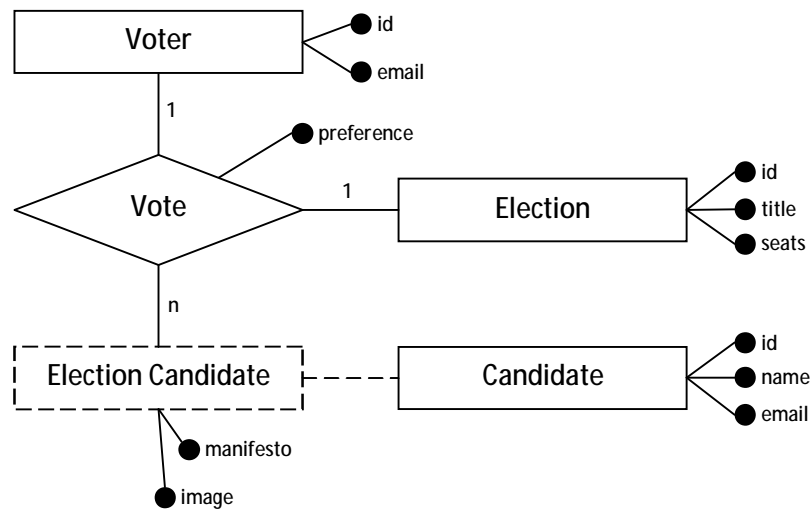


## Entity Relationship



## Relational Schema

voter(hash,email)

election(electionid,title,seats)

vote(electionid,candidateid,rhash)

candidate(candidateid,electionid,name,email,manifesto,image)

## Comments

The relational schema has merged entity *Candidate* with associated weak entity *Election Candidate* and specifies that they are to be implemented as a single table. This ad-hoc design decision was specific to version 0.1 and was made to accommodate the NUS LGBT delegate elections alone. Future versions will better reflect the entity relationship model.

## Hashes as unique identifiers for voters

High-level code is required to generate the 'hash' and 'reverse hash' used as unique identifiers for voters. The idea is to apply a secure hash function to the voter's email address as a string. This has the useful effect of creating an obfuscated URL with the hash as a parameter value within the query string. These strings are intended to be irreversable; they can only be attributed to a voter by being recorded against an email address in the *voters* table.

The 'reverse hash' works on the same principle; a hash generated with the voter's email address used as a parameter somewhere along the line; simply put: the more complex, the more secure. Higher-level programs will generate this hash when accessing the database and use it to prevent

duplicate voting. The presence of any records in the *votes* table against this voter identifier indicates that they have already voted.

The reverse hash also has the added effect of protecting voter anonymity; voters cannot be identified by simply querying the database AND the reverse hash cannot be transformed back into the email address through higher-level code.

### Calculation of Results

In the specific instance of the NUS delegate elections, the number of seats was indeterminate. This was reflected as a '0' value in the *seats* field of the record on table *election*. As this factor prevented the setting of a definitive quota for an STV or ATV election, a method of calculating an election score that takes all preferences into account was devised instead:

$$S_c = \sum_{i=1}^n \frac{(n - i + 1)}{n} v_{c,i}$$

Score  $S$  for candidate  $C$  ( $S_c$ ) is derived from the sum of each vote of a particular preference for candidate ( $v_{c,i}$ ) multiplied by an intrinsic value for that preference. Variable  $n$  defines the total number of candidates for the election, while  $i$  is an iterative value representing the preferences between 1 and  $n$ . In an election with 10 candidates, first-preference votes have a value of 1, or more precisely  $\frac{10}{10}$ , second-preference votes have a value of  $\frac{9}{10}$ , third-preference votes have a value of  $\frac{8}{10}$ ; this pattern continues until the 10<sup>th</sup> preference which has a value of  $\frac{1}{10}$ . This produces a realistic score which does not dismiss the assigned preferences and needs no quota.

The data can, of course, be used to accommodate the ATV and STV algorithms if so implemented in a higher-level program.