Projekt: System Zarządzania Partią Polityczną

Krzysztof Pyrkosz
Instytut Informatyki Uniwersytetu Wrocławskiego

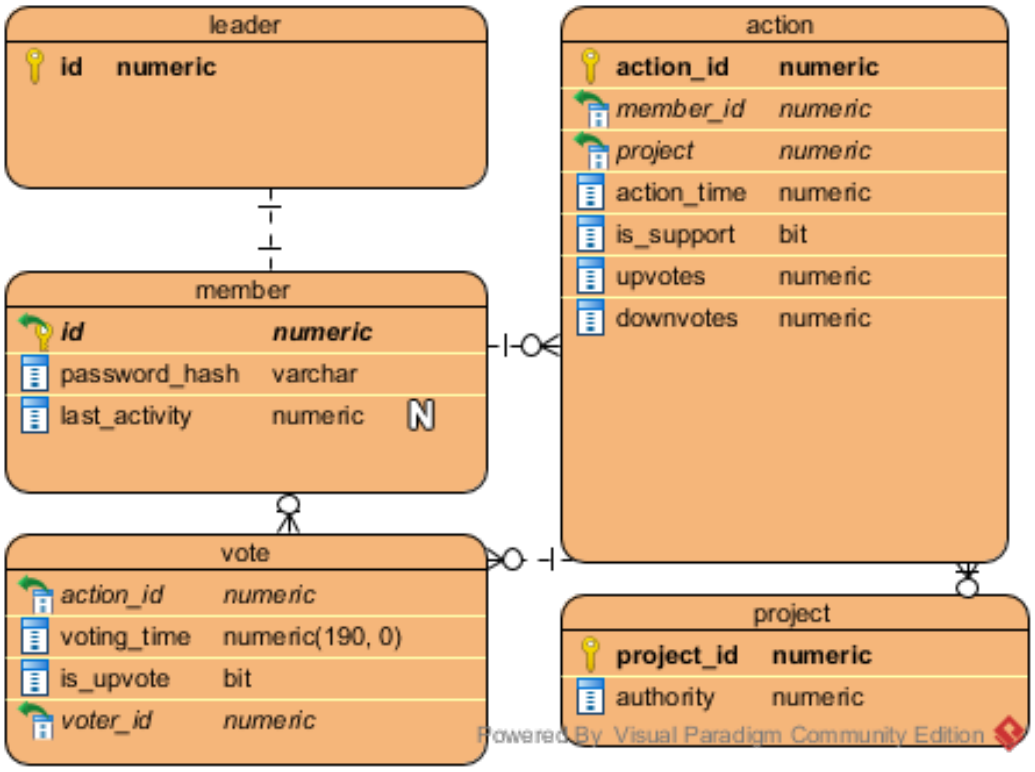
10 czerwca 2019

Streszczenie

Niniejszy plik stanowi dokumentację systemu napisanego w C++. Znajdują się tu model konceptualny, informacje o tabelach i innych elementach bazy danych, opisy praw użytkowników init/app, zwięzła informacja jak zaimplementowane są poszczególne funkcje oraz informacje dotyczące kompilacji i uruchamiania.

Spis treści

1 Model konceptualny



1.1 Więzy i zależności

Kluczem głównym tablicy *member* jest *id*. Lider wyróżniony jest poprzez istnienie krotki z jego *id* w tablicy *leader*, odwołującej się do *id* członka.

Action posiada referencję do *member* identyfikującą inicjatora akcji protestu lub wsparcia, oraz do *project* oznaczającą konkretny projekt którego akcja dotyczy.

Dane w tabeli *vote* związane są z głosującym *member* oraz *action*.

1.2 Funkcje pomocnicze

Zadeklarowałem dwie pomocnicze *SQL*-owe funkcje

- *make_leader(id, password)* dodająca nowego lidera, działająca wyłącznie w trybie –init
- *is_member_active(id, timestamp)* zwracająca prawdę lub fałsz w zależności od statusu członka

1.3 Opis praw użytkowników init i app

Użytkownik *init* odpowiedzialny jest za utworzenie tabel, więzów, funkcji oraz pozostałych elementów bazy danych. Musi również przygotować użytkownika *app* i nadać mu odpowiednie prawa, wystarczające do użytkowania świeżo zainicjowanej bazy.

Użytkownik *app* posiadać musi minimalny zbiór uprawnień wystarczający do działania, tak aby mógł na przykład odczytać informacje o liderach, lecz nie był w stanie ich zmodyfikować. Operacje *SELECT*, *INSERT* oraz *UPDATE* dostępne są dla tabel *member* (aktualizowanie timestampów) oraz *action* (aktualizowanie liczników).

1.4 Sposób implementacji funkcji API

Funkcje wymagające upoważnienia hasłem przed właściwą akcją sprawdzają, czy członek o danym identyfikatorze istnieje, następnie jego hasło i na koniec stan aktywności (czas ostatniej akcji).

- *open* - to wywołanie musi być podane jako pierwsze po uruchomieniu programu. Wyspecjalizowana klasa odpowiedzialna za pośredniczenie między aplikacją a bazą danych spróbuje nawiązać połączenie, w przypadku niepowodzenia zgłosi błąd, w przeciwnym razie aplikacja przejdzie w stan nasłuchiwania kolejnych poleceń.
- *leader* - polega na wywołaniu funkcji *make_leader(id, password)*, która pod spodem wstawia krotkę do tablicy *member* oraz *leader* oznaczającą członka będącego liderem.
- *support protest* - obie funkcje zostaną zaimplementowane jako jedna, a rozpoznawane będą przez flagę *is_support* w tabeli *action*. W pierwszej kolejności utworzę *project* jeśli nie istniał, upewniając się przy tym, że *authority* zostało podane. Później tworzę wpis w tabeli *action* z detalami zapytania, aktualizuję czas ostatniej aktywności inicjatora.

- *upvote downvote* - również zaimplementowane jako jedna klasa. Po walidacji danych członka sprawdzam, czy już głosował, jeśli nie, dodaję odpowiednią krotkę do *vote* oraz aktualizuję liczniki w *action*.

- *actions* - oprócz sprawdzania poprawności hasła dodatkowo upewniam się, że osoba jest liderem. Zapytanie będzie agregować liczby *upvote* oraz *downvote*, z dodatkowymi obostrzeniami zależnymi od wystąpienia ograniczeń w postaci *type*, *project* czy *authority*.

- *projects* - sprowadza się do wypisania danych z tabeli *project*, ewentualnie z ograniczeniem do jednego *authority*.

- *votes* - polega na złączeniu tabeli *member* z *action* aby mieć pewność, że uwzględnię również członków którzy nigdy nie głosowali. Zapytanie będzie korzystać z liczników w tabeli *action*.

- *trolls* - dzięki redundancji w tabeli *action* nie muszę dynamicznie sumować głosów za i przeciw z tabeli *vote*, co przyspiesza zapytanie. Na tej podstawie trywialnie wyznaczam i odpowiednio sortuję podejrzanych członków, dodając przy tym informację o statusie aktywności.

2 Budowa i uruchamianie aplikacji

2.1 Struktura katalogów

- *inc* - zawiera pliki nagłówkowe C++.

- *src* - zawiera pliki źródłowe C++.

- *resources* - skrypt *init.sql* odpowiedzialny za odpowiednie zainicjowanie bazy w trybie –init, pomocny skrypt *drop.sql* służący do wyczyszczenia bazy do stanu sprzed –init.

- *documentation* - ten plik pdf.

- *third_party* - znajduje się tu open-source'owa biblioteka do parsowania obiektów JSON.

2.2 Kompilacja

Do zbudowania projektu wymagane są system budowania *CMake*, kompilator wspierający standard C++11 oraz oficjalna biblioteka służąca do łączenia z bazą z poziomu kodu C/C++ *libpq*. Program buduję się z włączonymi flagami ostrzeżeń oraz posiada odpowiednie asercje w trybie *Debug*.
Polecam utworzyć katalog *build*, wywołać w nim *cmake ..* (CMakeLists.txt znajduje się wtedy w katalogu jeden poziom wyżej). Następnie wykonujemy polecenie *make*, które zbuduje program oraz skopiuje zależności takie jak *init.sql* do katalogu z aplikacją. Program jest gotowy do uruchomienia.

2.3 Co zostało zaimplementowane

Działają wszystkie polecenia API. Program zakłada, że pierwszą linią wejścia musi być komenda *open*. W trybie –init dostępna jest wyłącznie funkcja *leader*, wszystkie pozostałe polecenia w trybie "zwykłym".