Fakulta informatiky a informačných technológií
STU Bratislava

Pokročilé databázové technológie

Protokol - Zadanie 1

Jakub Povinec
https://github.com/kuko6/bulk-tweets-import

Použité technológie

Na vypracovanie zadania som použil programovací jazyk Python (3.10.6) s knižnicou **psycopg2**, ktorá predstavuje adaptér pre databázu Postgres. Pomocou tejto knižnice som sa dokázal na databázu pripojiť ako aj vykonávať jednotlivé príkazy, buď pomocou metódy `cur.execute()` alebo `execute_values()`. Ako databázu som použil Postgres 14 a riešenie som testoval na M1 MacBook Air so 16 GB pamäte.

Opis algoritmu

Samotné importovanie dát pozostáva zo štyroch hlavných častí:

- vytvorenie tabuliek
- importovanie autorov
- importovanie konverzácií a všetkých ostatných relácií okrem `conversation_references`
- importovanie `conversation_references`

Vytvorenie tabuliek

Tabuľky sú vytvárané v metóde `create_tables()`, kde sa pomocou metódy `cursor.execute()` z knižnice **psycopg2** vykoná SQL zo súboru `schema.sql`, ktorý obsahuje potrebné príkazy pre vytvorenie všetkých tabuliek.

```
CREATE TABLE IF NOT EXISTS conversation_references (
    id bigserial PRIMARY KEY,
    conversation_id int8 NOT NULL,
    parent_id int8 NOT NULL,
    type varchar(20) NOT NULL
);
```

Fig. 1 - Časť SQL pre vytvorenie tabuľky conversation_references

Pri vytváraní tabuliek som sa rozhodol nedefinovať obmedzenia pre cudzie kľúče, nakoľko to je aj jedným z odporúčaní v samotnej Postgres dokumentácií (14.4. Populating a Database) podľa, ktorej by sa tak mal zrýchliť čas importu. V mojom riešení sa tieto obmedzenia vytvoria vždy na konci importovania pomocou príkazu `ALTER TABLE`. Nižšie sa nachádza ukážka vytvárania týchto obmedzení.

```
ALTER TABLE conversation_references
      ADD CONSTRAINT fk_conversations
      FOREIGN KEY (conversation_id)
      REFERENCES conversations(id);

ALTER TABLE conversation_references
      ADD CONSTRAINT fk_parent
      FOREIGN KEY (parent_id)
      REFERENCES conversations(id);
```

Fig. 2 - SQL pre vytvorenie obmedzení pre tabuľku conversation_references

Pri importovaní dát sa v kóde taktiež kontroluje, či záznam s daným id existuje, preto sa nemôže stať, že by hodnota odkazovala na neexistujúci záznam.

Importovanie dát do tabuliek

Importovanie dát je z veľkej časti rovnaké pre všetky tabuľky. Pozostáva z čítania príslušného súboru (`authors.jsonl.gz`, `conversations.jsonl.gz`) po riadkoch, kde sa každý riadok prekonvertuje na `dictionary` (`dict`) aby sa s ním dalo pracovať. Následne sa overí či sa záznam s rovnakým `id` už v tabuľke nenachádza, kedy by sa takýto objekt preskočil. Kvôli tomuto overeniu používam dátovú štruktúru `set`, do ktorého sa ukladajú `id` vložených záznamov. `Set` používam pretože jednotlivé hodnoty sú v ňom zahashované a vyhľadávanie má teda zložitosť O(1).

```
inserted_ids = set()
data = []
cur = connection.cursor()
with gzip.open('./file.jsonl.gz') as file:
      for line in file:
            json_obj = json.loads(line)

            if json_obj['id'] in inserted_ids: continue

            data.append((json_obj['id'], ...))
            inserted_ids.add(json_obj['id'])

            if len(inserted_ids)%10000 == 0:
                  execute_values(cur, "INSERT INTO table (...) VALUES %s", data)
                  data = []

      if len(inserted_ids) > 0:
            execute_values("INSERT INTO table (column_names) VALUES %s;", data)

cur.execute("ALTER TABLE table ADD CONSTRAINT fk_othertable FOREIGN KEY
(some_id) REFERENCES othertable(id);")
connection.commit()
cur.close()
```

Fig. 3 - Pseudokód pre importovanie dát

Hodnoty, ktoré sa majú z daného objeku zapísať do databázy sa uložia do listu. Keď tento `list` obsahuje 10000 záznamov, dáta sa naraz vložia do databázy pomocou príkazu `INSERT`. Do databázy sa naraz vkladá všetkých 10000 nových riadkov, čo je rýchlejšie ako individuálny `INSERT` pre každý riadok (pri tabuľke `authors`, trvalo vkladanie záznamov individuálnymi insertami okolo 440s, zatiaľ čo pri skupinovom inserte okolo 115s).

```
INSERT INTO authors (id, name, username, description, followers_count,
following_count, tweet_count, listed_count) VALUES
      (1, 'Jakub', 'jakub22', 'aaaaaaa', 1000, 2, 3000, 400),
      (2, 'Juraj', 'juraj01', 'bbbbbbb', 420, 300, 200, 200),
      (3, 'Jana', 'jana10', 'ccccccc', 200, 20, 100, 120),
      ...
;
```

Fig. 4 - Ukážka príkazu INSERT pre tabuľku authors

Ešte rýchlejšou metódou by bolo použitie príkazu `COPY` (kedy trvalo naplnenie tabuľky `authors` 90s), ktorý sa do môjho riešenia ale veľmi nehodil, hlavne kvôli spôsobu akým som riešil relácie, ktoré vychádzajú z konverzácií (`links, annotations, context_domains, ...`). Pre skupinový `INSERT` som použil metódu `execute_values()` z knižnice **psycopg2**. Po prejdení celého súboru sa ešte vložia dáta, ktoré sa nezmestili do celého bloku a pridajú sa už spomínané obmedzenia pre cudzie kľúče. Vo svojom riešení "commitujem" prebiehajúce transakcie vždy až úplne na konci, čo by malo byť podľa už spomínanej Postgres dokumentácie (14.4. Populating a Database) rýchlejšie ako "commitovať" napr. po každom príkaze `INSERT`.

Importovanie dát zo súboru authors.jsonl

Na importovanie dát do tabuľky authors slúži metóda `import_authors()`, ktorá je prakticky rovnaká ako priložený pseudokód (Fig. 4). Jediným rozdielom je, že v tomto prípade sa nevytvárajú žiadne obmedzenia. Výstupom tejto metódy je `set inserted_authors`, ktorý obsahuje `id` autorov, ktorí boli vložení do databázy. Tento `set` sa neskôr používa v metóde pre importovanie konverzácií, kde slúži na odhalenie chýbajúcich autorov.

Importovanie dát zo súboru conversations.jsonl

Na importovanie dát do tabuľky `conversations` a k nej prislúchajúcim tabuľkám (`links`, `annotations`, `context_references`, `context_domains`, `context_entities`, `hashtags` a `conversation_hashtags`) slúži metóda `import_conversations()`. Samotné pridávanie konverzácií je rovnaké ako v pseodokóde (Fig. 4). Rozdielom je kontrolovanie chýbajúcich autorov daných tweetov, ktorých `id` sa pridá do `set`-u `missing_authors` a na konci sa spoločným insertom pridajú do databázy `authors`.

Ďalšou rozdielnou časťou je napĺňanie ostatných tabuliek, kedy sa najskôr zistí či daný objekt (tweet) obsahuje pole `entities` a polia v `entities` (`hashtags`, `annotations` a `urls`), resp. `context_annotations`. Importovanie do tabuliek `annotations` a `links` je veľmi jednoduché, kedy sa len prejde cez príslušné pole a relevatné údaje z objektov sa pridajú do databázy. Pri importovaní hashtagov si udržiavam `dict` (tag: id), `inserted_hashtags`, kde sa vždy pred importovaním hashtagov najskôr skontroluje, či sa daný tag nenachádza v `dict`, kedy by sa vytvoril nový záznam len v databáze `conversation_hashtags` s `id` príslušného tagu v `dict`. Podobný prístup využívam aj pri importovaní do tabuliek `context_annotations`, `context_domains` a `context_entities`, ktoré sa v jednotlivých tweetoch často opakujú. V tomto prípade si, ale uchovávam `set` s `id` pre dané objekty (`inserted_context_domains`, `inserted_context_entities`) a vždy pri pridávaní týchto objektov do databázy sa najskôr skontroluje, či je potrebné vytvoriť nový záznam v príslušnych databázach alebo sa len vytvorí záznam v databáze `context_annotations`.

Na koniec sa vytvoria obmedzenia pre cudzie kľúče v príslušných tabuľkách, explicitne sa vymažú pomocné `listy`, `sety` a `dict` a taktiež sa spustí garbage collector (pomocou `gc.collect()`) aby sa uvoľnila pamäť. Metóda vracia `set` s `id` vložených tweetov, ktorý sa používa v ďalšej metóde na kontrolovanie neexistujúcich parent tweetov.

Importovanie dát zo súboru conversations.jsonl

Na importovanie `conversation_references` slúži metóda `import_references()`, v ktorej sa znovu prechádza súbor `conversations.jsonl`. Na rozdiel od ostatných metód sa v tejto metóde ešte kontroluje, či `id` v objekte `referenced_tweets` neodkazuje na neexistujúci záznam v tabuľke `conversations`.

Záznamy by sa dali do tabuľky `conversation_references` taktiež importovať v metóde `import_conversations()`, kedy by sa len na konci vymazali tie záznamy, ktoré obsahovali `id` odkazujúce na neexistujúce tweety. Tento spôsob a obzvlášť samotné vymazávanie záznamov bol, ale príliš pomalý.

Dĺžka trvania importu

Priebeh importovania je zaznamenaný v súbore `log.csv`, kde sa zapisoval celkový čas a čas importovania daného bloku (100 000 prejdených záznamov). Celková dĺžka importu bola okolo 86 minút. Dĺžka importovania jedného bloku v metóde `import_authors()` trvala okolo 1 až 4 sekúnd a konečné "commitovanie" trvalo okolo 11 sekúnd. V metóde `import_conversations()` trval jeden blok 12 až 15 sekúnd a konečné "commitovanie" trvalo okolo 3 minút a 79 sekúnd. Importovanie `conversation_references` trvalo okolo 2 až 9 sekúnd po blokoch a konečné "commitovanie" trvalo okolo 32 sekúnd.

Počet a veľkosť záznamov

Informácie o finálnom objeme dát je možné vidieť na obrázku nižšie (Fig. 5).

table_name	number_of_rows	size
annotations	19458972	1303 MB
authors	5895176	904 MB
context_annotations	134285948	7714 MB
context_domains	88	16 kB
context_entities	29438	3248 kB
conversation_hashtags	54613745	2718 MB
conversation_references	27917087	1801 MB
conversations	32347011	7915 MB
hashtags	773865	40 MB
links	11540704	1774 MB

Fig. 5 - Finálny počet a veľkosť záznamov