Dokumentacja / Model Koncepcyjny

Karol Mirek

May 2019

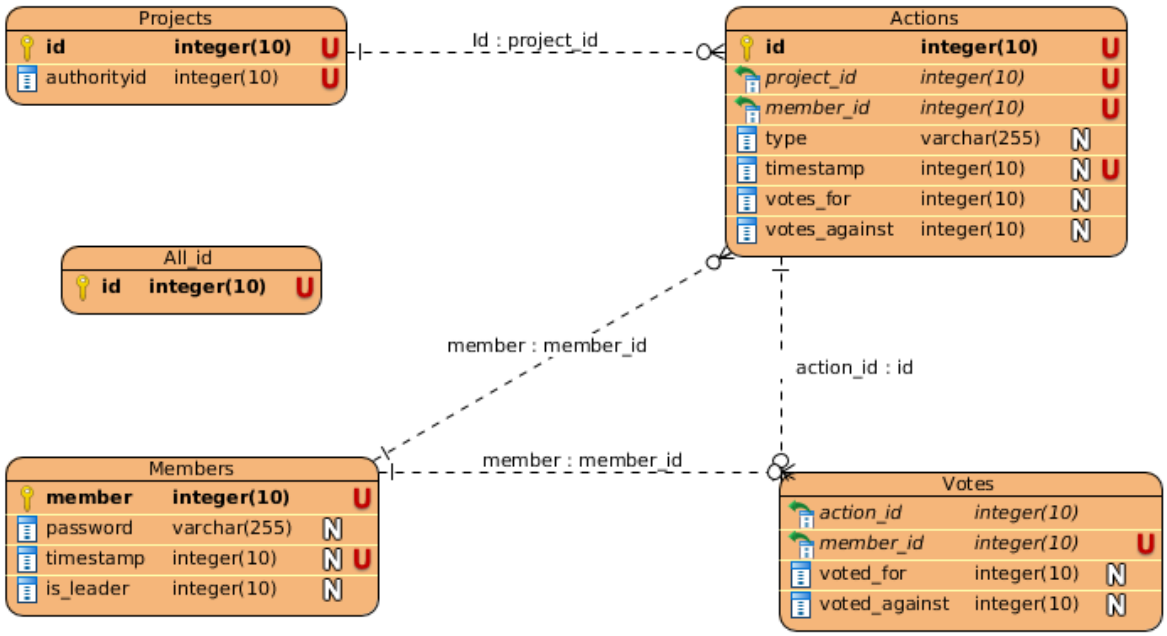
1 Opis Projektu

Projekt został napisany w języku python z rozszerzeniami psycopg2(do sql), json(do jsona), sys(wejście), bcrypt(hashowanie haseł).

Instrukcja uruchomienia

Przed uruchomieniem trzeba upewnić się, że posiadamy SQL usera 'init' z hasłem 'qwerty' oraz bazę 'student'.
Komendy sql do stworzenia potrzebnych rzeczy :
CREATE USER init with password 'qwerty';
CREATE database student;
ALTER ROLE init with superuser ;
GRANT SELECT, INSERT, UPDATE ON ALL TABLES IN SCHEMA public TO init;
Uruchomienie programu z terminalu :
python3 projektMirek.py input-file >output-file
W archiwum daję dodatkowe pliki init.in.json oraz app.in.json do testowania . Zawierają testy, którę zostały wrzucone na gita przez wykładowcę.
Przykładowe wywołania :
python3 projektMirek.py init.in.json >init.out.json zawiera(Open z initem + leader)
python3 projektMirek.py app.in.json >app.out.json zawiera(Open z appem + tworzenie akcji + głosowania wywołanie trolls)

2 Diagram E-R


Projects
id integer(10) U
authorityid integer(10) U
Id : project_id
Actions
id integer(10) U
project_id integer(10) U
member_id integer(10) U
type varchar(255) N
timestamp integer(10) N U
votes_for integer(10) N
votes_against integer(10) N
All_id
id integer(10) U
member : member_id
action_id : id
member : member_id
Members
member integer(10) U
password varchar(255) N
timestamp integer(10) N U
is_leader integer(10) N
Votes
action_id integer(10)
member_id integer(10) U
voted_for integer(10) N
voted_against integer(10) N

3 Opis Tabel

Tabela Members

Zawiera wszystkie informacje dotyczące członków partii:
- member (id membera)
- password (zhashowane hasło użytkownika)
- timestamp (ostatnia aktywność użytkownika)s
- is-leader (czy jest liderem)

Table Projects

Posiada informację o projetkach partii politycznej oraz ich authorities.
- id (identyfikator projektu)
- authority (idetyfikator organu władzy)

Tabela Actions

Zawiera wszystkie akcję stworzone przez partie
- id (identyfikator akcji)
- project-id (identyfikator projektu, którego dotyczy akcja, klucz obcy)
- member-id (id membera, który stworzył akcję, klucz obcy)
- type (typ akcjii support/protest)
- timestamp (czas utworzenie)
- votes-for/against (ilośc głosów za i przeciwko)

Tabela Votes

Zbiór głosów oddanych na dane akcje.
- action-id (id akcji na, którą jest głos, klucz obcy)
- member-id (kto oddał dany głos, klucz obcy)
- voted-for/voted-against (czy głos był za czy przeciw)

Tabela All-id

Zawiera wszystkie możliwe id w bazie (aby uniknąć duplikatów).

4 Uprawnienia INIT oraz APP

Init Init jest użytkownikiem inicjalizującym całą bazę danych. Czyli po jego wywołaniu stworzy wszylkie tabele. Może on też na samym początku dodać krotki członków do bazy danych. Tworzy użytkownika app.

App App jest typowym użytkownikiem naszej aplikacji. Posiada możliwość zaproponowania nowych akcji oraz głosowanie na nie. Może wyświetlać i modyfikować zawartości wszystkich tabel. Nie może modyfikować schematu bazy danych .

5 Funkcję API

Open

Gdy funkcja 'Open' jest uruchamiana z loginem = 'init', inicjuje całą baze danych wraz z użytkownikiem app, w przeciwnym wypadku po prostu tworzy połączenie.
W moim przypadku łączymy się za do bazy SQL za pomocą rozszerzenia psycpg2 funkcja psycopg2.connect(). Na samym początku połączenie jest nawiązywane od użytkownika 'init' z hasłem 'qwerty' do bazy 'student'. Aby uruchomić program za pierwszym razem trzeba mieć od razu stworzonego w sql takiego użytkowika i bazę . W kolejnych uruchomieniach połączenie powinno być nawiązywanie od użytkownika app, który został stworzony z inicjacją bazy, hasło app to również 'qwerty'.

Leader

Do tabeli Members wstawi członka, którego atrybut is-leader będzie wynosił 1.
Sql : INSERT INTO Members (timestamp,password,member,is-leader) VALUES (x,y,z,w);

Support oraz Protest

Funkcje tworzące akcję supportu lub protestu. Do tabeli Actions doda odpowiednie krotki, po wcześniejszym sprawdzeniu membera(czy jest zamrożony czy istnieje). Jeśli akcja jest dla projektu, który nie istnieje to go tworzymy.
Dodawanie do tabeli Actions: INSERT INTO Actions (id,project-id,member-id,type,timestamp) Values (x, y, w, z, v);
Tworzenie projektu :INSERT INTO Projects (id,authorityid) VAlues (x, y);

Upvote, Downvote

Do tabeli Votes doda odpowiednie krotki, musi na początku sprawdzić czy dana akcja istnieje oraz czy dany użytkownik już nie głosował na daną akcje. Oczywiśćie standardowe checki dla membera. Jeśli akcja nie istnieje to zwraca "ERROR" . Dla odpowiedniej akcji zmieniamy atrybut votes-against lub votes-for.

Dodawanie votes: Insert into votes (action-id,member-id,voted-against) Values (x, y, w);
Zmiana atrybutu Akcji :Update Actions SET votes-against = votes-against+1 Where id = x;

Actions, Projects, Votes

Sprawdzamy czy użytkownik, który wywołuje dane funkcję jest liderem . Potem odpowiednym SELECTEM zwracamy wartości.
Przykładowe wywołanie Actions:
SELECT Actions.id,type,project-id,authorityid,votes-for,votes-against from Actions join Projects ON(Actions.project-id = Projects.id) ORDER By Actions.Id ;
Przykładowe wywołanie Projects :
SELECT * from Projects ORDER By id asc;
Przykładowe wywołanie Votes :
SELECT member,SUM(coalesce(voted-for,0)) as a,SUM(coalesce(voted-against,0)) as b from Members LEFT JOIN Votes on (Members.member=Votes.member-id) Group by member Order by member;

Trolls

Dla każdego użytkownika, który stowrzył akcję sumujemy liczbę głosów oddanych za i przeciwko jego akcjom. A potem wykluczamy tych użytkowników u, których liczba głosów za była większa niż przeciw .
Zapytanie sql :
Select * from (Select member, SUM(votes-for) as a, SUM(votes-against) as b, Case WHEN timestamp - Members.timestamp <31536000 Then 'true' else 'false' END from Actions join Members ON(Actions.member-id=member) Group by member) as foo where b >a Order by b-a DESC, member ASC;