Shawn S. Hillyer
Jason Goldfine-Middleton
CS 340 Database Project Documentation
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

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. Lowercase letters are automatically converted to uppercase for this attribute. Finally, the state name must be in the range of [3..255] characters.

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, and that it must be in the range of [3..255] characters.

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. We enforce that the contest type’s name must be in the range of [3..255] characters.

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.

We require that the first and last names be within the same [3..255] characters range we've used thus far.

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.

- delegates: This attribute is just the number of delegates (maximum) in a specific contest. Defaults to null since a contest may not be associated with delegates whatsoever.

`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. We enforce that the votes and delegates values simply be non-negative values but do not validate them further. In production code / real-world project this would be something we would examine in more detail.

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.

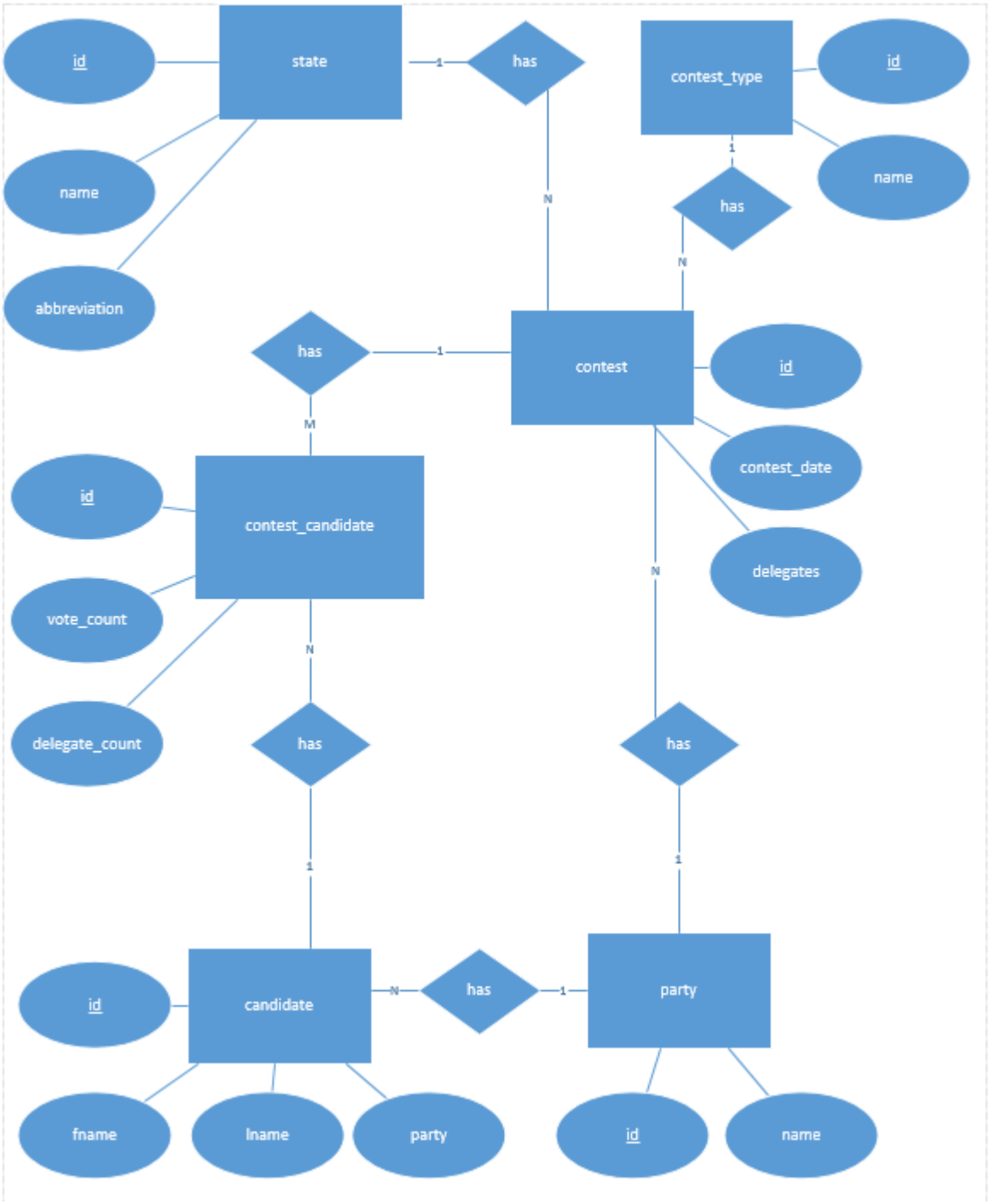
Other constraints / business rules

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. There is no DELETE functionality built in to the website so this was a decision based on a good back-end design that will be safer for a back-end admin.

We prepopulated the database with some real, and some silly, test data. See our github repo (link at end of document) file called populate_data.sql for the bulk insert statements we ran – none of these were executed as part of the web site so we exclude that source from this document. We also left in our own insertions from various tests for flavor.

See the following pages for our ER diagram, Schema, and SQL queries.

ER Diagram of Database



Database Schema

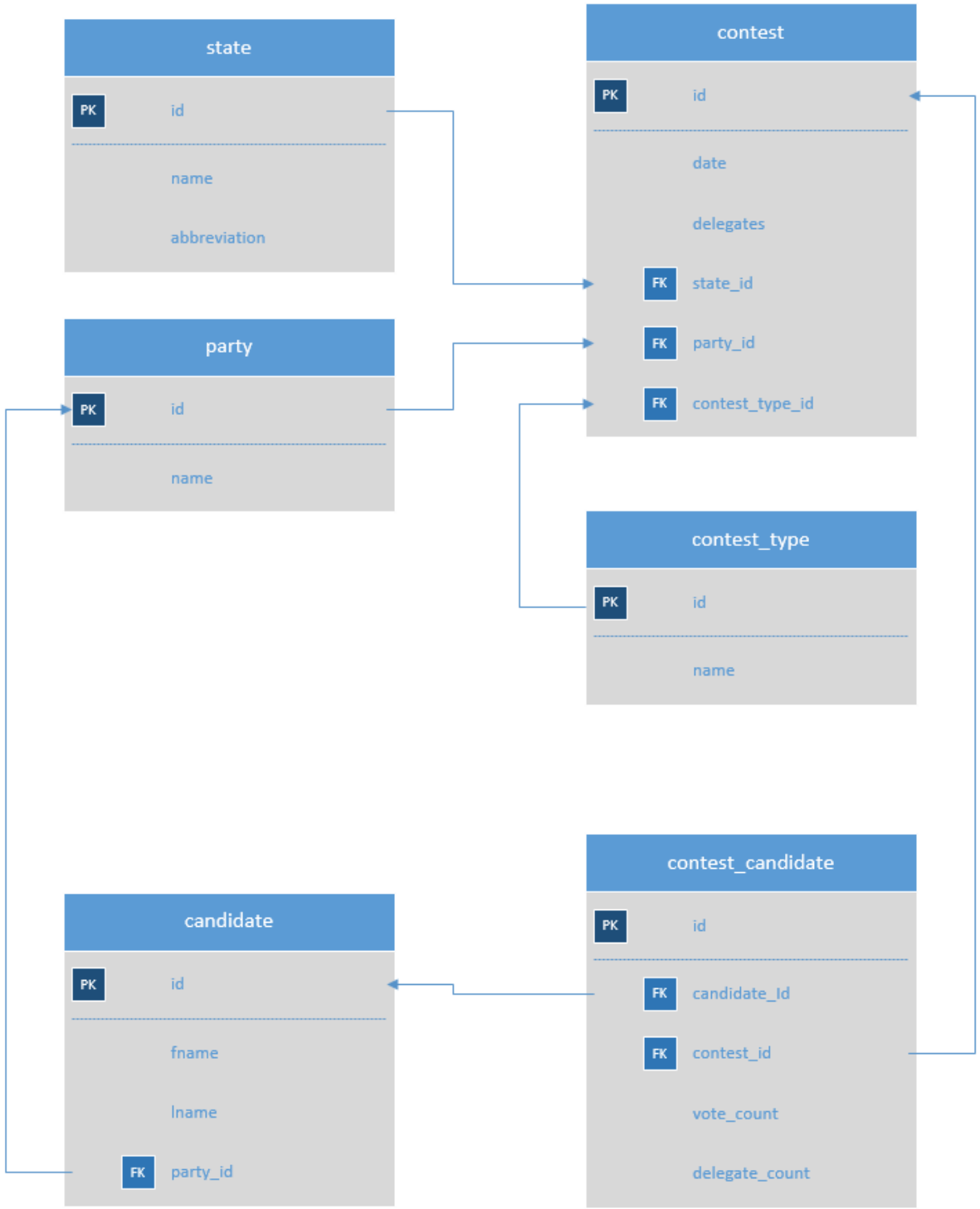


Table Creation Queries

```
-- ------------------------------------------------------------------------------------------------
--  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,
  `name` varchar(255) NOT NULL,
  `abbreviation` char(2) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `name` (`name`,`abbreviation`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=61 ;

-- e.g. democrat, republican, etc.
-- allows us to even put in candidates that are not demo/repub which just aren't associated with any
contests
CREATE TABLE IF NOT EXISTS `party` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(255) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `name` (`name`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=8 ;

-- Political candidates names and parties
CREATE TABLE IF NOT EXISTS `candidate` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `fname` varchar(255) NOT NULL,
  `lname` varchar(255) NOT NULL,
  `party_id` int(11) NOT NULL,
  PRIMARY KEY (`id`),
  KEY `fk_party_id` (`party_id`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=25 ;

-- The type of contest that's held for a specific political party in a specific state. e.g. caucus |
primary
CREATE TABLE IF NOT EXISTS `contest_type` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(255) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `name` (`name`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=4 ;

CREATE TABLE IF NOT EXISTS `contest` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `contest_date` date DEFAULT NULL,
  `state_id` int(11) NOT NULL,
  `party_id` int(11) NOT NULL,
  `contest_type_id` int(11) NOT NULL,
  `delegates` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `fk_state_id` (`state_id`),
  KEY `fk_party_id` (`party_id`),
  KEY `fk_contest_type_id` (`contest_type_id`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=116 ;
-- Results (votes, delegates) for a specific candidate at a specific event.
CREATE TABLE IF NOT EXISTS contest_candidate (
        id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
```

```
        candidate_id INT(11) NOT NULL,
        contest_id INT(11) NOT NULL,
        vote_count INT(11) DEFAULT NULL,
        delegate_count INT(11) DEFAULT NULL,
        FOREIGN KEY fk_candidate_id(candidate_id) REFERENCES candidate(id)
                ON DELETE RESTRICT,
        FOREIGN KEY fk_contest_id(contest_id) REFERENCES contest(id)
                ON DELETE RESTRICT,
        UNIQUE KEY `contest_candidate_unique` (candidate_id, contest_id)
) ENGINE=InnoDB;

CREATE TABLE IF NOT EXISTS `contest_candidate` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `candidate_id` int(11) NOT NULL,
  `contest_id` int(11) NOT NULL,
  `vote_count` int(11) DEFAULT NULL,
  `delegate_count` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `unique_index` (`candidate_id`,`contest_id`),
  KEY `fk_contest_id` (`contest_id`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=299 ;
```

General Use Queries

For our queries, we used a strategy of using Dropdown menus built using a PHP select query for inserts into any table with a Foreign Key. That is, if adding a candidate (for example), the insert table has a dropdown that is populated at runtime by the php script to select a party that actually exists in the party table. Embedded in the dropdown menu HTML tags… basically it created dropdowns with this syntax:

```
<option value=”[id]”>$label</option>
```

This was done after we wrote more complex queries to select a party where the party.name = [userInput]. Doing this vastly simplified the rest of our program and also made it essentially impossible for a single user of our site to try and insert something with an invalid foreign key because the ID’s are provided by select queries mere moments before.

With that said, the rest of this section consolidates the final queries for each of the queries used on our site. You can see all of the source code in PHP form on our repository at https://github.com/sshillyer/poli-race-2016 if desired. The select statements are in a route file at php/select-route.php, which we used to centralize all of the select statements. The insert queries are embedded in individual insert-*.php pages.

Final note: The aggregate function is the one with header: `-- Display Contest Events`

Select Statements used to build dropdowns for foreign keys on the insert page:

```
SELECT id, [attribute] FROM [table_name] ORDER BY [attribute]

-- Display all states
SELECT
```

```
        name AS 'State',
        abbreviation AS 'Abbreviation'
FROM `state` AS s
ORDER BY 'State' ASC;

-- Display Political Parties
SELECT name as 'Party'
FROM `party` AS p
ORDER BY 'Party' ASC;

-- Display Types of Contests
SELECT name AS 'Contest Type'
FROM `contest_type` AS ct
ORDER BY 'Contest Type' ASC;

-- Display Political Candidates
SELECT
        CONCAT(c.`fname`, ' ', c.`lname`) AS 'Candidate',
        p.`name` AS 'Party'
FROM
        `candidate` AS c
INNER JOIN
        `party` AS p ON c.`party_id`=p.`id`
ORDER BY
        'Candidate' ASC;

-- Display Contest Events
SELECT
        DATE(`contest`.`contest_date`) as `Date`,
        `state`.`name` as `State`,
        `party`.`name` as `Party`,
        `type`.`name` as `Contest Type`
FROM
        `contest`
INNER JOIN
        `state` ON `state`.`id`=`contest`.`state_id`
INNER JOIN
        `party` ON `party`.`id`=`contest`.`party_id`
INNER JOIN
        `contest_type` AS `type` ON `type`.`id`=`contest`.`contest_type_id`;

-- Display Contest Events
SELECT
        CONCAT(c.`fname`, ' ', c.`lname`) AS `Candidate`,
        p.`name` AS 'Party',
        SUM(`details`.`vote_count`) AS `Total Votes`,
        SUM(`details`.`delegate_count`) AS `Total Delegates`
FROM
        `candidate` as c
INNER JOIN
        `party` AS p
        ON p.`id`=c.`party_id`
INNER JOIN
        `contest_candidate` AS `details`
        ON `details`.`candidate_id`=c.`id`
GROUP BY
        `details`.`candidate_id`;

-- Display Delegates Available in Each Contest
```

```
SELECT
        s.`name` AS `State`,
        p.`name` AS `Party`,
        c.`delegates` AS `Delegates`,
        1.0 * c.`delegates` / `total_delegates`.`dels` AS `Percentage of Total Delegates`
FROM
        `state` AS s
INNER JOIN
        `contest` AS c ON s.`id` = c.`state_id`
INNER JOIN
        `party` AS p ON c.`party_id` = p.`id`
INNER JOIN
        (
        SELECT
                co.`party_id`,
                SUM(co.`delegates`) AS `dels`
        FROM
                `contest` AS co
        GROUP BY
                co.`party_id`
        ) `total_delegates` ON `total_delegates`.`party_id` = p.`id`
ORDER BY `State` ASC, `Party` ASC
```

-- Display each candidate’s most recent win

```
SELECT
        CONCAT(cn.`fname`, ' ', cn.`lname`) AS `Winner`,
        pty.`name` AS `Party`,
        st.`name` AS `State`,
        ct.`name` AS `Event`,
        DATE(`recent_win`.`contest_date`) AS `Date`,
        ctc.`delegate_count` AS `Delegates Won`,
        ctc.`vote_count` AS `Votes Received`,
        cs.`delegates` AS `Total Delegates`
FROM
        (
        SELECT
                can.`id` AS `candidate_id`,
                MAX(c.`contest_date`) AS `contest_date`
        FROM
                (
                SELECT
                        con.`id` AS `contest_id`,
                        MAX(conca.`delegate_count`) AS `max_dels`
                FROM
                        `contest` AS con
                INNER JOIN
                        `contest_candidate` AS conca ON conca.`contest_id` = con.`id`
                GROUP BY
                        con.`id`
                ) `max_per_contest`
        INNER JOIN
                `contest_candidate` AS cc ON cc.`contest_id` = `max_per_contest`.`contest_id`
        INNER JOIN
                `candidate` AS can ON can.`id` = cc.`candidate_id`
        INNER JOIN
                `contest` AS c ON c.`id` = cc.`contest_id`
        WHERE
                cc.`delegate_count` = `max_per_contest`.`max_dels`
```

```
        GROUP BY
                `candidate_id`
        ) `recent_win`
INNER JOIN
        `candidate` AS cn ON cn.`id` = `recent_win`.`candidate_id`
INNER JOIN
        `contest_candidate` AS ctc ON ctc.`candidate_id` = cn.`id`
INNER JOIN
        `contest` AS cs ON cs.`id` = ctc.`contest_id`
                                        AND `recent_win`.`contest_date` = cs.`contest_date`
INNER JOIN
        `contest_type` AS ct ON ct.`id` = cs.`contest_type_id`
INNER JOIN
        `party` AS pty ON pty.`id` = cn.`party_id`
INNER JOIN
        `state` AS st ON st.`id` = cs.`state_id`
INNER JOIN
        (
        SELECT
                con.`id` AS `contest_id`,
                MAX(conca.`delegate_count`) AS `max_dels`
        FROM
                `contest` AS con
        INNER JOIN
                `contest_candidate` AS conca ON conca.`contest_id` = con.`id`
        GROUP BY
                con.`id`
        ) `max_per_contest` ON `max_per_contest`.`contest_id` = cs.`id`
                                                AND `max_per_contest`.`max_dels` = ctc.`delegate_count`
ORDER BY
        `Party` ASC

-- Insert Candidate
INSERT INTO candidate(fname, lname, party_id)
VALUES([candidate_fname], [candidate_lname], [candidate_party_id]);

-- Insert contest-candidate
INSERT INTO contest_candidate(candidate_id, contest_id, vote_count, delegate_count)
VALUES( [candidate_id],
        (SELECT id FROM contest WHERE
                contest.state_id=[contest_state_id] AND
                contest.party_id=(SELECT party_id FROM candidate WHERE candidate.id=[candidate_id])
        ),
        [contest_votes],
        [contest_delegates]);

-- Insert contest-type
INSERT INTO contest_type(name) VALUES([contest_type]);

-- Insert contest
INSERT INTO contest(contest_date, state_id, party_id, contest_type_id, delegates)
VALUES ([contest_date], [contest_state_id], [contest_party_id], [contest_type_id], [contest_delegates]);

-- Insert party
INSERT INTO party(name) VALUES([party_name]);

-- Insert state
INSERT INTO state(name, abbreviation) VALUES([state_name], [state_abbrev]);
```

```
-- Search for Candidate by Last Name
SELECT CONCAT(c.`fname`, ' ', c.`lname`) AS 'Candidate', p.`name` AS 'Party' FROM `candidate` AS c INNER
JOIN `party` AS p ON c.`party_id`=p.`id` WHERE c.`lname`='[lname]' ORDER BY 'Candidate' ASC
```

HTML and PHP

Our site is hosted at:
http://web.engr.oregonstate.edu/~hillyers/poli-race-2016/site/index.php

Our source code is hosted at:
https://github.com/sshillyer/poli-race-2016