CSC343 Winter 2023
Assignment #3: Design and normalization
Due Thursday April 6 by 5:00pm

learning goals

Once you've finished this assignment you'll be able to:

- evaluate tradeoffs in database design and make reasonable choices between various alternatives
- create an appropriate schema in SQL's Data Definition Language (DDL)
- realize there are some desirable constraints that can't be expressed or enforced in DDL and that the relational model may force awkward design choices
- reason about functional dependencies and apply functional dependency theory to database design

Part 1: informal design

This assignment is live in parallel to lectures and weekly exercises on functional dependencies, using them to design relational schemas in a systematic way, and Entity-Relationship diagrams to come up with draft schema.

In this exercise, though, you will design a database informally, starting from our text description in natural English, plus some prioritized goals for the schema you devise.

our domain

Soon the City of Toronto will be holding a by-election for mayor. You are to design a database to manage some public records about that election (see bold items below). The information is not yet complete, but your draft schema is responsible for only what's mentioned.

election campaigns have candidates, volunteers, staff, and a spending limit. Candidates, volunteers, and staff are uniquely identified by their email, and there is no overlap between these categories.

debates have a time and date, candidates, and a moderator. Different debates will have different collections of candidates, but it is not possible for the same candidate or moderator to be at two or more debates at the same date and time.

donors have addresses, are uniquely identified by their email, donation amounts, which campaign donations are to, whether a donation is individual or from an organization.

workers are either volunteers or staff, are uniquely identified by their email address, and have scheduled blocks of time they undertake to work on campaign activities. Campaign activities could be phone banks or door-to-door canvassing.

We prune some of the real-world detail from our domain for this assignment.

define your schema

Construct a relational schema for our domain in DDL, and store it in a file called **schema.ddl**.

There are many possible schemas that satisfy our informal description. You must, in addition, follow our prioritized principles below, where (1) has the highest priority:

1. every table has a primary key
2. avoid redundancy
3. avoid the possibility of **null** values
4. try to enforce any of the constraints that follow from our informal description **except** if they require assertions or triggers
5. you may enforce additional constraints that make sense to you but aren't part of our domain description

Of course, use **UNIQUE**, foreign keys, and **NOT NULL** appropriately to implement the above priorities. Be sure that any constraints that apply to a table are either part of the table definition or else appear in **schema.ddl** immediately after the table they constrain.

To make your workflow easier, SQL statements in your **schema.ddl** should begin with the standard SQL below, which will make it easier to adjust your schema as you experiment with it.

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

Feel free to create tables, and invent IDs as appropriate.

In a real project you would have an opportunity to get supplemental information from your client on additional questions about this domain. In this case, make reasonable assumptions and make notes on those assumptions. You may need those notes for the next section.

schema documentation

At the top of file **schema.ddl**, before any of the SQL code, include a comment (lines prefaced with `--` characters) that answers:

could not enforce some constraints from the domain description without using assertions or triggers; list these unenforceable constraints

did not enforce some constraints from the description, even though they could have been enforced without assertions or triggers. Explain why not.

extra constraints you added, beyond those in the description, that you enforce

assumptions you made

privileges

We want you to express several SQL queries on your election schema. These queries are designed to be used by accounts with different access to this database. Before each query, set your userid to have minimum privilege on each table in your schema:

```
set search_path to election;
revoke all on <table1>, ..., <tableN> from <userid>;
```

... for each ⟨table⟩. Devise your query, and then restore the minimum privilege back to your user that is necessary to run the query:

```
grant <privilege> on <table> to userid;
```

Save both your query and the `grant...` expressions to a file called **queries.sql**. Here are the queries:

1. List total organizational donations and total individual donations for each campaign.
2. Find those volunteers who offer to work on every campaign in the dataset.
3. Find candidates who are involved in every debate.

deliverables for 'informal design'

Your schema and queries must load / execute in psql without error (although the queries may return an empty relation). In addition, submit a small database instance, **instance.sql** such that each of your queries above return a relation with at least one tuple. Your three files must load into **psql** without error. You must be able to execute the following commands in a **psql** session launched from whatever directory you have **schema.ddl**, **instance.sql**, and **queries.sql** saved in:

```
=> \o a3.output
=> \i schema.ddl
=> \i instance.sql
=> \i queries.sql
```

Your schema will be assessed on how well it represents the election domain while implementing the list of priorities we gave above.

Your queries will be assessed on whether they express the given request, whether you have granted a minimal access to tables, and whether any SQL you invoke is clear and well-commented. Be sure to format your code so that:

- no more than 80 characters per line
- keywords are either all uppercase or all lowercase
- table names are camel case: start each constituent word with a capital letter, e.g. CamelCase
- don't capitalize attribute names
- use line breaks and indentation on the left to make your code more readable

Submit **schema.ddl**, **queries.sql**, **instance.sql**, and **a3.output** to MarkUs.

Part 2: functional dependencies, normal forms, decompositions

Store your answers to the questions below in PDF document **normalization.pdf** that you produce by typing with word processing software (e.g. Word, Google doc, LaTeX). Show the steps in your work, and submit it to MarkUs.

1. Relation R_1 has attributes: $DEFGHIJK$ and functional dependencies S_1:

$$S_1 = \{D \to FG, E \to HK, F \to EIJ, F \to K\}$$

   (a) Which of the dependencies violate BCNF?

   (b) Use the BCNF decomposition algorithm to produce a lossless and redundancy-preventing decomposition of R_1 into a set of relations that are in BCNF. Make it clear to the reader which relations are in the final decomposition. There may well be more than one correct answer, depending on which FD you use to make a choice at each step. List your final relations so that (i) the attributes are in alphabetical order from left to right and (ii) the relations are in alphabetical order from top to bottom.

   (c) Does your solution preserve dependencies? Explain how you know whether or not it does.

   (d) Although a lossless join is guaranteed by using the BCNF algorithm to decompose the original relation, prove this is true using the Chase Test.

2. Relation R_2 has attributes $JKLMNOPQ$ with functional dependencies S_2:

$$S_2 = \{JLM \to N, K \to LM, KN \to JLO, M \to JKO, N \to JL\}$$

   (a) Find a minimal basis for R_2. In your answer, put the FDs in order by making sure that:

      i. attributes on each LHS and RHS are in alphabetical order

      ii. your list of FDs is in alphabetical order by LHS; break ties by alphabetical order on their RHS

   (b) use your minimal basis from the previous part to find **all** keys for R_2.

   (c) use the 3NF synthesis algorithm to produce a lossless and dependency-preserving decomposition of R_2 into a collection of relations that are in 3NF. Be sure to combine all FDs with the same LHS to create a single relation. If you have a relation whose attributes are a subset of another relation, remove the relation with fewer attributes.

We will split the weight of this assignment approximately two-thirds for the first section on informal design and queries, and one-third for the second section on normalization and functional dependencies.

If you work with a partner and decide to divide the topics in some way **be sure** that you understand your partner's work backwards, forwards, and perhaps sideways. The reason for this is that both partners are responsible for the academic integrity of the entire assignment, and both partners should understand topics that may turn up on the final exam.

You can double-check that you have submitted the correct version of your work by downloading it from MarkUs; we will not accept new files after the due date.