System zarządzania partią polityczną - dokumentacja

Patryk Kumor

11 czerwca 2019

Spis treści

1 Potrzebne pakiety i uruchomienie programu

1.1 Potrzebne pakiety

Program został napisany w Pythonie 2.7, z użyciem następujących bibliotek:

- argparse
- json
- sys
- psycopg2

Wszystkie powyższe moduły, oprócz ostatniego - psycopg2 - powinny być dostarczone wraz z podstawową dystrybucją pythona.
Aby zainstalować psycopg2 należy skorzystać z PIP - package managera do języka python, w tym celu należy wykonać polecenie:

```
\$ pip install psycopg2
```

1.2 Uruchomienie programu

Program do zarządzania partią przyjmuje na wejściu obiekty json, które są odczytywane jako ciąg wywołań funkcji API.
Program rozróżnia kilka typów wywołań:

Aby wywołać program z odczytem linii zawierających obiekty json (jeden obiekt na linię) ze standardowego wejścia w pętli:

- dla pierwszego wywołania wraz z flagą init:

  ```
  \$ python main.py --init
  ```

- dla kolejnych wowołań:

  ```
  \$ python main.py
  ```

Aby wywołać program wraz z odczytem standardowego wejścia zawartego w pliku (każda linia w pliku jest obiektem json):

- dla pierwszego wywołania wraz z flagą init:

  ```
  \$ python main.py --init < <input_file>
  ```

- dla kolejnych wowołań:

  ```
  \$ python main.py < <input_file>
  ```

Aby wywołać program wraz z odczytem zawartości pliku podanego jako argument (każda linia w pliku jest obiektem json) należy użyć flagi --f
(program po skończeniu odczytywania zawartości pliku przechodzi w tryb ciągłego czytania ze standardowego wejścia):

- dla pierwszego wywołania wraz z flagą init:

```
\$ python main.py --init --f <input_file>
```

- dla kolejnych wowołań:

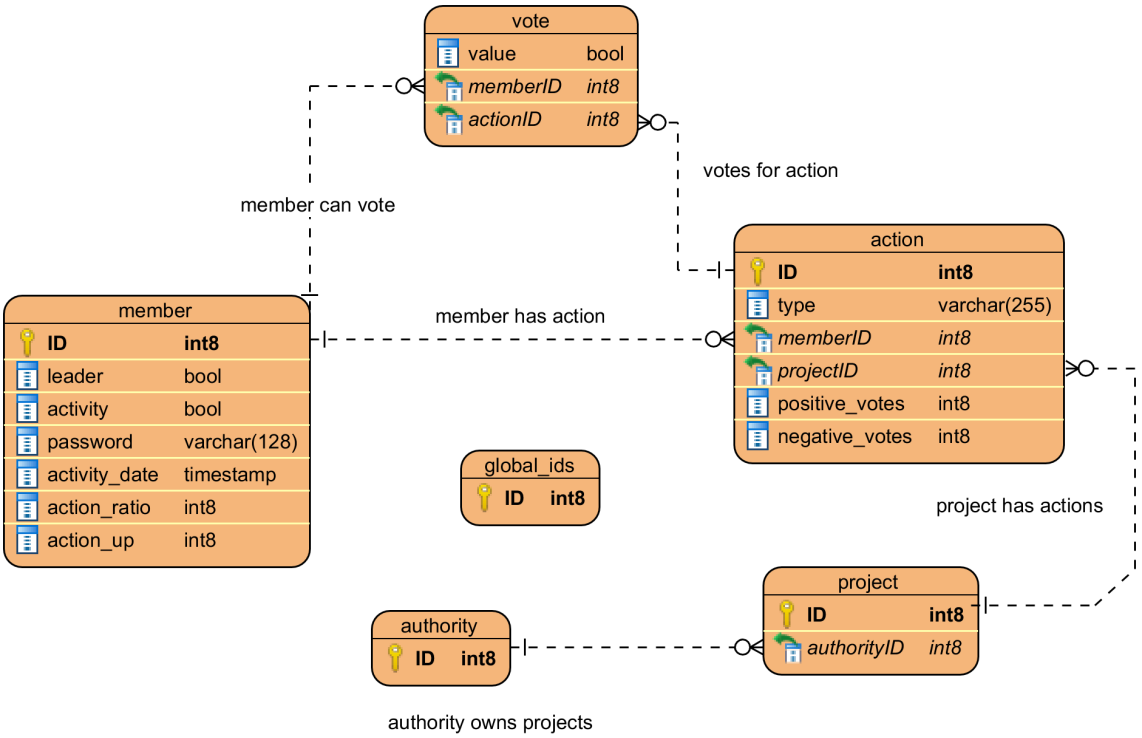
```
\$ python main.py --f <input_file>
```

Zakończenie pracy programu

Program kończy się automatycznie w przypadku odczytania całej zawartości pliku, w przypadku ciągłego odczytywania kolejnych linii program zakończy się gdy zostanie użyty skrót **ctrl**+**c**.

2 Model fizyczny

Program przy swoim pierwszym uruchomieniu (wraz z argumentem --init) po pomyślnym połączeniu z bazą danych tworzy w niej wszystkie potrzebne mu elementy, które są przedstawione na poniższym rysunku.
(Cały schemat użyty podczas tworzenia zawartości bazy danych zawarty jest w pliku **base.sql**)



Użytkownik init, za pomocą którego łączymy się z bazą, posiada wszystkie uprawnienia potrzebne w programie, w tym musi mieć uprawienia do: CREATE TABLE, ADD CONSTRAINT, CREATE user, GRANT <przywilej>. Użytkownik init również tworzy nową rolę **app** z uprawieniami UPDATE, INSERT i SELECT na powyższych tabelach.

Tabela **global_ids** przechowuje wszystkie id, które zostały użyte do tej pory w programie, dzięki temu jesteśmy w stanie kontrolować globalną unikalność id we wszystkich tabelach
Wytłumaczenie zawartości kolumn, które mogą być niejasne:

- value w **vote** - przechowuje informację o tym czy oddany głos jest za (True) czy przeciw (False)
- positive_votes i negative_votes w **action** - przechowuje informację o oddanej sumarycznej liczbie głosów za/przeciw wobec danej akcji
- action_up w **member** - przechowuje wartość wszystkich głosów upvotes wobec projektów które dany członek utworzył
- action_ratio w **member**- przechowuje wartość (downvotes - upvotes), która potrzebna jest do uznania członka za trolla (jeśli jest dodatnia)

3 Implementacja

3.1 Dokładny opis wejścia

Obiekty json podawane na wejściu mają strukturę:

```
{
    <function> : <value>        // nazwa funkcji
    {
        <arg1> : <value>        // argumenty funkcji
        <arg2>  : <value>
        . . .
        <argn>  : <value>
    }
}
```

Na przykład funkcja **open** z argumentami <**database**> <**login**> oraz <**password**> może zostać przekazana na wejściu jako:

```
{ "open": { "database": "student", "login": "app", "password": "qwerty"}}
```

Skutkuje to wywołaniem przez program odpowiedniej funkcji przetwarzającej taki obiekt (mającej na celu ustanowienie połączenia z bazą danych wraz z danymi dostępowymi podanymi w argumentach).

3.2 INIT i kolejne wywołania programu

Podczas pierwszego uruchomienia programu należy użyć flagi --init. Podczas uruchomienia --init program pobiera wyłącznie jsony z funkcją open i leader. Jako pierwszy json pobrany powinien być ten zawierający <open>, który definiuje elementy bazy i nawiązuje z nią połączenie, w przypadku niepowodzenia funkcji obsługującej <open> program zakończy działanie.
Kolejne wywoałania funkcji obsługującej <leader> będą definiować nowe krotki członków, którzy są leaderami.

Kolejne wywołania programu (bez flagi --init) pobierają i obsługują tylko jsony z funkcjami: open, support/protest, upvote/downvote, actions, projects, votes i trolls, które zostaną szczegółowo opisane w dalszej części dokumentacji. Również tutaj w przypadku niepowodzenia <open> program zakończy pracę.

Program przetwarza wejście parserem json, po czym przekazuje je do funkcji **if_case(dic, case)**, której argumentem jest nowo utworzony słownik powstały na wskutek działania parsera, oraz nazwa funkcji zawarta w jsonie.

W dalszej części dokumentacji opisane będą sposoby działania funkcji do których zostanie przekierowany program. Wyjaśnienie oznaczenia we oznaczenia użytego we wszystkich funkcjach głównych odpowiedzialnch za wykonywanie poleceń z jsona:

- arg - jest zawartością jsona
- conn = psycopg2.connect(args)
- cur = conn.cursor()

3.3 <open> w funkcjach f_open_init i f_open_normal

Pierwsza z funkcji f_open_init przeznaczona jest do wykonania poleceń pierwszego jsona na wejściu wraz z wykrytą flagą --init, funkcja ta dodatkowo jest odpowiedzialna za pobranie poleceń bazodanowych z pliku .sql:

Opis działania: Na początku należy ustanowić globalne połączenie z bazą danych, pobieramy dane logowania z obiektu json. Iniciujemy globalny kursor odpowiadający za wykonywanie poleceń bazodanowych. Następnie pobieramy zawartość pliku .sql i wykonujemy execute poleceń towrzących niezbędne elementy bazy danych (wraz z użytkownikiem app). Wykonujemy commit zmian w bazie danych, w przypadku wykrycia jakiegokolwiek błędu cofamy wszystkie zmiany i zamykamy program.

```
def f_open_init(arg):
    try:
        global conn
        conn = psycopg2.connect(host="localhost", port="5432",
                                dbname=arg['open']['database'],
                                user=arg['open']['login'],
                                password=arg['open']['password'])
        global cur
        cur = conn.cursor()
        database_create = open('base.sql','r')
        cur.execute(database_create.read())
        conn.commit()
        print '{"status" : "OK"}'
    except Exception as err:
        conn.rollback()
        print '{"status" : "ERROR",\n "debug" : "%s" }' % str(err)
        sys.exit(0)
```

Funkcja f_open_normal działa na takiej samej zasadzie, ale pomijamy tu pobieranie poleceń z pliku .sql (które powinny być już wcześniej wykonane)

3.4 <leader> w funkcji f_leader

Opis działania: W funkcji pomocniczej **check_id** poprzez zapytanie SELECT do bazy danych (wobec tabeli global_ids) sprawdzamy czy dane nam w argumencie ID jest wolne do użycia. Jeśli jest wolne to wywołujemy funkcję **add_member**, która odpowiada za INSERT krotki do bazy danych o podanych argumentach wejściowych w tabeli member oraz wpisaniu id do tabeli global_ids (w celi kontroli argumentów kolejnych wywołań - unikalnego id).

```
def f_leader(arg):
    if check_id(arg['leader']['member']):
        try:
            add_member(arg['leader']['member'],
                       arg['leader']['password'],
                       arg['leader']['timestamp'],
                       True) # Czy użytkownik jest leaderem
            conn.commit()
            print '{"status" : "OK"}'
        except Exception as err:
            # Rollback jeśli coś poszło nie tak
            conn.rollback()
            print '{"status" : "ERROR",\n "debug" : "%s" }' % str(err)
    else:
        print '{"status" : "ERROR",\n "debug" : "ID jest już używany" }'
```

W celu bezpiecznego przechowywania hasła używamy modułu pgcrypto prosto w bazie danych

```
crypt(%s, gen_salt('md5'))
```

3.5 <protest>/<support> w funkcji f_protest/f_support

Opis działania: Na starcie w **authorize_or_create_member** dokonujemy autoryzacji użytkownika (jeśli istnieje, wpp dodajemy nową krotkę do bazy) sprawdzając jego hasło, potem jego aktywność (przy okazji aktualizując ją, jeżeli użytkownik powinien zostać zamrożony, lub wpp aktualizując ostatnią aktywność użytkownika za pomocą timestampa).
Jeśli podano opcjonalny argument 'authority' jest on pobierany lub wpp oznaczany jako None. Następnie argumenty są przekazywane do funkcji pomocniczej **authorize_action**, która sprawdza wolne ID akcji po czym dodaje je (wraz z kontrolą poprawności ID, istnienia i poprawności projektu/authority etc.) - funkcja powinna zwrócić 1 w przypadku powodzenia wpp 0.
Commit w przypadku powodzenia, rollback w przypadku niepowodzenia, tak jak w poprzedniej funkcji.

```
def f_protest(arg):
    if authorize_or_create_member(arg['protest']['member'],
                                  arg['protest']['password'],
                                  arg['protest']['timestamp']):
        if 'authority' in arg['protest']:
            authority = arg['protest']['authority']
        else:
            authority = None
        if create_action('protest', # analogicznie 'support' w funkcji f_support
                         arg['protest']['action'],
                         arg['protest']['project'],
                         authority,
                         arg['protest']['member']):
            conn.commit()
            print '{"status" : "OK"}'
        else:
            conn.rollback()
            print '{"status" : "ERROR", "debug" : "Nie można dodać akcji"}'
    else:
        conn.rollback()
```

Funkcja f_support działa analogicznie wobec f_protest

3.6 <upvote>/<downvote> w funkcji f_upvote/f_downvote

Opis działania: Podobnie jak w poprzedniej funkcji dodawania protestu/supportu na starcie sprawdzamy członka, który ma zamiar wykonać działanie (w razie potrzeby dodając krotkę członka do bazy danych).
Następne w funkcji **vote** dodawana jest krotka do tabeli votes, która pomaga nam rozróżniać członków, którzy już oddali głos na dany projekt, następnie aktualizowane są wartości w tabelach member(autor akcji) i action. W member - aktualizowane wartości ratio autora akcji (w celu późniejszego szukania trolli), w action - suma głosów za/przeciw (W vote sprawdzamy także czy akcja jest uprzednio zdefiniowana). W przypadku gdy wszystko zakończyło się pomyślnie funkcja zwraca 1.

```
def f_upvote(arg):
    if authorize_or_create_member(arg['upvote']['member'],
                                  arg['upvote']['password'],
                                  arg['upvote']['timestamp']):
        # Oddajemy głos => 1 - upvote, -1 - downvote
        if vote(1, arg['upvote']['member'], arg['upvote']['action']):
            conn.commit()
            print '{"status" : "OK"}'
        else:
            conn.rollback()
            print '{"status" : "ERROR"}'
    else:
        conn.rollback()
```

f_downvote - analogicznie.

3.7 <project> w funkcji f_projects

Opis działania: Sprawdzamy członka (w tym jego uprawienia leadera, hasło i aktywność) po czym zgodnie z opcjonalnymi argumentami funkcji votes wysyłamy zapytanie do bazy danych. Otrzymane krotki formatujemy tak by spełnić wymagania formatu wyjściowego po czym je zwracamy w obiekcie json.

```
def f_projects(arg):
    if authorize_leader(arg['projects']['member'],
                        arg['projects']['password'],
                        arg['projects']['timestamp']):
        try:
            if 'authority' in arg['projects']:
                cur.execute("""SELECT id, authorityID FROM project
                               WHERE authorityID = %s ORDER BY id;""",
                           (arg['projects']['authority'],))
            else:
                cur.execute("SELECT id, authorityID FROM project ORDER BY id;")
            wynik =  cur.fetchall()
            print '{"status" : "OK", "data" : %s}' % format_fetch(wynik)
        except Exception as err:
            print '{"status" : "ERROR",\n "debug" : "%s" }' % str(err)[0:-1]
    else:
        print '{"status" : "ERROR", "debug" : "Błąd leadera"}'
```

3.8 <votes> w funkcji f_votes

Opis działania: Sprawdzamy członka (w tym jego uprawienia leadera, hasło i aktywność) po czym zgodnie z opcjonalnymi argumentami funkcji votes wysyłamy zapytanie do bazy danych. Otrzymane krotki formatujemy tak by spełnić wymagania formatu wyjściowego po czym je zwracamy w obiekcie json.

```
def f_votes(arg):
    if authorize_leader(arg['votes']['member'],
                        arg['votes']['password'],
                        arg['votes']['timestamp']):
        try:
            if 'action' in arg['votes']:
                cur.execute("""SELECT mem.id,
                                      sum(case when value then 1 else 0 end) as votes_for,
                                      count(value) as votes
                               FROM MEMBER as mem LEFT JOIN
                               (SELECT member.id,
                                       vote.value,
                                       action.id as actionid,
                                       project.id as projectid,
                                       project.authorityid as authorityid
                               FROM member
                                   LEFT JOIN vote ON(member.id = vote.memberID)
                                   LEFT JOIN action ON(vote.actionid = action.id)
                                   LEFT JOIN project ON(action.projectid = project.id)
                               WHERE action.id = %s
                               ) as foo ON(mem.id = foo.id)
                               GROUP BY mem.id ORDER BY mem.id;""", (arg['votes']['action'],))
            elif 'project' in arg['votes']:
                cur.execute("""SELECT mem.id,
                                      sum(case when value then 1 else 0 end) as votes_for,
                                      count(value) as votes
                               FROM MEMBER as mem LEFT JOIN
                               (SELECT member.id,
                                       vote.value,
                                       action.id as actionid,
                                       project.id as projectid,
                                       project.authorityid as authorityid
                               FROM member
                                   LEFT JOIN vote ON(member.id = vote.memberID)
                                   LEFT JOIN action ON(vote.actionid = action.id)
                                   LEFT JOIN project ON(action.projectid = project.id)
                               WHERE action.projectid = %s
                               ) as foo ON(mem.id = foo.id)
                               GROUP BY mem.id ORDER BY mem.id;""", (arg['votes']['project'],))
```

```
        else:
            cur.execute("""SELECT mem.id,
                                  sum(case when value then 1 else 0 end) as votes_for,
                                  count(value) as votes
                           FROM MEMBER as mem LEFT JOIN
                           (SELECT member.id,
                                   vote.value,
                                   action.id as actionid,
                                   project.id as projectid,
                                   project.authorityid as authorityid
                           FROM member
                               LEFT JOIN vote ON(member.id = vote.memberID)
                               LEFT JOIN action ON(vote.actionid = action.id)
                               LEFT JOIN project ON(action.projectid = project.id)
                           ) as foo ON(mem.id = foo.id)
                           GROUP BY mem.id ORDER BY mem.id;""")
            wynik = cur.fetchall()
            wynik = map(helper_votes_tuple, wynik)
            print '{"status" : "OK", "data" : %s}' % format_fetch(wynik)
        except Exception as err:
            print '{"status" : "ERROR",\n "debug" : "%s" }' % str(err)[0:-1]
    else:
        print '{"status" : "ERROR", "debug" : "Błąd leadera"}'
```

3.9 <actions> w funkcji f_actions

Opis działania: Sprawdzamy członka (w tym jego uprawienia leadera, hasło i aktywność) po czym zgodnie z opcjonalnymi argumentami funkcji votes wysyłamy zapytanie do bazy danych. Otrzymane krotki formatujemy tak by spełnić wymagania formatu wyjściowego po czym je zwracamy w obiekcie json.

```
def f_actions(arg):
    if authorize_leader(arg['actions']['member'],
                        arg['actions']['password'],
                        arg['actions']['timestamp']):
        try:
            if 'type' in arg['actions']:
                if 'project' in arg['actions']:
                    cur.execute("""SELECT action.id, action.type, action.projectID,
                                          project.authorityID, positive_votes,
                                          negative_votes FROM action
                                   JOIN project ON(action.projectID = project.id)
                                   WHERE action.type = %s AND project.id = %s
                                   ORDER BY action.id;""",
                                (arg['actions']['type'],arg['actions']['project']) )
                elif 'authority' in arg['actions']:
                    cur.execute("""SELECT action.id, action.type, action.projectID,
                                          project.authorityID, positive_votes,
                                          negative_votes FROM action
                                   JOIN project ON(action.projectID = project.id)
                                   WHERE action.type = %s AND project.authorityID = %s
                                   ORDER BY action.id;""",
                                (arg['actions']['type'],arg['actions']['authority']) )
                else:
                    cur.execute("""SELECT action.id, action.type, action.projectID,
                                          project.authorityID, positive_votes,
                                          negative_votes FROM action
                                   JOIN project ON(action.projectID = project.id)
                                   WHERE action.type = %s
                                   ORDER BY action.id;""", (arg['actions']['type'],))
            else:
                if 'project' in arg['actions']:
                    cur.execute("""SELECT action.id, action.type, action.projectID,
                                          project.authorityID, positive_votes,
                                          negative_votes FROM action
                                   JOIN project ON(action.projectID = project.id)
                                   WHERE project.id = %s
                                   ORDER BY action.id;""", (arg['actions']['project'],) )
```

```
            elif 'authority' in arg['actions']:
                cur.execute("""SELECT action.id, action.type, action.projectID,
                                      project.authorityID, positive_votes,
                                      negative_votes FROM action
                                JOIN project ON(action.projectID = project.id)
                                WHERE project.authorityID = %s
                                ORDER BY action.id;""", (arg['actions']['authority'],) )
            else:
                cur.execute("""SELECT action.id, action.type, action.projectID,
                                      project.authorityID, positive_votes,
                                      negative_votes FROM action
                                JOIN project ON(action.projectID = project.id)
                                ORDER BY action.id;""")
        wynik = cur.fetchall()
        print '{"status" : "OK", "data" : %s}' % format_fetch(wynik)
    except Exception as err:
        print '{"status" : "ERROR",\n "debug" : "%s" }' % str(err)[0:-1]
else:
    print '{"status" : "ERROR", "debug" : "Błąd leadera"}'
```

3.10 <trolls> w funkcji f_trolls

Opis działania: Na starcie wywołujemy funkcję wykonującą UPDATE aktywność = false w przypadku gdy timestamp na wejściu jest oddalony o ponad rok od daty ostatniej aktywności członka. Zwraca 1 gdy nie napotkano na żadne błędy bazy danych. Zadajemy zapytanie do bazy danych, po czym formatujemy krotki wynikowe.

```
def f_trolls(arg):
    if skan(arg['trolls']['timestamp']):
        cur.execute("""SELECT id, action_up, action_ratio + action_up, activity FROM member
                       WHERE action_ratio > 0 ORDER BY action_ratio desc, id;""")
        wynik = cur.fetchall()
        print '{"status" : "OK", "data" : %s}' % format_fetch(wynik)
    else:
        print '{"status" : "ERROR", "status" : "błąd bazy danych - update"}'
```