

Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií

Elasticsearch

Martin Schön , Bc.

AIS ID: 103121

E-mail: xschon@stuba.sk

GitHub repozitár: <https://github.com/FIIT-DBS/zadanie-pdt-xSchon>

Predmet: Pokročilé databázové technológie

Zimný semester 2022/2023

Elasticsearch	1
1. Rozbehanie Elasticu	3
2. Vytvorenie indexu	3
3. Vytvorenie mappingu	4
4. Analyzéry	4
a. Englando	4
b. Custom_ngram	5
c. Custom_shingles	6
d. Pridanie mapovania	6
i. Pridanie englando	6
ii. Author analyzéry	6
iii. Lowercase hashtagy	7
5. Bulk import tweetov	7
a) Denormalizácia tweetov SQL	7
b) Python script na nahrávanie do Elasticu	8
6. 5000 tweetov do Elasticu	9
7. Experimenty s nódami	9
8. Upravenie počtu retweetov	10
9. Importovanie všetkých tweetov	11
10. Vyhľadávanie v tweetoch	12
a. Must	12
i. put1n chr1stian fake jew	12
ii. referencies.content	13
iii. hashtag	13
b. Filter	14
c. Should	15
i. person	15
ii. soros	15
iii. words swap	15
d. Agregácie	17
Záver	18

1. Rozbehnutie Elasticu

Zadanie som vypracoval na operačnom systéme Ubuntu 22.04.1 LTS x86_64. Na Linuxe som teda potreboval rozbehať aj tri inštancie Elasticsearchu verzie 8.5.3 - to som dosiahol pomocou dockeru, najskôr je potrebné stiahnuť si docker image a potom je možné rozbehnúť docker, v ktorom beží localhost server na porte 9200. Podľa návodu z oficiálnej [dokumentácie](#) Elasticu sa mi podarilo spustiť elastic a mať k nemu prístup. Elastic beží na troch nódoch - teda tri inštancie, ktoré bežia súčasne pre prípad toho, že by niektorá padla.

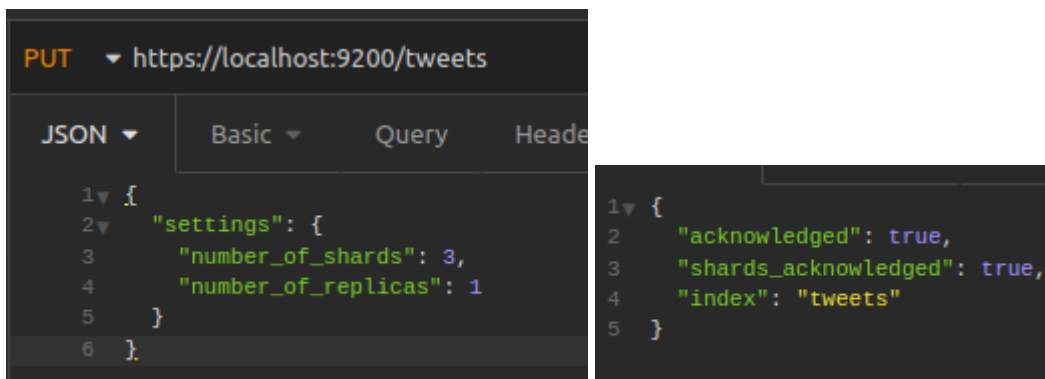
```
9200/tcp, 9300/tcp          z05_es03_1
9200/tcp, 9300/tcp          z05_es02_1
0.0.0.0:9200->9200/tcp, :::9200->9200/tcp, 9300/tcp  z05_es01_1
```

Na Elastic query, teda HTTP requesty používam aplikáciu [Insomnia](#) verzie 3.7.0. Skript na import dát som napísal v jazyku Python 3.10.6.

2. Vytvorenie indexu

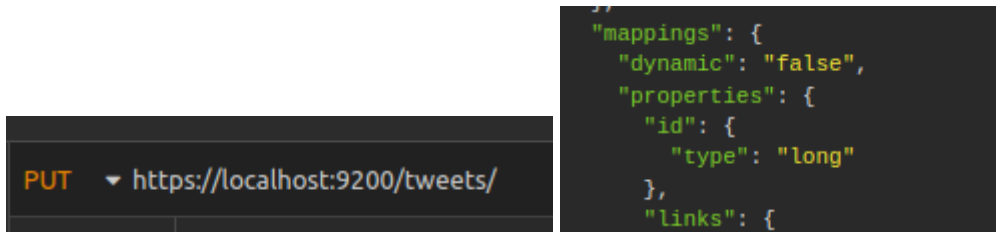
Pri vytváraní indexu tweets je potrebné použiť PUT request zaslaný na adresu Elasticsearch databázy - mojom prípade na localhost:9200. Podľa dokumentácie [elasticu](#) je ideálna veľkosť jedného shardu medzi 10 a 50GB. Vyšší počet shardov umožňuje jednodušší a rýchlejší process obnovy v prípade, že shard spadne. Menšie shardy sú ale pomalšie na vyhľadávanie, preto je potrebné nájsť v nich správny pomer. V mojom prípade očakávam okolo 60 GB dát, preto rozdelím databázu na 3 shardy.

Počet replík som využil základné elastic-u je jedna a to aj používam. Ukladám všetky dáta na jednom disku a teda viac replík by bolo stále uložených na jednom úložisku. Jedna replika mi zároveň šetrí miesto na disku (viac by sa mi už nezmestilo :(). Na produkčnom serveri by som použil repliky 2, aby som zabezpečil funkčnosť vyhľadávania aj v prípade výpadku.

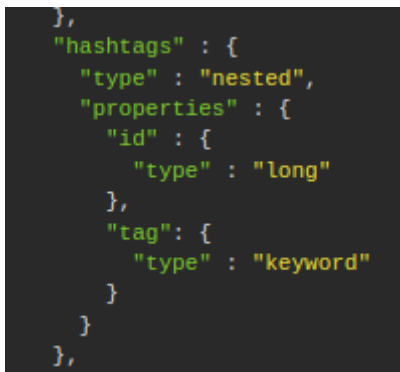


3. Vytvorenie mappingu

V rámci tejto časti zadania som vytvoril mapping nad tweetmi. Ten je možné pridať do tvorby indexu, takže rovno som to tak urobil - ak premažem index, tak naimportujem jeho nastavenia, aj mapping pomocou jedného dopytu. Tento mapping predstavuje štruktúru denormalizovaných dát z PostgreSQL databázy. Všetky informácie o jednej konverzácii sa tak nachádzajú na jednej stránke elasticu. Taktiež je vypnuté dynamické mapovanie, teda mapovanie je explicitné a dáta typu doň musia sedieť, inak budú vynechané.



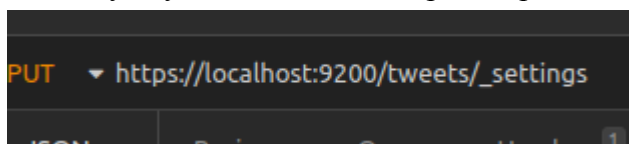
Využil som viaceré dátové typy - základne samozrejme long, text, integer, booleany. Okrem toho je niekoľko políček označených aj ako “keywordy”. Toto označenie je užitočné na vyhľadávanie, napríklad tagu hashtagov, ale použil som ho aj na usernamy, url linky a podobne. Nad keywordom nebude prebiehať full textové vyhľadávanie, to sa vykonáva iba nad textami.



Okrem toho som definoval aj niekoľko premenných typu nested - teda vnorené dáta. Takéto vnorenia zaberajú viac miesta, ale umožňujú vyhľadávanie aj nad vnorenými dátami. Z prečítania si zadania som usúdil, ktoré dáta bude potrebné prehľadávať a ktoré nie a nested som použil pri hashtagoch, anotáciách, doménach, entitách a referenciách.

4. Analyzéry

V prípade, že chcem analyzéry vytvoriť priamo, je možné využiť PUT do `/_settings` na pridanie analyzárov. Predtým je potrebné zastaviť index a následne ho spustiť pre ďalšie časti programu. Takto vzniknuté analyzéry je možné používať, avšak pri použití na mapovanie je výhodné analyzéry rovno definovať a použiť pri nastavovaní indexu.



a. Englando

Ako prvý analyzér je potrebné vytvoriť englando. Englando má za úlohu odstrániť anglické stopwords, dať slová do základného tvaru, odstrániť y a podobne. Je potrebné najskôr definovať stemmery na prácu s anglickým jazykom - [stemmer](#) dokumentácia nám definuje viacero anglicky dostupných stemmerov, ktoré je treba v settingoch zadeklarovať a následne je možné použiť ich v našom novom filtri englando. Podobné platí aj pre anglické

stop slová - ako a, an, the, to,... Následne už len pridáme filtrovanie HTML znakov a štandardný tokenizér.

```
"analyzer": {
  "englando": {
    "filter": [
      "lowercase",
      "english_stemmer",
      "english_possessive_stemmer",
      "english_stop"
    ],
    "char_filter": [
      "html_strip"
    ],
    "type": "custom",
    "tokenizer": "standard"
  },
  "english_stemmer": {
    "type": "stemmer",
    "language": "english"
  },
  "english_possessive_stemmer": {
    "type": "stemmer",
    "language": "possessive_english"
  },
  "english_stop": {
    "type": "stop",
    "ignore_case": true,
    "stopwords": "_english_"
  }
}
```

b. Custom_ngram

Na vytvorenie analyzátoru ngramov je potrebné najskôr definovať ngram filter. Ak chceme ngramy o veľkosti 1 až 10, je treba do settingov pridať riadok

```
"max_ngram_diff": "10",
```

načítka Elastic základne definuje maximálny rozdiel medzi min a max dĺžkou ngramu ako 1. Potom už je možné vytvoriť filter a, podobne ako v predchádzajúcom prípade, spojiť jednotlivé časti do analyzéra. Custom_ngram slúži na dobré vyhľadávanie - má zaindexované ngramy slov, malým textom, bez špeciálnych znakov ASCII.

```
"custom_ngram": {
  "filter": [
    "lowercase",
    "asciifolding",
    "filter_ngrams"
  ],
  "char_filter": [
    "html_strip"
  ],
  "type": "custom",
  "tokenizer": "standard"
},
"filter_ngrams": {
  "type": "ngram",
  "min_gram": "1",
  "max_gram": "10"
},
"filter_shingles": {
  "token_separator": "",
  "type": "shingle"
},
```

c. Custom_shingles

Posledný analyzátor je custom_shingles. Shingle funguje ako “n-gram slov”, teda vytvára dvojice za sebou idúcich slov a porovnáva výsledky. Napríklad “Harry Potter” by sme našli aj pod vyhľadávaním bez medzery - “HarryPotter”. Zvyšné časti analyzátoru sa nelíšia od predchádzajúcich.

```
},
"custom_shingles": {
  "filter": [
    "lowercase",
    "asciifolding",
    "filter_shingles"
  ],
  "char_filter": [
    "html_strip"
  ],
  "type": "custom",
  "tokenizer": "standard"
}
},
"filter_shingles": {
  "token_separator": "",
  "type": "shingle"
},
}
```

d. Pridanie mapovania

Následne som tieto analyzéry pridal do mapovania, podľa inštrukcií zo zadania. Vytváram ich pri tvorbe indexu, vďaka čomu je možné aplikovať ich priamo do mapovania.

i. Pridanie englando

Englando je potrebné pridať do viacerých anglických textových polí. Primárne sa jedná o content conversations a author description. Okrem toho som ho pridal aj do popisu a mena linku, popisu domén a entít a vnoreného description referencovaného tweetu.

```
},
"content": {
  "type": "text",
  "analyzer": "englando"
},
"hashtags": {
```

ii. Author analyzéry

Do polí autora bolo potrebné pridať násobné analyzéry - teda viac na jedno pole. To som vyriešil pomocou “fields”, kedy som jednotlivým parametrom pridal viacero polí, ktoré je možné analyzovať rôznymi prístupmi - každé iným.

```
"name": {
  "type": "text",
  "fields": {
    "with_custom_ngram": {
      "type": "text",
      "analyzer": "custom_ngram"
    },
    "with_custom_shingles": {
      "type": "text",
      "analyzer": "custom_shingles"
    }
  }
}
```

iii. Lowercase hashtagy

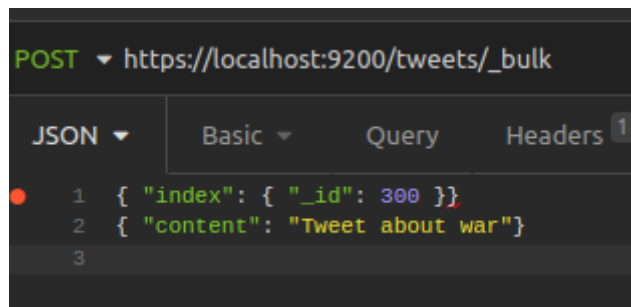
Keďže hashtagy mám vytvorené ako keywordy, lowercase funkcionality zabezpečuje normalizer s filtrom na lowercase.

```
},
"tag": {
  "type": "keyword",
  "normalizer": "lowercase"
}
},
"analysis": {
  "normalizer": {
    "keyword_lowercase": {
      "type": "custom",
      "filter": [
        "lowercase"
      ]
    }
  }
}
```

5. Bulk import tweetov

V tomto bode popisujem aj informácie potrebné pre riešenie úloh 6. a 9.

Na bulkový import tweetov používam bulk API, ktorá je dostupná na `/_bulk` adrese indexu. Je do nej možné poslať jsony, prípadne ich pole a tie sú následne importované do indexu v Elasticu.



```
POST https://localhost:9200/tweets/_bulk

JSON Basic Query Headers 1
1 { "index": { "_id": 300 }}
2 { "content": "Tweet about war" }
3
```

Do databázy chcem poslať tweety uložené v SQL databáze. To znamená, že potrebujem rozbiť štruktúru tabuliek z postgresu a dáta denormalizovať tak, aby pasovali na mapping indexu - respektíve som to robil naopak a najskôr som denormalizoval dáta a podľa toho vytváral mapping.

a) Denormalizácia tweetov SQL

Nakoľko elastic pracuje s jsonmi, potreboval som tabuľky zmeniť na riadky jsonov, kedy jeden riadok predstavuje jednu konverzáciu - s jej autorom, referenciami, hashtagmi, anotáciami aj kontextom. Táto kompletná query je priložená v `denormalization_query.sql` súbore, ale jej časti preberiem aj tu.

O každej konverzácii vyberiem všetky dostupné dáta z jej poľa, tak ako pri bežnom SQL vyhľadávaní. Následne potrebujem namapovať jednotlivé tabuľky. V prípade autorov ide o pomerne jednoduchú operáciu - stačí mi vybrať autora ako json, keďže každý tweet má jedného autora.

```
LEFT JOIN LATERAL (
  SELECT to_jsonb(authors) AS authors
  FROM authors
  WHERE authors.id = c.author_id
) authors ON true
```

Na viacerých miestach používam LEFT JOIN LATERAL, nakoľko ten mi umožňuje vnútri joinu používať filter na základe informácie z vonku - v tomto prípade c.author_id, kde c predstavuje hlavnú konverzáciu na ktorú mapujem. Všetky takto získané dáta konvertujem do jsonov - v tomto prípade do čistého jsonu, pomocou funkcie to_jsonb.

Pri iných prípadoch, napríklad pri hashtagoch používam funkciu json_agg. Tá mi umožňuje vytvoriť pole jsonov všetkých použitých hashtagov. Táto tabuľka je špeciálna aj tým, že je to many to many vzťah, kedy conversation_hashtags predstavuje pomocnú tabuľku, vďaka ktorej dostanem všetky hashtagy do jsonu hashtags ktorý patrí danej c.id. Nakoľko tweet *nemusí* mať hashtag, je možné pridať aj COALESCE funkciu, ktorá None nahradí prázdny polom. Túto štandardne nevyužívam, nakoľko Elastic sa s none popasuje dobre a vynechá danú hodnotu.

```
LEFT JOIN LATERAL (
  SELECT json_agg(hashtags) AS hashtags
  FROM hashtags
  LEFT JOIN conversation_hashtags ch on hashtags.id = ch.hashtag_id
  WHERE c.id = ch.conversation_id
) hashtags ON true
```

Podobným štýlom teda dostanem všetky údaje o konverzácii do jedného riadku, ktorý obsahuje dáta a jsony. Z toho celého následne vytvorím jeden json riadok pre jeden tweet- opäť za pomoci funkcie to_jsonb.

```
SELECT to_jsonb(cnvs) AS conversations
FROM ( --data about conversation
```

Pre účely tohto importu odporúčam vytvoriť si aj indexy, ktoré sú napísané v priloženom súbore indexes.sql. Tieto indexy pomáhajú násobne zrýchliť vytváranie jsonu a vďaka tomu je možné, aby zbehli v reálnom čase.

b) Python script na nahrávanie do Elasticu

Query z bodu a) využívam v Python scripte SQL_to_elastic.py, ktorý slúži na stiahnutie dát z databázy postgresu a ich následné zaslanie do Elasticsearchu. Používa na to knižnice elasticsearch a pycpg2, kedy načíta jsony z postgresu a (po miernom spracovaní) ich pošle na bulk API Elasticu. Tento skript je tiež priložený v súboroch a využívam ho v bode 6. aj 9.

6. 5000 tweetov do Elasticu

Pomocou postupu popísanom v bode [Bulk import tweetov](#) dostanem do databázy 5000 záznamov, ktoré si viem skontrolovať aj v bulk POST query, kde nájdem všetky dokumenty.

```
POST https://localhost:9200/tweets/_search?size=100

JSON Basic Query Headers 1

1 {
2   "query": {
3     "match_all": {}
4   }
5 }
6

{
  "hits": {
    "total": {
      "value": 5000,
      "relation": "eq"
    },
    "max_score": 1.0
  }
}
```

	health	status	index	uuid	pri	rep	docs.count	docs.deleted	store.size	pri.store.size
1	green	open	news	MHyvQLFpQP006-QYrf1fyA	2	2	0	0	1.3kb	450b
2	green	open	test	vGedCEfYRpS2tkCe1gx-iA	2	2	0	0	1.3kb	450b
3	green	open	tweets	IPONe0zQTFG7o5VRM5BENG	3	1	5000	0	5.5mb	2.7mb

7. Experimenty s nódami

Každá inštancia elasticu vytvára takzvaný nód. Na začiatku mám spustené tri lokálne nódy, pričom node 01 je nastavený ako master - teda hlavný.

ip	heap.percent	ram.percent	cpu	load_1m	load_5m	load_15m	node.role	master	name
172.25.0.3	55	100	7	0.91	0.97	1.07	cdfhilmrstw *	*	es01
172.25.0.4	49	100	7	0.91	0.97	1.07	cdfhilmrstw -	-	es02
172.25.0.5	79	100	7	0.91	0.97	1.07	cdfhilmrstw -	-	es03

Ak vypnem docker es01, tak sa zvyšné dva nódy zostanú bežať a funkcionality zostane stále rovnaká - dokážem robiť všetky operácie. V mojom prípade je teraz master shard es03, teda dohodli sa na ňom.

172.25.0.5	53	100	10	2.50	1.30	0.89	cdfhilmrstw *	*	es03
172.25.0.4	49	100	10	2.50	1.30	0.89	cdfhilmrstw -	-	es02

Akonáhle by som vypol aj ten, es02 už nie je možné samostatne nastaviť ako master - pretože mám nastavené electovanie mastera - samostatne jeden node nedokáže sám seba zvoliť. To je možné vypnúť, avšak nie je to odporúčané, nakoľko pri prirodzenom výpadku môžu nastať aj iné chyby, ako len vypadnutie dockera - napríklad môže vypadnúť komunikácia medzi nódmi a v tom prípade by viacnásobný master spôsoboval problémy. Keďže som ostal bez mastera, nie je možné vkladať a mazať záznamy v indexoch, keďže by to mohlo spôsobiť problémy.

Je možné, aby aplikácia bežala aj na jednom nóde - napríklad ak vypnem hlasovanie o master nóde, alebo ak vytvorím cluster len s jedným nódom. Toto riešenie ale nie je ideálne, pretože v prípade výpadku mám aplikáciu nefunkčnú.

8. Upravenie počtu retweetov

Vybral som si konverzáciu, ktorá ma nulový počet retweetov a vytvoril som jednoduchý skript na úpravu. Skript funguje v POSTe a na `_update/_id` dokumentu pošle `retweet_count++`, čím sa retweet count zvyšuje o jedna.

POST `https://localhost:9200/tweets/_update/QM2YOoUBs-CE6nDg2TbU`

```
{
  "script": {
    "source": "ctx._source.retweet_count++"
  }
}
```

V returnovej hodnote potom vidím `_seq_no` aj `_primary_term` hodnoty. **primary_term** slúži na uloženie informácií o tom, koľkokrát sa zmenil primárny shard. Zakaždým, keď sa zmení, tak `primary_term` stúpne o jedna. `_seq_no` potom počíta, koľká operácia prebehla na indexe.

```
{
  "_seq_no": 5006,
  "_primary_term": 1
}
```

V mojom prípade je teda `primary_term` 1 a `seq_no` 5006 - po predchádzajúcej úlohe som nanovo prebudoval index, takže mám 5000 vkladových operácií, preto som na takom vysokom čísle. `_primary_term` je 1, nakoľko som reštartoval docker.

Potom, čo som zabil node, na ktorom bol tweets shard a znova ho spustil, zvýšil sa mi `primary_term`, keďže primárny shard bol zmenený.

tweets 0 p STARTED 5000 8.4mb 172.25.0.4 es02

```
{
  "_seq_no": 5012,
  "_primary_term": 2
}
```

`_seq_no` je zmenené, ak dropnem index a znova ho vytvorím - pôjdem opäť od nuly

9. Importovanie všetkých tweetov

Popis a skript, ktorý používam v tomto bode je podrobnejšie popísaný [vyššie](#). Do databázy pošlem všetkých 32miliónov konverzácií, postupne v chunkoch po 250 tisíc. Táto hodnota sa dá ľahko upraviť v skripte.

Predtým som vytvoril index nanovo, tento krát s 0 počtom replík:

```
{
  "settings": {
    "number_of_shards": 3,
    "number_of_replicas": 0,
    "max_perm_diff": "15"
```

Do Elasticsearchu som tak dostal všetky tweety a ich vlastnosti.

10. Vyhľadávanie v tweetoch

a. Must

i. *putln christian fake jew*

V tejto časti som spravil zloženú query, ktorá obsahuje viacero častí. V prvej časti hľadám zhodu medzi contentom a `putln christian fake jew`, v druhej tento text hľadám v autoroch - konkrétne `authors.description.with_custom_shingles`.

Bolo by možné použiť aj `multi_match` s operátorom OR, ale ja mám autorov uložených ako nested (kvôli ďalšej úlohe, b). `Multi_match` by fungoval, ak by som do tvorby indexu pridal `"include_in_root": true`, avšak na to by som musel preindexovať celé zadanie. Moje riešenie s nested funguje - dve časti spájam pomocou `should` funkcionality.

```
1 {
2   "query": {
3     "bool": {
4       "should": [
5         {
6           "function_score": {
7             "functions": [
8               {
9                 "weight": 10,
10                "filter": {
11                  "match": {
12                    "content": {
13                      "query": "putln christian fake jew",
14                      "fuzziness": "AUTO"
15                    }
16                  }
17                }
18              }
19            ]
20          }
21        },
22        {
23          "nested": {
24            "path": "authors",
25            "query": {
26              "function_score": {
27                "functions": [
28                  {
29                    "weight": 6,
30                    "filter": {
31                      "match": {
32                        "authors.description.with_custom_shingles": {
33                          "query": "putln christian fake jew",
34                          "fuzziness": "AUTO"
35                        }
36                      }
37                    }
38                  }
39                ]
40              }
41            }
42          }
43        }
44      ]
45    }
46  },
47  {
48    "nested": {
49      "path": "authors",
50      "query": {
51        "function_score": {
52          "functions": [
53            {
54              "weight": 6,
55              "filter": {
56                "match": {
57                  "authors.description.with_custom_shingles": {
58                    "query": "putln christian fake jew",
59                    "fuzziness": "AUTO"
60                  }
61                }
62              }
63            }
64          ]
65        }
66      }
67    }
68  }
69  ],
70  "minimum_should_match": 1
71 }
```

Pomocou `function_score` a `functions` nastavím váhu (10 a 6) obom prístupom. Na detekciu preklepov používam `fuzziness` - funkcia Elasticu, ktorá sleduje vzdialenosť medzi písmenami na klávesnici a napísanými preklepmi. Použitá je hodnota `AUTO`, ktorá sa dynamicky prispôsobuje textu a nastaví maximálny možný počet preklepov podľa napísaného textu.

ii. *referencies.content*

Na nájdenie referencií opäť používam nested query, ktorá jednoducho matchuje referencie konverzácií obsahujúce nazi.

```
1▼ {
2▼   "query": {
3▼     "bool": {
4▼       "must": [
5▼         {
6▼           "nested": {
7▼             "path": "conversation_references",
8▼             "query": {
9▼               "bool": {
10▼                "must": [
11▼                  {
12▼                    "match": {
13▼                      "conversation_references.content": "nazi"
14▼                    }
15▼                  }
16▼                ]
17▼              }
18▼            }
19▼          }
20▼        ]
21▼      }
22▼    }
23▼  }
24▼}
```

iii. *hashtag*

Hashtag tagy fungujú rovnako ako convent referencií, s tým rozdielom, že hashtagy sú zaindexované ako lowercase, takže aj Ukraine hashtag je validná návratová hodnota.

```
1▼ {
2▼   "query": {
3▼     "bool": {
4▼       "must": [
5▼         {
6▼           "nested": {
7▼             "path": "hashtags",
8▼             "query": {
9▼               "bool": {
10▼                "must": [
11▼                  {
12▼                    "match": {
13▼                      "hashtags.tag": "ukraine"
14▼                    }

```

```

[\\n\\nThe eminent surround
  "hashtags": [
    {
      "id": 9,
      "tag": "Ukraine"
    },

```

Tieto tri časti nakoniec spájam do jednej MUST podmienky, ktorá zabezpečuje, že všetky tri časti musia platiť.

```
{
  "query": {
    "bool": {
      "must": [
        {
```

b. Filter

```
1 {
2   "query": {
3     "bool": {
4       "filter": [
5         {
6           "nested": {
7             "path": "authors",
8             "query": {
9               "bool": {
10                "must": [
11                  {
12                    "range": {
13                      "authors.following_count": {
14                        "gt": 100
15                      }
16                    }
17                  },
18                  {
19                    "range": {
20                      "authors.followers_count": {
21                        "gt": 100
22                      }
23                    }
24                  }
25                ]
26              }
27            }
28          },
29          {
30            "exists": {
31              "field": "links"
32            }
33          }
34        ]
35      }
36    }
37  }
```

Filter sa skladá z 3 častí - prvá a druhá sa týkajú nested autorov, preto tieto podmienky môžem v jednej nested query vyriešiť obe - pomocou range skontrolujem following aj followers_count autora a nastavím ich na greater than (gt) 100.

Tretí filter rieši existenciu vnoreného links jsonu - funkcia exists skontroluje pole a vráti len tie záznamy, ktoré obsahujú validnú hodnotu na danom mieste (teda odfiltruje nulls - môj prípad, ale aj prázdne polia [], ["foo"], [null] a neexistujúce polia).

Tento dopyt mi teda vráti len tie záznamy, ktoré majú aspoň jeden link a ich autor má viac ako 100 followerov a zároveň followuje viac ako 100 ľudí.

c. Should

- i. *person*
- ii. *soros*

Person aj Soros hľadanie prebieha podobným štýlom - v nested query vytiahnem Person, alebo Soros a pridelím mu hodnotu 5 v prípade, že query splňa podmienku.

```
{
  "query": {
    "bool": {
      "should": [
        {
          "nested": {
            "path": "context_domains",
            "query": {
              "function_score": {
                "functions": [
                  {
                    "weight": 5,
                    "filter": {
                      "match": {
                        "context_domains.name": "Person"
                      }
                    }
                  }
                ]
              }
            }
          }
        ]
      }
    }
  }
}
```

Pri hľadaní Sorosa som sa stretol s problémom, že mám entities.name zaindexované ako keywordy - nie ako text. Preto som musel upraviť Soros na George Soros, aby som dostal exaktný match. Alternatíva je preindexovať celé zadanie a entities.name zmeniť na text - potom by query fungovala aj so Sorosom.

```
    "weight": 10,
    "filter": {
      "match": {
        "context_entities.name": "George Soros"
      }
    }
  }
}
```

iii. *words swap*

V prípade putin christian fake jew fráze, v ktorej môže prebehnúť výmena slov - ako prvé matchujem frázu, nie slová. Na výmenu slov následne slúži funkcia slop 1, ktorá umožňuje aby práve dve slová (jedna výmena) bola stále braná ako validný return.

```
    "function_score": {
      "functions": [
        {
          "weight": 5,
          "filter": {
            "match_phrase": {
              "content": {
                "query": "putin christian fake jew",
                "slop": 1
              }
            }
          }
        }
      ]
    }
  }
}
```

Na záver som spojil úlohy A, B a C dokopy a získal som tak jednu query, ktorá je súčasťou odovzdania pod menom A+B+C FULL QUERY.

```
1 {
2   "query": {
3     "bool": {
4       "must": [ ↩ 3 ↩ ],
83      "should": [ ↩ 2 ↩ ],
123     "filter": [ ↩ 2 ↩ ]
155   }
156 }
157 }
```

Príklad výstupu:

```
5   "max_score": 26.609734,
6   "hits": [
7     {
8       "_index": "tweets",
9       "_id": "FMYiPYUBWf7NPixNiAnb",
10      "_score": 26.609734,
11      "_source": {
12        "id": 1497633460847976456,
13        "links": [
14          {
15            "id": 2498006,
16            "url": "https://twitter.com/JaRaNo1128/status/1497625277794508806/photo/1",
17            "title": null,
18            "description": null,
19            "conversation_id": 1497633460847976456
20          }
21        ],
22        "source": "Twitter for iPhone",
23        "authors": {
24          "id": 4085092173,
25          "name": "Carlos Watemberg",
26          "username": "cwatemala1",
27          "description": "🇺🇸 Ardent Zionist, Proud Jew. Love what USA stands for! 🇺🇸 Anti-woke 100%. Pro
husband and father of 3 (plus the 4 legged).",
28          "tweet_count": 5404,
29          "listed_count": 0,
30          "followers_count": 150,
31          "following_count": 681
32        },
33        "content": "RT @JaRaNo1128: When #Putin says he is trying to defeat 'neo-Nazis' 🇺🇸. #Ukraine
#UkraineUnderAttack https://t.co/kf69JZH8q9",
34        "hashtags": [
```


d. Agregácie

V tejto úlohe som najskôr vybral všetky hashtagy - pomocou `hashtags.tag` a funkcie `terms`, ktorá umožňuje nachádzanie dokumentov na základe zhody voči poľu - dostanem tak tie dokumenty, ktoré obsahujú aspoň jeden z daných hashtagov.

Následne takto vybrané dokumenty pospájam to intervalov na základe agregácie podľa dátumov. Na to používam funkciu `agg` s `date_histogram`, kde je možné nastaviť interval na 7d, teda na jeden týždenné buckety. Túto query, podobne ako predchádzajúce vyhľadávania, posielam na adresu https://localhost:9200/tweets/_search, avšak pridám `?size=0`, aby som dostal rovno výsledky bucketov.

```
GET https://localhost:9200/tweets/_search?size=0
```


```
1 {
2   "query": {
3     "bool": {
4       "must": [
5         {
6           "nested": {
7             "path": "hashtags",
8             "query": {
9               "bool": {
10                "must": [
11                  {
12                    "terms": {
13                      "hashtags.tag": ["istandwithputin", "racism", "itrillion", "istandwithrussia",
14                      "isupportrussia", "blacklivesmatter", "racism", "racistukraine", "africansinukraine", "palestine", "israel",
15                      "freepalestine", "istandwithpalestine", "racisteu", "putin"]
16                    }
17                  }
18                ]
19              }
20            }
21          }
22        }
23      ]
24    }
25  }
```

Čo sa týka výstupu, formát je nasledovný:

```
{
  "key_as_string": "2022-01-27T00:00:00.000Z",
  "key": 1643241600000,
  "doc_count": 86
},
{
  "key_as_string": "2022-02-03T00:00:00.000Z",
  "key": 1643846400000,
  "doc_count": 103
},
```

Nachádzajúca sa v ňom všetky týždne podľa kľúčov. Hodnoty `doc_count`ov nebudú všetky sedieť, pretože som si premazával index a až potom som zistil, že nemám dobre zaznamenané výstupy z tejto časti. Nemal som už čas importovať všetky dáta, takže výsledky v priloženom súbore reprezentujú asi len 15 miliónov dokumentov (z 32000000 záznamov).

Záver

Zadanie som vypracoval samostatne, opísané sú moje vedomosti a poznatky. V prílohe sa nachádza .json súbor, ktorý je možné otvoriť v insomnia verzii 4 a pozrieť si všetky queries podrobnejšie. Taktiež obsahuje sql súbory a python script spomínaný v tomto dokumente - priečinku migrate-data. Všetky tieto súbory je možné nájsť aj na [GitHube](#) uvedenom na titulnej strane. Ďakujem za čítanie dokumentácie mojej práce a prajem pekné prežitie Vianočných sviatkov .