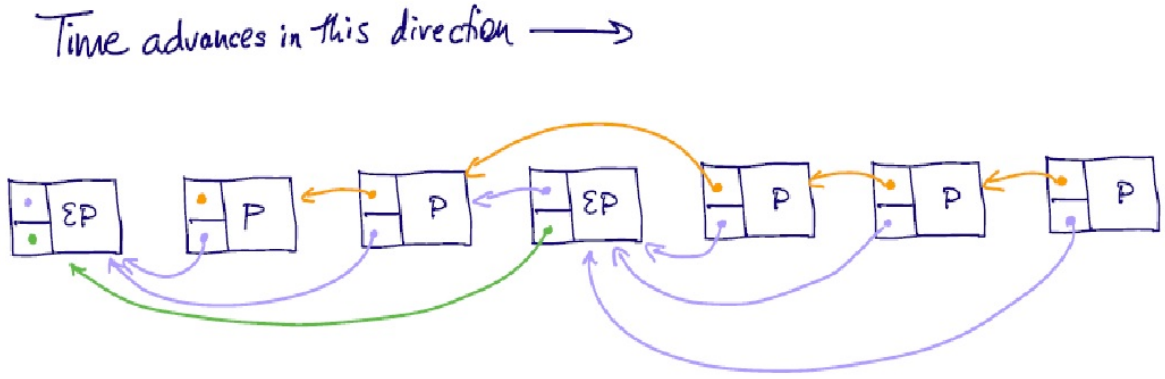
Warmup: Getting to know the schema

A similar schema to the one we are using in this assignment has been used previously in 343, but has been changed for this term, so be sure you are following exactly the specification in this handout. 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 or 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.)

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 14 files: `q1.sql`, `q1_order.sql`, `q2.sql`, `q2_order.sql` ..., `q7.sql`,`q7_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. An example correct solution is the following.

- In q1.sql:
  - `SET search_path TO parlgov;`
  - `drop table if exists q1 cascade;`
  - `create table q1(sID integer, cgpa integer);`
  - `insert into q1 select sID, cgpa from student where cgpa > 3;`
- And then in q1_order.sql:
  - `SET search_path TO parlgov;`
  - `select * from q1 order by cgpa;`

Notice that we have given you the `create table` statements for each of the questions in the answerTablesDefinition.zip file.. You are encouraged to use views to make your queries more readable. However, each file should be entirely self-contained, and may not depend on any other files. Each file 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`. For development and marking, you should use statements like `drop table if exists q1 cascade;` (and similarly for any views or other DDL you create) in your files.

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

IMPORTANT: In all queries other than Q7, we will define the **winning party** to be the party of the cabinet for that election. Use the code below to determine the winning parties for a given election.

```
-- get all of the winning parties based on the cabinet
create view election_winners as
    select election.id as election_id, cabinet_party.party_id
    from election join cabinet
            on election.id = cabinet.election_id
        join cabinet_party
            on cabinet.id = cabinet_party.cabinet_id
    where cabinet_party.pm = true;
```

Listing 1: View containing election id and the winning party of that election

Write SQL queries for each of the following.

1. **Longterm Political Trends**

   Analyze the change in party positions for parliamentary elections in the 20th and 21st centuries (note that the year 2000 is part of the 20th century). Per metric (i.e., left-right, state-market, liberty-authority), report the average position of the winning party (or parties, if an alliance) in all parliamentary elections held in the century per country. For alliance governments, the party position for that alliance should be considered the average across each member; i.e., when averaging positions over the century consider alliances as one entity.

Attribute	**Description**	**Data Type**
century	will be '20' or '21'	varchar
country	name of the country	varchar
left_right	average of the winning parties left_right dimension in the indicated century	real
state_market	average of the winning parties state_market dimension in the indicated century	real
liberty_authority	average of the winning parties liberty_authority dimension in the indicated century	real
Order by	century ascending, then country ascending	
Everyone?	all countries for each of two centuries	
Duplicates?	none	

2. **Electoral Systems & Alliances**

   Analyze the number and size of alliance governments that result from different electoral systems (parliamentary elections only). Report the number of governments that were formed comprised of 1 party (no alliance), 2-3 parties, 4-5 parties, and 6 or more parties.

Attribute	Description	Data Type
country	name of country	varchar
electoral_system	a description of the electoral system used by the country	varchar
single_party	number of governments formed from exactly one party	int
two_to_three	number of governments formed from 2-3 parties	int
four_to_five	number of governments formed from 4-5 parties	int
six_or_more	number of governments formed from 6 or more parties	int
Order by	country ascending	
Everyone?	all countries	
Duplicates?	none	

3. **Dissolution**

   Analyze the number of dissolved parliaments across countries (only parliamentary elections). All countries elect a government every `election_cycle` years. Report the number of elections that occurred "off-cycle" (less than every `election_cycle` years). For each country, report the number of dissolutions (i.e., off-cycle elections) and the number of "on-cycle elections" (which occur `election_cycle` years apart). Consider the very first election recorded to be on-cycle regardless of whether the subsequent election was off-cycle. Also, report the most recent off-cycle and on-cycle elections.

   For this question, just consider the year in which an election is held. So if an election was held 01-01-2000 and the next election was held 12-31-2002 then these elections are two years apart and would be on-cycle if and only if `election_cycle` is two years.

Attribute	Description	Data Type
country	name of the country	varchar
num_dissolutions	number of parliamentary dissolutions which is same as number of off-cycle elections	int
most_recent_dissolution	most recent dissolution	date
num_on_cycle	number of parliamentary elections occurring on cycle	int
most_recent_on_cycle	most recent on-cycle election	date
Order by	country ascending	
Everyone?	all countries	
Duplicates?	none	

4. **Incumbency**

   Analyze the influence of incumbency (by leader and by party) on election results per country. Report, per country, the total number of elections, the number of sequential elections won by the same party, and the number of elected prime ministers who have previously been elected (not necessarily sequentially). Again, assume that the winning party is the one that formed the cabinet for that election.

Attribute	Description	Data Type
country	name of the country	varchar
num_elections	total num of parliamentary elections	int
num_repeat_party	total num elections won by the party who most recently held office	int
num_repeat_pm	total num of elected prime ministers who have already served as prime minister (of the same country)	int
Order by	country ascending	
Everyone?	all countries	
Duplicates?	none	

5. **Close Calls**

   Find the parliamentary elections where the vote difference was **strictly** less than 10% between the winning party or alliance (based on the party in cabinet) and the runner up party or alliance (based on number of votes). In the case of an alliance, compute the number of votes garnered by the winning party (in cabinet), and sum the votes received by any party in an alliance with the winning party.

Attribute	Description	Data Type
electionId	election identifier	int
countryName	name of a country	varchar
winningParty	name of winning party (alliance)	varchar
closeRunnerUp	party (alliance) with votes within 10% of winner	varchar
Order by	electionId ascending, closeRunnerUp	
Everyone?	Only elections with one or more close runner ups	
Duplicates?	An election can have more than one close runner up	

6. **Longest winning streak**

   Find the parties with the longest winning streak in parliamentary elections for each country. Report all if there are ties.

Attribute	Description	Data Type
countryId	id of a country	int
partyName	short name of party	varchar
number	number of consecutive wins	int
Order by	countryId descending, then partyName descending	
Everyone?	Every country	
Duplicates?	There can be multiple parties with the same number of consecutive wins	

7. **Strong Parties**

   A strong party is one that has served (as a winning party or part of a winning alliance) before at least one European Parliamentary election and in **every** period before **every** European Parliamentary election. For example, if the only European Parliamentary elections were on 01-01-2000, 08-01-2004, and 12-31-2008, then a strong party must have won a parliamentary election strictly before 01-01-2000, sometime on or after 01-01-2000 and strictly before 08-01-2004, and sometime on or after 08-01-2004 and strictly before 12-31-2008. Report all strong parties along with the family (if any) of the party.

Attribute	Description	Data Type
partyID	Party id of the strong party	int
partyFamily	Name of the party's family if it exists, otherwise, null	varchar
Order by	partyID	
Everyone?	Include all strong parties	
Duplicates?	There should be no duplicates	