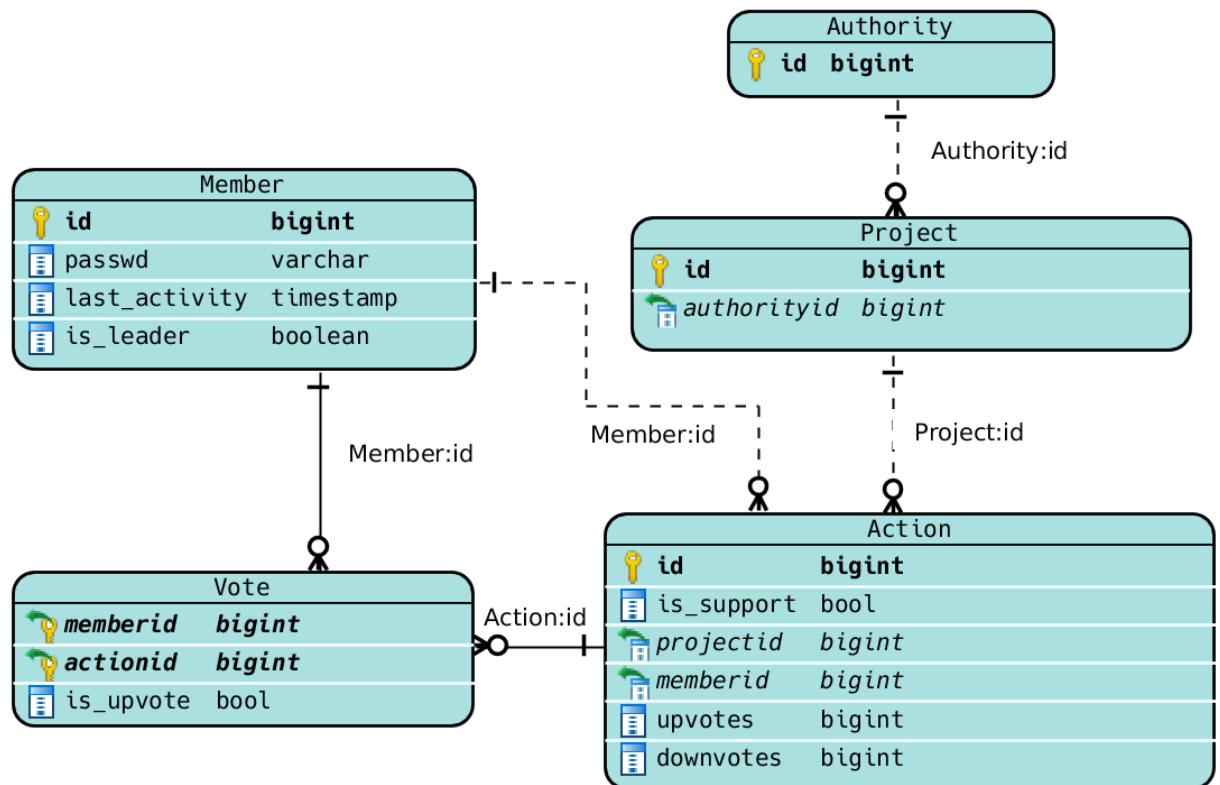


System zarządzania partią polityczną – dokumentacja projektu

Jakub Grobelny

12 czerwca 2019

1 Diagram E-R



2 Opis tabel

- Tabela *Authority* zawiera spis wszystkich organów władzy (przechowywane są jedynie ich identyfikatory.)
- Tabela *Member* zawiera dane wszystkich członków partii.
 - *id* – identyfikator członka.
 - *passwd_hash* – zahaszkowane hasło członka.
 - *last_activity* – czas ostatniej aktywności członka używany w celu stwierdzenia, czy jego konto powinno być zamrożone.

- *is_leader* – wartość boolowska prawdziwa jeżeli dany członek jest liderem partii. W przeciwnym razie fałsz.
- Tabela *Project* zawiera wszystkie projekty organizowane przez organy władzy.
 - *id* – identyfikator projektu.
 - *authorityid* – identyfikator organu władzy organizującego dany projekt. Klucz obcy.
- Tabela *Action* zawiera wszystkie akcje stworzone przez członków partii.
 - *id* – identyfikator akcji
 - *is_support* – wartość boolowska prawdziwa, gdy dana akcja popiera projekt organu władzy. Fałszywa, gdy akcja jest protestem
 - *projectid* – identyfikator projektu, którego dotyczy akcja. Klucz obcy.
 - *memberid* – identyfikator członka, który utworzył akcję. Klucz obcy.
 - *upvotes* – liczba wszystkich głosów za daną akcję. Pomaga w szybkim wyszukiwaniu *trolli*.
 - *downvotes* – liczba wszystkich głosów przeciw danej akcji. Pomaga w szybkim wyszukiwaniu *trolli*.
- Tabela *Vote* zawiera spis wszystkich głosów za i przeciw, które zostały oddane na akcje przez członków partii.
 - *membeid* – identyfikator członka, który oddał dany głos. Klucz obcy.
 - *actionid* – identyfikator akcji, na którą oddany został dany głos. Klucz obcy.
 - *is_upvote* – wartość boolowska prawdziwa, gdy dany głos jest głosem *za*. Fałsz gdy głos jest *przeciw*.

3 Użytkownicy

- **init** – użytkownik mający uprawnienia potrzebne do wstępnego zainicjowania bazy danych. Powinien móc tworzyć tabele (**CREATE**), wstawiać do nich wartości (**INSERT**) oraz nadawać uprawnienia innym użytkownikom (**GRANT**) aby móc utworzyć użytkownika **app** (**CREATE USER**).
- **app** – użytkownik mogący odczytywać i modyfikować zawartości wszystkich tabel (**SELECT**, **UPDATE**, **INSERT**) ale niemogący modyfikować schematu bazy danych.

4 Sposób implementacji funkcji API

Uwaga: ze względu na to, że identyfikatory powinny być globalnie unikatowe, to każda funkcja tworząca nowe krotki powinna również sprawdzać, czy dany identyfikator nie pojawił się już dotychczas w którejkolwiek z tabel.

Dodatkowo wszystkie działania wykonywane przez członków powodują, że ich atrybut *last_activity* zostaje zaktualizowany, jeżeli nie zostali jeszcze zamrożeni. Funkcje powinny również wcześniej sprawdzać, czy członek wykonujący jakieś działanie nie jest zamrożony – jeżeli jest, to zwracany będzie błąd.

Jeżeli członek nie istnieje, to zostanie utworzony poprzez dodanie odpowiedniej krotki do tabeli *Member*.

Jeżeli nie istnieje jakiś projekt bądź organ władzy wspomniany w argumentach któregoś z funkcji, to wówczas również powinien zostać dodany do bazy danych.

- **open** – w trybie **init** funkcja **open** utworzy wszystkie tabele opisane powyżej oraz stworzy użytkownika **app**. Przy kolejnych uruchomieniach utworzona zostanie sesja dla podanego użytkownika
- **leader** – funkcja wstawi do tabeli *Member* nowego członka, którego atrybut *is_leader* zostanie ustawiony na prawdę.
- **support** – funkcja doda odpowiednią tabeli *Actions*, której atrybut *is_support* przyjmie wartość prawdę.
- **protest** – funkcja analogiczna do **support**, ale w jej przypadku atrybut *is_support* ustawiony zostanie na fałsz.

- `upvote` – funkcja doda nową krotkę do tabeli *Vote* pod warunkiem, że dany członek nie głosował jeszcze na daną akcję. Atrybut *is_upvote* przyjmie wartość fałsz. Dodatkowo dla tej akcji zwiększona zostanie wartość *upvotes* w tabeli *Action*.
- `downvote` – funkcja analogiczna do `upvote` ale ustawiająca wartość atrybutu *is_upvote* na fałsz i zwiększająca atrybut *downvotes* zamiast *upvotes* w tabeli *Action*.
- `actions` – funkcja sprawdzi, czy członek wywołujący funkcję jest liderem, a następnie wykona odpowiednie zapytanie w celu pozyskania danych z tabeli *Action*, które zostaną odpowiednio przefiltrowane na podstawie podanych argumentów.
- `projects` – funkcja analogiczna do `actions` ale zapytanie dotyczy będzie tabeli *Project*.
- `votes` – funkcja sprawdzi, czy członek wywołujący funkcję jest liderem a następnie wykona zapytanie, które najpierw zliczy liczbę głosów każdego typu użytkowników w tabeli *Vote* a potem doda do wyników pozostałych użytkowników, którzy nie oddali być może żadnych głosów.
- `trolls` – funkcja użyje wartości atrybutów *upvotes* i *downvotes* z tabeli *Action* żeby szybko obliczyć bilans głosów dla każdej akcji, a następnie pogrupuje akcje według członków, którzy je inicjowali, obliczy sumaryczne bilanse głosów i wykluczy z wyniku tych członków, dla których ów bilans był pozytywny.

API zostanie zaimplementowane we języku *Haskell* przy użyciu biblioteki `postgresql-simple`.

5 Sposób uruchomienia aplikacji

Przed uruchomieniem programu należy zainstalować `ghc` oraz `cabal-install`. Aby skompilować aplikację należy albo uruchomić skrypt `build.sh` albo wykonać następujące polecenia:

- `cabal update`
- `cabal sandbox init`

- `cabal install --only-dependencies`
- `cabal build`

Skompilowany plik wykonywalny pojawi się w folderze projektu w katalogu `dist/build/databases-project`, i będzie miał nazwę `databases-project`.

Dodatkowo w głównym folderze projektu po skompilowaniu znajdować będzie się skrót `exe` do wygodniejszego uruchamiania programu.

Aby usunąć pliki powstałe po kompilacji należy uruchomić skrypt `clean.sh`.