[PDT] Zadanie 1

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 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 si extrahovali JSON ekvivalentný objektu **authors** alebo **conversations** zo súborov, mohli sme kontrolovať ich platnosť, skontrolovať, či obsahujú relevantné atribúty, ktoré žiadame v rámci danej schémy. V prípade, že dané atribúty neexistovali, tak na základe obmedzení definovaných v schéme, resp. podľa dôležitosti atribútu sme usúdili, či je daný JSON objekt platný, alebo ho preskočíme.

Napríklad pokiaľ objekt konverzácie nemá platné číselné IDčko, resp. neobsahovalo potrebné atribúty, ako napríklad **created\_at**, či **author\_id**, tak považujeme daný objekt za neplatný a nie je importovaný do relačnej databázy. Tzv. „gray area“ by som nazval riešenie hodnôt, ktoré sú síce povinné, teda nie sú nullable, ale existujú default hodnoty, ktoré im môžeme priradiť, aby sme nemuseli celý záznam vymazávať. Príkladom je napríklad atribút **possibly\_sensitive**, ktorý síce je povinný, ale pokiaľ ho nenájdeme v JSON objekte, tak mu priradíme default hodnotu – False.

# Algoritmus importovania dát

Náš celý proces importovania dát je rozdelený na viacero etáp, vďaka ktorým vieme takmer jedným prechodom dátami získať všetky atribúty potrebné na tvorbu preddefinovanej schémy.

Hlavné etapy nášho algoritmu sú vymenované nižšie:

* Vytvorenie tabuľky autorov
* Paralelné vytvorenie ostatných tabuliek bez použitia foreign key constraints
* Odstránenie neplatných záznamov z tabuľky conversation\_references
* Priradenie foreign key / unique constraints pre konkrétne atribúty v schéme

## Vytvorenie tabuľky autorov

Prvým krokom nášho algoritmu je vytvorenie tabuľky **authors** z dát získaných zo súboru **authors.jsonl.gz.** Tento proces vie byť paralelne realizovaný v rámci tvorby ostatných tabuliek, avšak nakoľko ide o relatívne málo dát, ktoré sú rýchlo importované (do 5 minút), preferovali sme takúto sekvenčnú tvorbu tabuliek. Ďalšou výhodou separátnej tvorby tabuľky **authors** je fakt, že následne pri tvorbe tabuľky **conversations**, máme k dispozícii zoznam všetkých IDčok autorov, a tak vieme určiť, kedy je potrebné vytvoriť nového doposiaľ neexistujúceho autora.

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.

## 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**, kedy najväčším „bottleneckom“ bolo získanie dát pre tabuľky **context-domains, context-entities** a **context-annotations**, ktorý importoval dáta rýchlosťou 8 sekúnd na 100 000 JSON konverzácií.

Konkrétne sme spustili 4 procesy (P1 = conversations; P2 = hashtags + conversation\_hashtags; P3 = annotation + links + conversation\_references; P4 = context\_entities + context\_domains + context\_annotations), ktoré sa sústreďovali na prvotné parsovanie dát z **conversation.jsonl.gz** súboru a následného importu do vlastných tabuliek. Úmyselne sme pri tvorbe tabuliek nedefinovali obmedzenia pre foreign keys, nakoľko dané obmedzenia spomaľujú vkladanie dát, a neboli pre nás potrebné, nakoľko sme si ich spracovávali sami.

Vo všetkých štyroch 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.

Nižšie na obrázku máme zobrazenú konkrétnu funkciu, ktorú sme použili na importovanie dát do tabuľky **annotations** s vyznačenými logickými časťami danej funkcie.

TODO obrazok, do ktoreho vizualizujem hlavne casti daneho kodu

## Odstránenie neplatných záznamov z tabuľky conversation\_references

V rámci jedného prechodu dátami sme však neboli schopný verifikovať platnosť záznamov **conversation\_references**, konkrétne či existuje konverzácia s ID hodnotou v atribúte **parent\_id**, nakoľko sme v rámci tvorby danej tabuľky nemali celý zoznam IDčok konverzácií. V rámci tejto etapy identifikujeme všetky záznamy, ktoré svojim atribútom **parent\_id** ukazujú na neexistujúcu konverzáciu a následne sa odstránia.

## Priradenie foreign key / unique obmedzenia pre konkrétne atribúty v schéme

V poslednej etape zfinalizujeme schému databázy, aby korešpondovala so zadanou schémou na začiatku projektu. Konkrétne pridáme FK constraints či UNIQUE pre hashtagy, ktoré by spomalili pridávanie dát v rámci importovania a radšej sme si ich manuálne zabezpečili pri importovaní dát.

# Použité technológie

Na realizáciu tohto zadania sme použili programovací jazyk Python, ktorý obsahuje veľké množstvo knižníc na prácu s dátami všeobecne, či so 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

asdasdasd