Shawn S. Hillyer

Jason Goldfine-Middleton

CS 340 Database Project Documents

2016 U.S. Presidential Nomination Process

# Outline

This project will track information and statistics related to the 2016 United States presidential nomination process. The nomination process for the two main U.S. political parties, the Republican Party and the Democratic Party, consists of a series of statewide (or territory-wide) voting or selection events that allocate delegates for the parties’ respective nomination conventions. Candidates from each party contest each other for as many of their party’s delegates as possible, seeking to gain a majority by the time that all states and territories have held their events. Each state has a pre-allocated number of delegates for each party. The events take a few different forms: primary elections, caucuses, and statewide conventions.

A primary election is a direct vote held by residents of a state, whereby delegates are allocated to each candidate roughly proportional to the number of votes earned. A caucus is a bit more indirect. Voters meet at locations throughout their state and vote openly for a candidate. Their votes are used to proportionally allocate local delegates, who in turn vote to proportionally allocate regional and ultimately statewide delegates. Statewide conventions generally involve no voter participation. Candidates will typically appeal directly to delegates for their support in these conventions.

As the 2016 presidential nomination process is both dynamic and current, we decided that a database project to link parties, their candidates, states, their delegate allocation events, and the results of these events could serve as an interesting and relevant resource for the analysis of an important phenomenon as well as an educational tool to demonstrate the power of relational databases to synthesize real-world data.

# Database Outline in Words

[[ Tell me how the data is supposed to work. This is similar to the description I gave for the question 3 of assignment 1. What constraints should be in place. What tables are related to what other tables. A lot of the grading will be based on if things match this description of your database so make sure it is complete. If you say a constraint exist and you don’t enforce it, that is incorrect. If you enforce a constraint you don’t describe, that is incorrect. ]]

There will be a few simple tables and a couple of more complex ones. We discuss the `state`, `party`, `contest\_type`, `candidate`, `contest`, and `contest\_candidate` tables here. After planning and discussing these tables, we felt that they were sufficient and necessary to provide the types of data our database planned to achieve. Note that I use PK in place of Primary Key and FK in place of Foreign Key when discussing the tables. All “id” PKs are auto\_increment.

## `state` Table

The state entity is simple and yet very important. It stores all of the states or territories in the United States that are relevant to the dataset.

It has only three attributes: id (PK), a name, and an abbreviation. Because our database is strictly for the U.S. election process, our database/business rules are that 2-letter abbreviation and name are both unique to the table. This means there can be only one Oregon and only one state with the abbreviation OR. This follows the real world rules, where every state/territory has a unique 2-letter abbreviation and names are not reused. We will also enforce that the 2-letter abbreviation be exactly two letters, and that none of the attributes shall be NULL in any case.

The `state` entity is related only directly to the `contest` table, which is a table that, in short, describes on specific “voting” event on a certain date for a certain party. Each state has a 1:N relationship with contest because each state can have multiple events (one for each political party, in particular)

## `party` Table

The party entity stores the names of all of the relevant political parties. Traditionally this would be “Democratic” and “Republican”. We just to use a separate table for this data because, by following best practices in normalizing our database, we realized that this data should be its own entity. Further, it would allow for more attributes to be added later if we felt the need, such as a description of the party or other things we hadn’t realized yet.

This entity has an id (PK) and a name attribute. The party name should be unique (as is the PK, which is unique by virtue of being a PK). We enforce that the name of each party be unique, as it would make no sense to have two democrat parties, etc.

The party entity has a 1:N relationship with candidates. That is, each candidate has 1 party, and each party can have N candidates.

## `contest\_type` Table

Contest type is very similar to party. There are different kinds of contests, such as caucuses, statewide conventions and primary elections, so we enforce a unique constraint on the name for each contest and assign an id (PK) for each. Each contest\_type can be associated with many contest records. There is a single relationship between this table and contest which is 1:N, where each contest has one contest\_type, but a contest\_type can be associated with many contest.

## `candidate` Table

This is a table which holds data about each of the candidates. IT has an id (PK), first name (`fname`), last name (`lname`), and a party\_id (FK). None of these are unique identifiers on their own, and we would hope that Americans would not need to choose between two candidates with the same name for the same party, but we can imagine a scenario where this would happen. Therefore, we are not requiring that any of the fields alone or combined as unique composite key (using UNIQUE KEY … syntax) is a good idea.

Candidates each have a party, which is a separate table, so this attribute is a foreign key. We will require that the party\_id attribute not be NULL, as we consider a candidate without a party to be an invalid concept for the purposes of our miniworld. Each candidate has exactly one party, as mentioned earlier, and each party can have multiple candidates, so this is a 1:N relationship.

Candidate is also related to the contest\_candidate table. The contest\_candidate table, as we describe in detail later, basically holds the details of a specific candidate’s received votes at a specific contest. THerefore there is a 1:N relationship with contest\_candidate as each candidate can have multiple records of details (one for each contest), while the contest\_candidate record would be associated with exactly one candidate.

## `contest` Table

As we mentioned earlier, a record in the contest table represent a unique event on a specific date, in a specific state, for a specific party. It also has a specific contest\_type. So it has a handful of foreign keys. Let’s step through all the attributes in a list form to keep this more organized:

* id: PK, auto-incremented as usual.
* contest\_date. The day the event happens. This is not necessarily unique, as it’s possible that events are happening on the same day in various states. We named it contest\_date so that it does not get confused with the ‘date’ datatype. We DO allow this to be NULL as it would allow us to populate events with information prior to the date being set.
* state\_id: FK. A contest must have a valid foreign key reference. There is a 1:N relationship from states to contests (each contest has one state; each state can have N contests).
* party\_id: FK. A contest must have a valid foreign key reference to a party id. This is alos 1:N, as each contest has one contest\_type but contest\_types can be associated with N contest records.
* contest\_type\_id: FK. Again, this must not be NULL and references a valid record in contest\_type. The relationship is also 1:N because each contest can have only 1 contest\_Type, but contest\_type can be associated with many contests.

## `contest\_candidate` Table

This table has a few attributes: an id (PK), a vote\_count, and a delegate\_count are attributes that describe the voting results for a specific candidate. The candidate\_id and contest\_id are foreign keys. Here are the rules we used in list form, as this table is also slightly more complex.

* candidate\_id: FK, Not NULL. It must reference some valid id from candidate.
* contest\_id: FK, not null. It must reference some valid id from contest.
* vote\_count: Defaults to NULL. This is because a candidate might only have delegate votes or no votes at all (yet). By setting it to NULL, rather than 0, we can find contest\_candidate results that are NULL and ignore them (for calculating things like averages), select them (for finding candidates that are missing information), or other uses.
* Delegate\_count: Same rules as vote\_count for the same reasons.

## Other constraints / business rules

We set ON DELETE RESTRICT to prevent the deletion from a parent record. This removes the risk of creating records with invalid foreign key references.

# ER Diagram of Database

[[ append the ER Diagram at end maybe, or separate file, depending on submission requirements ]]

# Database Schema

[[ append the Schema at end maybe, or separate file, depending on submission requirements ]]

# Table Creation Queries

[[ Grab these from whatever file we use -- probably the polirace.sql file in project/sql ]]

[[ SAMPLE: ]]

-- --------------------------------------------------------------------------------------------------

-- Create the database

-- --------------------------------------------------------------------------------------------------

CREATE DATABASE IF NOT EXISTS poli\_race\_2016;

-- --------------------------------------------------------------------------------------------------

-- Create the tables for the database

-- --------------------------------------------------------------------------------------------------

-- See list of states at wikipeida.org, list\_of\_states\_and\_territories\_of\_the\_United\_States

CREATE TABLE IF NOT EXISTS state (

id INT(11) NOT NULL AUTO\_INCREMENT PRIMARY KEY, -- PK

name VARCHAR(255) NOT NULL,

abbreviation char(2) NOT NULL,

UNIQUE(name, abbreviation)

) ENGINE=InnoDB; -- Check syntax on this

# General Use Queries (30%)

*[[ “I want to see all of the queries that will be used to select, update, add or delete data. Because many of these will be based on user input, use square brackets to act as place holders for variables that will be user provided. For example, if I were going to query based on employee salaries, I might have a query like this:”*

*SELECT salary FROM employee WHERE salary > [salaryInput ]; “]]*

# HTML (10%) and PHP (10%)

Our site is hosted at [[ URL HERE ]]. [[ Do we include the source as part of turnin or just URL? Probably an attachment at most ]]