

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE

Fakulta informatiky a informačných technológií

PDT

Zadanie 1 - import databázy

2021/2022

1. Opis algoritmu

1.1 Vytvorenie tabuliek

1.2 Import autorov

1.3 Import konverzácií

1.4 Import referencií

1.5 CSV export

2. Použité technológie

Ruby

Sequel

Dotenv

Zlib

3. Popis SQL

4. Časový opis priebehu a dĺžka importu

5. Počet a veľkosť záznamov v tabuľkách

Annotations

Authors

Context_annotations

Context_domains

Context_entities

Conversation_hashtags

Conversation references

Conversations

Hashtags

Links

1. Opis algoritmu

Program funguje v štyroch hlavných častiach, ktoré sa starajú o vytvorenie tabuliek, import autorov, import konverzácií a import referencií. Program je rozdelený na tieto 4 hlavné časti, pretože je dôležité poradie ich vykonávania. Všetky 3 import-y majú podobný algoritmus len vo väčšej, alebo menšej miere.

1.1 Vytvorenie tabuliek

Prvé sa v programe logicky vytvárajú tabuľky. Pomocou Sequel knižnice sa nad databázou volá metóda `database.create_table?(NAZOV_TABULKY)`.

```

database.create_table? :links do
  primary_key :id, type: :Bignum, null: false
  foreign_key :conversation_id, :conversations, type: :Bignum, null: false
  String :url, size: 2048, null: false
  String :title, text: true
  String :description, text: true
end

```

Vďaka tejto metóde sa zistí, či už daná tabuľka existuje. Ak áno, tak sa nič nestane, ak nie, tak sa táto tabuľka vytvorí. Takto sa prejdú všetky tabuľky z danej schémy.

1.2 Import autorov

Prvá vec ktorá sa importuje sú autori, keďže neobsahujú žiadnu referenciu. Import prebieha v súbore *authors_import.rb*. Na začiatku sa vytvoria potrebné premenné.

```

batch_time = Time.now
batch_size = 100000
batch_number = 0
filepath = 'authors.jsonl.gz'
rows = 0

array_of_authors = []

```

Ukladá sa začiatok spracovania batch-u, veľkosť batch-u, číslo batch-u, cestu k súboru, počet riadkov a takisto pole autorov. Pole autorov slúži ako miesto kde sa odkladajú riadky autorov, ktoré sa následne vložia do databázy.

Ďalej nasleduje spracovanie samotného súboru.

```

Zlib::GzipReader.zcat(File.open(filepath)) do |line|      # open stream of unzipped rows
  parsed_line = JSON.parse(line.gsub('\u0000', ''))      # replace null byte and parse string to Hash

  array_of_authors << {id: parsed_line['id'],             # add row to array
    name: parsed_line['name'],
    username: parsed_line['username'],
    description: parsed_line['description'],
    followers_count: parsed_line.dig('public_metrics', 'followers_count'),
    following_count: parsed_line.dig('public_metrics', 'following_count'),
    tweet_count: parsed_line.dig('public_metrics', 'tweet_count'),
    listed_count: parsed_line.dig('public_metrics', 'listed_count')}

  rows += 1
end

```

Na čítanie zo súboru zazipovaného pomocou GZip-u sa používa Zlib knižnica ktorá má v sebe triedu GzipReader. Táto používa funkciu zcat, pretože pri používaní klasickej .read metódy sa do RAM načítal celý súbor a pri použití .open metódy a následného čítania v Ruby sa objavoval bug, že sa súbor neprečítal celý. Metóda zcat vracia celý súbor po

jednotlivých riadkoch. Načítaný riadok pošle ďalej do bloku ako premennú line. Predtým než je tento riadok spracovaný, tak sa všetky null bytes nahradia prázdny stringom.

Nasleduje konverzia riadku, ktorý je reprezentovaný ako String, na dátovú štruktúru Hash.

Hash (<https://ruby-doc.org/core-3.1.2/Hash.html>) je dátová štruktúra ktorá si drží veľmi podobne ako JSON alebo napríklad dictionary v Pythone, teda, že má nejaký kľúč a k tomu priradenú hodnotu. Vďaka tomu sa dá pristupovať k jednotlivým častiam daného záznamu priamo pomocou ich kľúča, cez `parsed_line['id']`, kde `parsed_line` je názov Hash objektu a v úvodzovkách je kľúč, ktorý vráti svoju hodnotu. Takto sa získajú všetky potrebné údaje, ktoré sa volžia do nového Hash, reprezentovaného kučeravými zátvorkami a s kľúčom, rovnakými pre hodnoty ktoré sa budú vkladať do tabuľky. Pole `array_of_authors`, tak v sebe drží 100 000 Hash-ov pre každý záznam v tabuľke. Na konci spracovania daného riadku sa ešte zvýši premenná `rows`, ktorá v sebe drží počet spracovaných riadkov.

```
if rows % batch_size == 0          # if rows are in batch size
  # insert array to DB with INSERT INTO command
  DATABASE[:authors].insert_conflict(:target=>:id).multi_insert(array_of_authors)
  puts "#{batch_number} - #{array_of_authors.size}"
  array_of_authors.clear           # free array
  batch_number += 1               # increment batch_number

  # add new times to csv_array
  csv_array << [Time.now.strftime('%Y-%m-%dT%H:%M%z'), seconds_to_hms(Time.now - start_time), seconds_to_hms(Time.now - batch_time)]

  # update new batch_time
  batch_time = Time.now
end
end

# insert array to DB with INSERT INTO command
DATABASE[:authors].insert_conflict.multi_insert(array_of_authors)

# add new times to csv_array
csv_array << [Time.now.strftime('%Y-%m-%dT%H:%M%z'), seconds_to_hms(Time.now - start_time), seconds_to_hms(Time.now - batch_time)]

# clear array and free RAM
array_of_authors.clear
puts "End je: #{Time.now}"
```

V tretej časti sa sa kontroluje či počet riadkov je deliteľný `batch_size`. Ak je, tak cyklus vstúpi do podmienky, kde sa pole dát o autoroch vkladá do databázy. Používa sa volanie dvoch metód `insert_conflict` a `multi_insert`. Obidve sú získané z knižnice `sequel` ktorá je zodpovedná za komunikáciu s databázou. Prvá, teda `insert_conflict` zaobahuje ORM volanie pre upsert. Vďaka ORM vygeneruje `ON CONFLICT DO NOTHING insert`, vďaka čomu, keď sa vkladá záznam s ID, ktoré už existuje, tak sa proste preskočí. Druhá metóda `multi_insert` vygeneruje z ORM jeden veľký `INSERT INTO ... VALUES ()` command, vďaka čomu sa 100000 záznamov nevkladá zakaždým po jednom `INSERT-e`, ale naraz.

Následne po vložení sa zapíšu tri údaje do poľa `csv_array`, ktoré metóda dostala ako parameter, a to sú: aktuálny čas, čas od začiatku behu programu a čas spracovania momentálneho batch-u. Nasleduje kontrolný výpis a potom sa pole autorov vyčistí, aby sa mohlo znova naplniť a v ďalšej iterácii znova mohlo byť vložené.

Takto sa to deje až do konca čítania zo súboru, s tým, že posledný batch sa už nedostane do tej if podmienky a preto sa to isté vykoná ešte raz po konci čítania do súboru, kde sa sa vloží posledný batch a vyčistia sa premenné, aby nezaberali miesto v pamäti.

Tento algoritmus sa v upravených verziach opakuje pri všetkých troch insertoch, ako bolo spomenuté vyššie, preto pri iných nebude takto do hĺbky rozpísaný.

1.3 Import konverzácií

Algoritmus pri konverzáciách je rozšírený o to, že naraz naplní 9 tabuliek. Okrem tabuľky conversations, sa naplňajú aj null autori (teda autori, ktorých IDčko sa nenachádza v tabuľke authors ale konverzácia sa na nich odkazuje, preto sa do tejto tabuľky vloží záznam, ktorí má ID, ale ostatné všetky riadky sú null), taktiež aj links, hashtags, conversation_hashtags, annotations, context_domains, context_entity a context_annotations. Vďaka tomu aj táto časť programu beží najdlhšie.

```
array_of_conversations << {id: parsed_line['id'], # Insert new conversation row to array
  author_id: parsed_line['author_id'],
  content: parsed_line['text'],
  possible_sensitive: parsed_line['possibly_sensitive'],
  language: parsed_line['lang'],
  source: parsed_line['source'],
  retweet_count: parsed_line.dig('public_metrics', 'retweet_count'),
  reply_count: parsed_line.dig('public_metrics', 'reply_count'),
  like_count: parsed_line.dig('public_metrics', 'like_count'),
  quote_count: parsed_line.dig('public_metrics', 'quote_count'),
  created_at: parsed_line['created_at']}

if parsed_line.dig('entities', 'urls') # If conversation has urls in entities
  parsed_line.dig('entities', 'urls').each do |url| # for each url in entities take one as url
    next if url['expanded_url'].size > 2048 # skip this url if length is longer than 2048
    links << {conversation_id: parsed_line['id'],
      url: url['expanded_url'],
      title: url['title'],
      description: url['description']}
  end
end
```

Konverzácie sa naplňujú úplne rovnako ako autori, samozrejme s inými poľami. Potom nasleduje naplňovanie poľa, ktoré sa vkladá tabuľky links. Toto sa najprv pozrie, že či daný Hash má v sebe nejaké urls. To sa robí cez metódu *dig* ktorá pozrie či daná hodnota v Hash-y je. Ak áno, tak sa tieto hodnoty začnú prechádzať a potom sa rovnakým spôsobom zapisujú do poľa, ktoré sa pri naplnení batch_size uloží do databázy. Výnimkou je, že sa kontroluje najprv veľkosť linku či neprekračuje maximálnu veľkosť poľa. Takto sa postupne prechádzajú všetky časti jedného záznamu, ktoré nás zaujímajú a potom sa vložia do databázy. Pri vkladaní do tabuliek hashtags, context_entities a context_domains sa najprv skontroluje či už neexistuje záznam s daným ID, alebo tag-om. Za týmto účelom si program drží vložené IDčka v dátovej štruktúre Set (pri hashtag-och sa používa Hash, pretože sa pod daným tag-om drží jeho ID, teda napr. {'Ukraine': 31, 'Russia': 12, ...}), keďže nás zaujíma iba, že či sa dané ID nenachádza v databáze, nepotrebujeme si držať nič iné len samé ID. Nedržíme ich však v poli, pretože prehľadávať niekoľko miliónové pole je časovo veľmi náročné. Preto sa používa Set, ktorý je zjednodušená verzia Hash-u, a má v podstate len kľúče bez hodnôt. Na rozdiel od poľa v ňom však vyhľadávanie nemá časovú komplexitu $O(N)$, ale má len $O(1)$, rovnako ako v Hash-y. Toto značne urýchľuje beh programu. Pri hashtag-och je ešte zaujímavé, že hashtagy nemajú priradené ID, ale závisí na nich cudzí kľúč v tabuľke conversation_references. Preto sa do tabuľky hashtags ukladá záznam aj s ID, ktoré je na začiatku 1. Každému novému hashtag-u sa pred pridaním vloží toto ID, ktoré si držíme v premennej a následne sa incrementuje. Tag hashtag-u sa potom spolu s ID vloží

do Hash-u a pri ďalšom vložení sa toto ID načíta pomocou tag-u, bez ďalšieho zbytočného dopytu na databázu.

Vkladanie do databázy vyzerá nasledovne:

```
# Insert all arrays to DB on one INSERT INTO statement
DATABASE[:authors].multi_insert(array_of_null_authors)
DATABASE[:conversations].multi_insert(array_of_conversations)
DATABASE[:links].multi_insert(links)
DATABASE[:hashtags].multi_insert(new_hashtags)
DATABASE[:conversation_hashtags].multi_insert(conversation_hashtags)
DATABASE[:annotations].multi_insert(annotations)
DATABASE[:context_domains].multi_insert(context_domain)
DATABASE[:context_entities].multi_insert(context_entity)
DATABASE[:context_annotations].multi_insert(context_annotations)

# clear all arrays
puts "#{batch_number} - #{array_of_conversations.size}"
array_of_conversations.clear
array_of_null_authors.clear
links.clear
annotations.clear
new_hashtags.clear
conversation_hashtags.clear
context_domain.clear
context_entity.clear
context_annotations.clear
```

Už sa nikde nepoužíva upsert, pretože sa priamo v kóde kontroluje, aby nemal žiadny záznam duplicitné ID.

1.4 Import referencií

Pri importe referencií, teda záznamov do tabuľky conversation_references sa postupuje takmer takisto veľmi podobne. Vytvoria sa základné premenné, pole, do ktorého sa ukladajú existujúce referencie, aby sa nevložili dvakrát duplicitné hodnoty, s tým, že sa načítajú, opäť do Set-u všetky IDčka konverzácií, aby sa referencie vkladali len pre záznamy, ktoré majú existujúcu rodičovskú referenciu. Program potom prechádza znova všetky riadky z conversations.jsonl.gz súboru, ak daný záznam má nejakú referenciu na existujúci conversations záznam, tak ho vlož do poľa, ktoré sa potom pri naplnení batch_size vloží do databázy. Zapiš časy pre CSV export, vyčisti pole a znova opakuj, pokiaľ nie sú všetky spravené. Toto opakuj pre celý súbor.

```

Zlib::GzipReader.zcat(File.open(filepath)) do |line|      # open stream
  parsed_line = JSON.parse(line.gsub('\u0000', ''))      # replace null
  next if existing_references.include?(parsed_line['id'].to_i)  # if c

  if parsed_line.key?('referenced_tweets')              # if tweet has
    parsed_line['referenced_tweets'].each do |tweet|    # for each ref
      next unless existing_conversations.include?(tweet['id'].to_i)
      referenced_tweets << {conversation_id: parsed_line['id'],
                            parent_id: tweet['id'],
                            type: tweet['type']}

      existing_references << parsed_line['id'].to_i      # update exist
    end
  end

  rows += 1
end

```

1.5 CSV export

Na záver programu sa vytvorí nový CSV súbor, ktorý sa naplní hodnotami z csv_array, ktoré reprezentujú časy pre jednotlivé batch import-y pri všetkých troch importoch.

```

# block which writes, all lines from csv_array to csv file
CSV.open( filename "pdt_adam-zak_v2.csv", mode "w") do | IO csv|
  csv_array.each do | Elem time|
    csv << time
  end
end

```

2. Použité technológie

1. Ruby

Zadanie som naprogramoval v programovacom jazyku Ruby. Tento jazyk som si zvolil, pretože, okrem iného s ním pracujem, takže sa v ňom celkom dobre vyznám (škoda, že sa to neráta ako dôvod :(). Ale hlavne pretože si myslím, že vďaka tomu, že sa primárne využíva na web development (najmä s Rails frameworkom) a vďaka tomu má veľa výborných knižníc a funkcionalít na prácu s databázami. Jednou z nich je aj Sequel knižnica na prácu s databázou.

2. Sequel

Pôvodne som zamýšľal používať klasickú knižnicu na prácu s databázami *activerecord* (<https://rubygems.org/gems/activerecord/versions/5.0.0.1>), ktorá je integrovaná v Rails frameworku a veľmi známa. Práve vďaka tomu spojeniu však obsahuje obrovské množstvo funkcionalít, ktoré pre toto zadanie nie sú potrebné a zbytočne by zväčšovali veľkosť programu a spomaľovali jeho beh. Preto som zvolil menší gem (knižnicu) s názvom *Sequel* (<https://github.com/jeremyevans/sequel> | <https://sequel.jeremyevans.net/>). Táto knižnica je menšia a ľahšia, vďaka tomu vykonáva svoju úlohu rýchlejšie a využíva oveľa menej zdrojov.

3. Dotenv

Táto knižnica slúži na načítanie premenných prostredia (env variables), na prístup k databáze.

```
Dotenv.load      # Load all env variables

DATABASE = Sequel.connect(adapter: :postgres,          # options for database connections
                           user: ENV.fetch("POSTGRES_USER"),
                           password: ENV.fetch("POSTGRES_PASSWORD"),
                           host: ENV.fetch("POSTGRES_HOST"),
                           port: ENV.fetch("POSTGRES_PORT"),
                           database: ENV.fetch("POSTGRES_DB"),
                           max_connections: 10)
```

4. Zlib

Knižnica (<https://ruby-doc.org/stdlib-2.4.0/libdoc/zlib/rdoc/Zlib/GzipReader.html>), ktorá načítava GZip súbory, odzipuje ich a ďalej vracia do programu.

3. Popis SQL

Všetky SQL príkazy sú generované pomocou ORM.

Na začiatku program prejde [všetky tabuľky](#) a pre každú z nich vytvorí CREATE TABLE príkaz, ktorý však obsahuje podmienku IF NOT EXISTS, vďaka čomu sa tabuľka vytvorí len keď neexistuje. Príkladom takého SQL, ktoré vytvorí tabuľku *hashtags* je napríklad:

```
CREATE TABLE IF NOT EXISTS "hashtags" ("id" bigint GENERATED BY DEFAULT AS
IDENTITY NOT NULL PRIMARY KEY, "tag" text NOT NULL UNIQUE)
```

Tabuľku *links* napíšeme týmto príkazom:

```
CREATE TABLE IF NOT EXISTS "links" ("id" bigint GENERATED BY DEFAULT AS
IDENTITY NOT NULL PRIMARY KEY, "conversation id" bigint NOT NULL REFERENCES
"conversations", "url" varchar(2048) NOT NULL, "title" text, "description"
text)
```


Pri oboch vidíme, vygenerované SQLko je CREATE TABLE command, ktorý sa vykoná len, ak tabuľka neexistuje. IDčka sú v oboch prípadoch generované automaticky, ak žiadne nebolo poskytnuté, alebo, ak v INSERT príkaze bolo vložené aj IDčko, tak sa použije to, plus IDčka nesmia byť null a sú typu primárny kľúč. Ostatné stĺpce podľa potreby, ak nesmie byť null, tak ORMko vygeneruje NOT NULL pri danom stĺpci, ako napríklad pri stĺpci *tag* pri hashtag-och, spolu s UNIQUE constraint.

Po vytvorení všetkých tabuliek nasleduje samotné vkladanie:

```
INSERT INTO "authors" ("id", "name", "username", "description",
"followers_count", "following_count", "tweet_count", "listed_count") VALUES
('257732199', 'えむわじ', 'mwazi_h9', 'ゲーム/特撮/日常雑多に呟いてます。ゲームはゼル
伝/FE/世界樹/パテンカイトス/テイルズ/FFなどなど。たまに同人系の発言が混ざりますのでご注意
下さい。成人済。', 220, 399, 236420, 9), ('1204816737662853121', 'Socrates in
brave new world 🇧🇷', 'Mohsen7729', 'Med Student,
Interested in art, literature, philosophy and politics,
Ravenclaw 🦉', 2198, 390, 15651, 4)
```

ORM vygenerovalo INSERT INTO command, spolu s VALUES, čo nám umožňuje na jeden prístup k databáze vložiť *batch_size* počet záznamov, v našom prípade 100 000 záznamov. Samozrejme, toto nie je celý vygenerovaný príkaz, ale len prvý vložený element z neho. INSERT do tabuľky autori na koniec tohto príkazu ešte pridáva ON CONFLICT DO NOTHING ako na obrázku:

```
'Documenting my experience on the piece of Rock 🇯🇲 and the rest of the
world', 948, 887, 15917, 0) ON CONFLICT DO NOTHING
```

Vďaka tomu sa preskakujú záznamy, ktoré majú duplicitné ID pri autoroch.

Ďalšia query ktorá sa využíva je klasická SELECT query, ktorá vráti všetky IDčka pre danú tabuľku. Príklad: `SELECT "id" FROM "authors"`

4. Časový opis priebehu a dĺžka importu

Importovanie všetkých tabuliek, spolu s ich vytvorením trvalo v poslednom behu programu 4 hodiny 13 minút a 37 sekúnd. Samozrejme sa okolnosti líšia a preto sa čas programu v sekundách, alebo možno aj minútach môže meniť. 4 hodiny 10-15 minút by však malo zostať, tak ako zobrazuje aj CSV súbor.

Veľkosť importovaných batch-ov je 100000 záznamov.. Import autorov zbehné približne do 6-7 minút. Dĺžka jedného batch-u pri autoroch je 5-6 sekúnd. Konverzácie sú určite najpomalšia časť programu a zaberajú približne 3 hodiny aj 10 minút. Referencie sú podobné autorom, ale ich import je pomalší, pretože si držia väčšie dáta, ako napríklad existujúce konverzácie. Ich import zaberá približne 50 minút.

Na obrázku je vidieť zlom, kedy sa skončil import konverzácií a 8 ďalších tabuliek a začal sa import referencií. Pri konverzáciách je dĺžka spracovania jedného batch-u okolo 40-45 sekúnd, pri referenciách je to okolo 7-9 sekúnd.

```
2022-10-04T03:26+0200,201:58,00:43
2022-10-04T03:27+0200,202:40,00:41
2022-10-04T03:27+0200,203:22,00:42
2022-10-04T03:28+0200,203:43,00:21
2022-10-04T03:29+0200,204:58,01:14
2022-10-04T03:29+0200,205:05,00:07
2022-10-04T03:29+0200,205:12,00:07
2022-10-04T03:29+0200,205:20,00:07
2022-10-04T03:29+0200,205:28,00:07
2022-10-04T03:29+0200,205:36,00:07
2022-10-04T03:30+0200,205:43,00:07
```

5. Počet a veľkosť záznamov v tabuľkách

Výsledná databáza má veľkosť 30GB.

Annotations

number_of_rows	size
19458972	1721 MB

Authors

number_of_rows	size
5895176	1069 MB

Context_annotations

number_of_rows	size
134285948	10 GB

Context_domains

number_of_rows	size
88	64 kB

Context_entities

number_of_rows	size
29438	4176 kB

Conversation hashtags

number_of_rows	size
54613745	3888 MB

Conversation references

Pri tejto tabuľke sú dva výsledky z dôvodu, že som nevedel, že či vkladáme všetky links, alebo vynechávame duplicitné linky, teda tie, ktorých conversation links boli už importovné.

Toto riešenie je použité aj vo výslednom kóde.

Prvý výsledok je v prípade, že takéto linky vynechávame.

number_of_rows	size
27917087	2399 MB

Druhý obrázok je v prípade, že tieto linky nevynechávame.

number_of_rows	size
27950190	2402 MB

Conversations

number_of_rows	size
32347011	8632 MB

Hashtags

number_of_rows	size
773865	88 MB

Links

number_of_rows	size
11540704	2022 MB