[PDT] Zadanie 1

Link na Github repozitár: <https://github.com/xvesely/PDT_Zadanie1>

V rámci prvého zadania z predmetu PDT našou úlohou bolo naimportovať dáta, ktoré boli uložené vo formáte .jsonl.gzip, do relačnej databázy Postgres s preddefinovanou schémou. Konkrétne sme disponovali dvomi komprimovanými súbormi, ktoré prakticky obsahovali JSON objekty reprezentujúce autorov, resp. tweety/konverzácie, ktoré sme následne mali rozparsovať a jednotlivé vzťahy medzi entitami vyjadriť pomocou nasledovnej schémy.



# Vstupné dáta

Všetky vstupné dáta sa nachádzajú v dvoch súboroch, **authors.jsonl.gz** a **conversation.jsonl.gz**. Pomocou knižnice **gzip** v rámci Pythonu sme boli schopný čiastočne extrahovať dáta, konkrétne iterovať pozdĺž jednotlivých riadkov korešpondujúcich k jednotlivým entitám (formát JSON Lines), a s ktorými sme mohli naďalej pracovať. Zatiaľ čo entita typu **authors**, nachádzajúca sa v rovnomennom súbore, obsahovala dáta potrebné na tvorbu záznamov v rámci **authors** tabuľky, tak ostatné dáta potrebné na vytvorenie dodatočných tabuliek v hore uvedenej schéme databázy sa nachádzali v rámci atribútov entít **conversations.**

## Parsovanie dát

Akonáhle sme extrahovali JSON ekvivalentný objektu **authors** alebo **conversations** zo súborov, mohli sme kontrolovať ich platnosť, a teda či obsahujú relevantné atribúty, ktoré žiadame v rámci danej schémy. Konkrétne, pokiaľ JSON objekt neobsahoval všetky atribúty, ktoré má definované v schéme databázy, a popritom ide o nie nullable atribúty, tak ide o neplatný objekt a preskakujeme ho. Pokiaľ však dané atribúty sú nullable, ako napríklad v prípade schémy relácie **authors**, tak dané atribúty boli nastavené na None (null), pričom sa ďalej pracovalo s daným súborom.

Taktiež za neplatný súbor považujeme entitu, ktorá používa rovnaké IDčko, ktoré už sa nachádza v databáze. V takomto prípade ďalej nekontrolujeme, či aj ich ostatné atribúty sú ekvivalentné, a v prípade, že dané entity s rovnakým ID sa nerovnajú, tak sa vždy zapíše do databázy ten objekt, ktorý bol spracovaný ako prvý.

Na obrázku nižšie máme zobrazený príklad parsovania a upravovania JSON objektu reprezentujúceho entitu autora.

Text

Description automatically generated

# Algoritmus importovania dát

Náš celý proces importovania dát je rozdelený na viacero etáp s cieľom efektívneho a rýchleho importovania dát.

Hlavné etapy nášho algoritmu sú vymenované nižšie:

* Vytvorenie tabuľky autorov
* Vytvorenie tabuľky konverzácií
* Paralelné vytvorenie ostatných tabuliek

## Vytvorenie tabuľky autorov

Prvým krokom nášho algoritmu je vytvorenie tabuľky **authors** z dát získaných zo súboru **authors.jsonl.gz.** V rámci importovania jednotlivých záznamov autorov, sme si ukladali do pamäte zoznam všetkých doterajších IDčok, aby v prípade, že narazíme na duplikát, sme ho mohli rovno ignorovať. Tento zoznam je následne použitý pri potenciálnej tvorbe ďalších autorov v rámci funkcie na importovanie konverzácií.

Druhou možnosťou, ako určovať, či ide o duplikát v tabuľke autorov, resp. všeobecne v hocijakých tabuľkách je spracovávať všetky objekty v danom gzip súbore a použiť Posgres mechanizmy ako napríklad INSERT ... ON CONFLICT DO NOTHING na zabránenie vzniku konfliktom medzi objektami s rovnakým ID. Avšak kvôli paralelnému charakteru našich funkcií v neskorších etapách, ako aj kvôli faktu, že sme použili Postgres príkaz COPY, ktorý nepodporuje ON CONFLICT klauzulu, sme sa rozhodli zostať pri jednoduchom ukladaní IDčok do poľa.

Konkrétne sme si ukladali IDčka do dátového typu dictionary v Pythone, kedy sme samotné ID použili ako kľúč. Takýmto štýlom nasledovné kontrolovania, či nové IDčko sa nachádza v zozname kľúčov dictionary je oveľa rýchlejšie ako v prípade, pokiaľ by sme museli realizovať lineárne prehľadávanie poľa.

## Text Description automatically generated

## Vytvorenie tabuľky konverzácií

V rámci druhého kroku migrácie dát do relačnej databázy sme sa sústredili na importovanie dát do tabuľky konverzácií, pričom hlavným dôvodom, prečo sme separátne spracovávali túto tabuľku, a nie spolu s ostatnými tabuľkami, je fakt, že veľké množstvo ďalších tabuliek je naviazaných na túto tabuľku pomocou foreign keys.

Ako aj v prípade importovania dát do tabuľky **authors**, aj v tomto prípade sme si ukladali do pamäte zoznam IDčok, ktoré boli už použité, aby sme sa vyhli duplikátom. Tento mechanizmus identifikácie duplicitných konverzácií následne používame aj v treťom kroku pri paralelnom spracovaní dát nad viacerými tabuľkami.

## Paralelné vytvorenie ostatných tabuliek bez použitia foreign key obmedzení

V rámci tejto etapy sme použili multiprocessing a použili viacero procesov, ktoré sa sústreďovali na parsovanie a následné importovanie dát do vlastných tabuliek. Vďaka tomuto prístupu sme výrazne znížili čas potrebný na importovanie dát získaných z **conversation.jsonl.gz**.

Konkrétne sme spustili 3 paralelné procesy (P1 = hashtags + conversation\_hashtags; P2 = annotation + links + conversation\_references; P3 = context\_entities + context\_domains + context\_annotations), ktoré sa sústreďovali na prvotné parsovanie dát z **conversation.jsonl.gz** súboru a následného importu do vlastných tabuliek. Tabuľky sú úmyselne rozložené medzi procesmi takým štýlom, aby tabuľky, ktoré zdieľali nejaké relácie vo forme foreign keys boli importované do databázy nami určeným poradím – aby vždy existovali dáta, na ktoré iné referencujú.

Vo všetkých troch uvedených procesoch sme sa držali podobnej logiky pridávania nových záznamov do tabuliek. Konkrétne sme iterovali .gz súbor, parsovali relevantné dáta z celého JSON **conversations** objektu, a v prípade, že sme narazili na duplikát konverzácií, resp. na neplatnú konverzáciu, tak daný objekt preskakujeme a neparsujeme z neho nami hľadané dáta.

Na obrázku nižšie nachádzame hlavnú logiku za importovaním dát do tabuliek, ktoré boli získané z **conversation.jsonl.gz** súboru. V rámci tejto vizualizácie sme sa sústredili na importovanie dát do tabuľky **annotations**, pričom ostatný kód v danej funkcii je kolapsnutý.

Text

Description automatically generated

Jednotlivé zvýraznené časti kódu realizujú:

* Iterácia cez .gz súbor
* Premenná, do ktorej akumulujeme dáta na vloženie – vkladáme dáta po batchoch
* Premenná typu dictionary, do ktorej si ukladáme všetky IDčka konverzácií
* Kontrola platnosti JSON objektu konverzácie a následné skontrolovanie, či nejde o duplikát
* Parsovanie anotácií
* Vloženie anotácií do premennej na akumulovanie dát, pokiaľ ich bude rozumné množstvo (nad batch), realizujeme vkladanie dát pomocou funkcie COPY
* Logovanie časov

# Použité technológie

Na realizáciu tohto zadania sme použili programovací jazyk Python, ktorý obsahuje veľké množstvo knižníc na prácu s dátami všeobecne, či s databázami. Konkrétne na prácu s Postgres databázou sme použili knižnicu **psycopg3**, ktorá oproti známejšej verzii **psycopg2** poskytuje efektívnu funkciu kopírovania dát COPY, ako aj default podporuje unicode encoding.

# Opis použitých queries

Nižšie máme zobrazené reprezentatívne query stringy rozličných úkonov nad databázou.

Text

Description automatically generated

Pomocou tejto query definujeme atribúty a ich obmedzenia v rámci tvorby tabuliek. Tieto queries sa nachádzajú na začiatku každej funkcie na importovanie dát. V tomto konkrétnom prípade vytvárame tabuľku autorov.

Text

Description automatically generated

Tieto typy príkazov sme primárne používali v rámci developmentu nášho kódu, avšak nie sú nutné na realizáciu v rámci finálneho tvaru nášho projektu. Pomocou tejto query vymazávame tabuľky z databázy, čo je vhodné v prípade, kedy chceme začať importovať dáta odznova.

Text

Description automatically generated

Tento tvar príkazu sme použili zo stránky <https://www.psycopg.org/psycopg3/docs/basic/copy.html>, pričom hlavnou výhodou príkazu COPY je vysoká rýchlosť importovania dát. Tento príkaz používame na vkladanie dát do všetkých tabuliek. Na obrázku je zobrazený príklad improtovania dát do tabuľky konverzácií, kedy do query stringu nešpecifikujeme žiaden placeholder, ale určíme štandardný vstup ako zdroj dát. Následne dáta, ktoré sú v tvare dvojrozmerného poľa, sú pomocou for cyklu vkladané a zapisované do databázy.

Text

Description automatically generated

Tento príkaz využívame na získanie všetkých platných IDčok konverzácií v databáze a na identifikáciu neplatných záznamov so zlým atribútom **parent\_id**, ktoré sú následne mapované na reláciu **conversation\_references**. V tomto snippete možno aj vidieť konverziu výstupu databázy do Python dictionary, ktorý používame na efektívne prehľadávanie IDčok.

# Časové logy

V tejto časti práce vizualizujeme tvar a kúsok z logov zo súboru **conversations.csv**. Nakoľko sme pracovali s viacerými procesmi, rozhodli sme sa jednotlivé procesy, či kroky v našom algoritme rozdeliť do separátnych .csv súborov, pričom však dodržiavame rovnaký začiatočný čas, druhý stĺpec, ktorý definuje strávený čas od spustenia programu.

Na obrázku nižšie možno vidieť logovaný čas v rámci prvotných importov do databázy. Čas strávený pred logovaním týchto importov, čas 2m46s bol strávený importovaním autorov.

Nakoľko relatívne rýchlo sa ukladali dáta do databázy, rozhodli sme sa zväčšiť krok, s ktorým logujeme časy do csv súborov, kedy namiesto 10 000 sme použili krok o veľkosti 1 000 000.

Table

Description automatically generated

# Výsledné tabuľky

Tu uvádzame početnosti a veľkosti jednotlivých tabuliek, ktoré sme vytvorili v rámci nášho migrovania dát do relačnej databázy. Naša celková databáza má okolo 29.35 GB.

Table

Description automatically generatedTable

Description automatically generated