PDT ZADANIE 1
PROTOKOL

Filip Vida

4.10. 2022

Github

https://github.com/xvidaf/Vida_PDT_1

https://github.com/FIIT-DBS/zadanie-pdt-xvidaf

Použité technológie

Skript je napísaný v jazyku Python 3.9. Python bol vybratý kvôli tomu, že jeho knižnica uľahčuje pandas uľahčuje prácu s veľkými dátami, a knižnica psycopg poskytuje jednoduchý spôsob napojenia sa na postgres databázu, na ktorej vieme následne vykonávať SQL.

Hlavné použité technológie:

Pandas: Ešte pred tým, ako sme začali s vkladaním dát sme si data rozdelili do súborov a pripravili na vloženie. Pre zjednodušenie parsovania poskytnutých súborov a ich zápis do .csv bola využitá knižnica pandas, špecificky dátový typ dataframe.

Psycopg3: Táto knižnica je rewrite pre často používanú knižnicu psycopg2, ktorý by mal zaistiť lepšiu integráciu s novšími verziami pythonu a postgresu. Po pripojení na databázu reálne využitie však spočívalo len vo funkcii .copy a .execute, ktoré sme používali pri kopírovaní záznamov do tabuľky a pre vykonanie queries. Keďže máme dáta vo veľkých množstvách a potrebujeme ich do databázy dostať čo najrýchlejšie, bolo zvolené priamo písanie SQL, nie ORM.

Pomocné technológie:

Datetime: Pre uľahčenie zápisu času.

Time: Pre uľahčenie sledovania ubehnuté času skriptu.

Csv: Pre jednoduchý zápis do .csv súboru, konkrétne pre časový záznam.

Opis algoritmu

Vytvorený skript na vloženie dát je v podstate rozdelený na 2 časti, a to na prípravu dát a samotné vloženie dát.

Pre prípravu dát sú pripravené funkcie prepareAuthors() a prepareConversations(). Po ich spustení sa z poskytnutých súborov vytvoria súbory .csv, ktoré sú rozdelené podľa tabuľky do ktorej budú neskôr nakopírované, pričom každý súbor obsahuje záznamy vytvorené zo 10 000 konverzácií alebo autorov. Toto znamená že pre tabuľky, ktoré sa vytvárajú z obsahu conversations, môžu súbory .csv obsahovať aj viac ako 10 000 záznamov. Ak sa v týchto 10 000 conversations nachádza duplikát, je odstránení. Táto metóda deduplikácie je rýchla a odstránila väčšinu duplikátov, no **nie všetky**. Tieto súbory .csv sú vytvorené tak, aby na nich mohol byť spustený jednoduchý COPY.

Takéto predvytvorenie súborov na kopírovanie bolo zvolené z niekoľkých dôvodov. Hlavný bol, že počas vytvárania skriptu je možné spúšťať skript už na roztriedených dátach, čo značne zvyšuje rýchlosť troubleshooting-u. Tiež sú tieto dáta aj po ich vložení do databázy pripravené, na jej prípadnú obnovu. Nevýhodou tohto riešenia je zvýšená potreba pre kapacitu disku.

Po pripravení súborov .csv je možné začať s importom. Pre každú tabuľku je vytvorená funkcia, ktorá do danej tabuľky vloží poskytnuté .csv súbory. Je možné preusporiadať, v akom poradí sa funckie spustia, pokiaľ ale neporušia pravidlá samotnej databázy(authors musia byť pred conversations a podobne). Každá jedna tabuľka je vložená pomocou COPY do tabuľky, ktorá je spravidla bez

constraintov PK, FK a UNIQUE, s výnimkou tabuliek, ktoré nemajú natvrdo stanovené PK, a PK musí byť vytvorené za chodu(links, context_annotations atď.). Po takomto “nahom” kopírovaní sú tabuľky pripravené na pridanie constraintov.

Jednoduchý popis akcií pre každú tabuľku(detaily sú uvedené v sekcii Opis použitých SQL):

Authors:

- Vytvorenie authors bez constraints
- Copy dát do authors
- Vymazanie záznamov s duplicitným ID
- Pridanie constraintu PK

Conversations:

- Vytvorenie conversations bez constraints
- Copy dát do conversations
- Odstránenie záznamov s duplicitným ID
- Pridanie constraintu PK
- Pridanie prázdneho záznamu do authors, ak conversation FK odkazuje na neexistujúci záznam
- Pridanie constraintu FK

Hashtags:

- Vytvorenie hashtags, ktorá obsahuje constraints a hashtags_temp bez constraints
- Copy dát do hashtags_temp
- Deduplikácia dát pomocou presunutia dát z hashtags_temp do hashtags, pri konflikte sa záznam nepridá
- Zmazanie temp tabuľky

Links:

- Vytvorenie tabuľky links, tabuľka obsahuje constraint PK, je dovolený väčšia dĺžka url ako 2048
- Copy dát do links
- Pridanie FK constraint
- Odstránenie záznamov s url dlhších ako 2048
- Pridanie constraint-u na dĺžku url

Annotations:

- Vytvorenie tabuľky annotations, tabuľka obsahuje constraint PK
- Copy dát do annotations
- Pridanie constraintu FK

Conversation_references:

- Vytvorenie tabuľky Conversation_references, s PK, bez FK
- Copy dát do Conversation_references
- Odstránenie záznamov s parent_id takých, ktoré sa nenachádzajú v Conversation_references
- Pridanie FK pre parent aj child conversation

context_domains:

- Vytvorenie tabuľky context_domains, tabuľka obsahuje constraint PK
- Vytvorenie tabuľky context_domains_temp, bez constraints
- Copy dát do context_domains_temp
- Deduplikácia dát pomocou presunutia dát z context_domains do context_domains_temp, pri konflikte sa záznam nepridá
- Zmazanie temp tabuľky

context_entities:

- Vytvorenie tabuľky context_entities, tabuľka obsahuje constraint PK
- Vytvorenie tabuľky context_entities _temp, bez constraints
- Copy dát do context_entities _temp
- Deduplikácia dát pomocou presunutia dát z context_entities_temp do context_entities, pri konflikte sa záznam nepridá
- Zmazanie temp tabuľky

context_annotations:

- Vytvorenie tabuľky context_annotations, tabuľka obsahuje constraint PK
- Copy dát do context_annotations
- Pridanie FK

conversations_hashtags:

- Vytvorenie tabuľky conversations_hashtags, tabuľka obsahuje constraint PK, tabuľka obsahuje navyše stĺpec tag, ktorý predstavuje tag z tabuľky hashtags
- Copy dát do conversations_hashtags, stĺpec hashtag_id je prázdny
- Vyplnenie stĺpca hashtag_id, pomocou query ktorá nájde ID z tabuľky hashtags, pre záznam rovný stĺpcu tag z conversations_hashtags
- Pridanie FK
- Odstránenie stĺpca tag

Opis použitých SQL

Každý insert do tabuľky sa začína v podstate rovnako, a to s vytvorením tabuľky samotnej. Začneme s tým že tabuľku dropneme ak už existuje, aby sme predošli nečakaným scenárom. Následne vytvoríme tabuľku podľa schémy, ale spravidla bez definovania constraint-ov, ktoré pridáme až na konci. Pridanie constraint-ov až po importovaní všetkých záznamov značne zrýchlilo vkladanie záznamov, a umožnilo vkladať aj záznamy s rovnakým PK, bez toho aby vkladanie padlo. Existovali však aj prípady, kde bol vytvorený constraint pre PK, a to boli prípady kedy PK nebol súčasťou vkladaných dát, ale bol automaticky pri vkladaní vytvorený.

```sql
DROP TABLE IF EXISTS conversations;

CREATE TABLE IF NOT EXISTS conversations (
    id bigint,
    author_id BIGINT NOT NULL,
    content text NOT NULL,
    possibly_sensitive bool NOT NULL,
    language varchar(3) NOT NULL,
    source text NOT NULL,
    retweet_count integer,
    reply_count integer,
    like_count integer,
    quote_count integer,
    created_at timestamp with time zone NOT NULL
);

ALTER TABLE links
    ADD FOREIGN KEY (conversation_id) REFERENCES conversations(id);
```

Samotné vloženie záznamov do tabuľky vykonávame pomocou príkazu COPY, nie INSERT. Príkaz COPY je značne rýchlejší ako INSERT, a preto bol aj zvolený. Kopírujeme priamo zo súboru CSV, ktorý si v python skripte otvoríme. Nakoniec špecifikujeme použitý oddeľovač a formát súboru z ktorého kopírujeme. Takýto druh vkladania údajov je použitý pre každú tabuľku, okrem tabuliek kde potrebujeme preskočiť nejaký stĺpec, najčastejšie pre ID, ktoré bude automaticky vygenerované(ukázané v 2. príklade pre COPY).

```sql
COPY conversations FROM STDIN (DELIMITER ';',FORMAT 'csv')

COPY links (conversation_id, url, title, description) FROM STDIN (DELIMITER ';',FORMAT 'csv')
```

Keďže sme údaje vkladali bez PK constraint-u, mohlo sa stat že vložené záznamy porušovali unikátnosť PK. Preto ešte predtým, ako sme constraint pridali, bolo nutné, takéto duplikáty odstrániť. Boli navrhnuté 2 metódy, a to použitie dočasnej tabuľky, alebo DELETE pomocou jednoduchého dopytu pre duplikáty. Takáto query pre duplikáty je uvedená nižšie, a je použitá pre deduplikovanie tabuľky conversations. Táto query nám odstráni všetky duplikátne záznamy okrem toho s najväčším ctid. CTID je použité z dôvodu, že sme potrebovali identifikovať ktoré záznamy vymažeme, a ktoré necháme, CTID obsahuje fyzickú pozíciu záznamu v tabuľke, no my ho použijeme v podstate ako PK, pre kontext rozlíšenia rovnakých záznamov. Tento druh odstránenia duplikátov je vhodný vtedy, keď sa nemaže veľká časť tabuľky, a preto je použitý pre deduplikovanie conversations či authors, ale nie na deduplikovanie context_domains.

```sql
DELETE FROM conversations USING conversations at2
    WHERE conversations.id = at2.id AND conversations.ctid < at2.ctid;

ALTER TABLE conversations
  ADD PRIMARY KEY (id);
```

Druhý spôsob deduplikácie používa dočasnú tabuľku, ktorá je bez constraints. Následne z tejto tabuľky presunieme všetky záznamy do tabuľky, ktorá constraint-y už má. Pre tabuľky kde sa vymaže

veľká časť záznamov je táto query značne rýchlejšia ako predchádzajúca, a preto je aj použítá pre context_domains a context_entities, či hashtags.

```
INSERT INTO context_entities
SELECT * FROM context_entities_temp
ON CONFLICT DO NOTHING;

DROP TABLE IF EXISTS context_entities_temp;
```

Po vložení conversations je pred pridaním FK na tabuľku authors nutné vytvoriť chýbajúcich autorov. Toto vykonáme podľa nasledújucej query, ktorá najprv nájde také author_id z conversations, ktoré neexistuje v tabuľke authors. Na výsledok tejto query aplikujeme DISTINCT, aby sa tam každé ID nachádzalo len raz. Potom takýto zoznam ID vložíme do authors, s tým že všetky ostatné polia sú NULL. Toto nám zarúčí, že každý referencovaný autor z conversations existuje.

```
INSERT INTO authors (id)
SELECT DISTINCT author_id FROM conversations lg
        WHERE  NOT EXISTS (
           SELECT FROM authors lr
           WHERE  lr.id = lg.author_id
           );

ALTER TABLE conversations
    ADD FOREIGN KEY (author_id) REFERENCES authors(id);
```

Z tabuľky links je tiež nutné odstrániť všetky url, ktoré sú dlhšie ako 2048. Ako všetky úpravy, tak aj túto kvôli rýchlosti vytvoríme na vyplnenej tabuľke. Najprv odstránime všetky linky, ktoré majú URL dlhšie ako 2048, následne môžeme pridať constraint na dĺžku.

```
DELETE FROM links where length(url) > 2048;

ALTER TABLE links ALTER COLUMN url TYPE varchar(2048);
```

V tabuľke conversation_references sa môžu nachádzať FK v stĺpci parent_id, ktoré sa v tabuľke conversations nenachádzajú. Preto je nutné query ktoré presne takéto záznamy vymaže. Ak sa ID, ktoré je v stĺpci parent_id nenachádza ako id v tabuľke conversations, je z tabuľky conversation_references vymazané. Následne môže byť priradený FK constraint.

```
DELETE FROM conversation_references lg
WHERE  NOT EXISTS (
   SELECT FROM conversations lr
   WHERE  lr.id = lg.parent_id
   );

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

Tabuľku conversations_hashtags vytvoríme tak, že obsahuje stĺpec navyše, ktorý sa volá tag a jeho obsah použijeme na spojenie s tabuľkou hashtags. Po vytvorení tabuľky pre každý riadok v conversations_hashtags nájdeme riadok v hashtags, ktorý korešponduje so stĺpcom tag v tabuľke conversations_hashtags . Z takto nájdeného riadku v hashtags zoberieme id, a priradíme ho k záznamu v conversations_hashtags. Následne môžeme pridať FK constraint a zmazať pomocný stĺpec tag.

```
UPDATE conversation_hashtags
SET hashtag_id = hashtags.id
FROM hashtags
WHERE hashtags.tag = conversation_hashtags.tag;

ALTER TABLE conversation_hashtags
    ADD FOREIGN KEY (hashtag_id) REFERENCES hashtags(id);

ALTER TABLE conversation_hashtags
    DROP COLUMN tag;
```

Ostatné použité SQL sú len variáciami už opísaných.

Časová náročnosť

Súbory obsahujúce časový výstup sa nachádzajú v priečinku TimeLogs.

Súbory sú rozdelené na 3 časti, na prečítanie a pripravenie súborov na import (createImportFilesAuthorsTime a createImportFilesConversationsTime) a na samotný celý import(insertTables). Do súborov je vždy napísané po spracovaní 10 000 záznamov.

Prečítanie a vytvorenie .csv súborov zo súboru authors trvalo 5 minút a 48 sekúnd, čas potrebný pre 10 000 záznamov bol menej ako 1 sekunda. Zvyšovanie času potrebného na 10 000 záznamov sa počas behu neprejavilo.

Prečítanie a vytvorenie .csv súborov zo súboru conversations trvalo 107 minút a 2 sekundy, čas potrebný pre 10 000 záznamov bol cca. 1 sekunda. Zvyšovanie času potrebného na 10 000 záznamov sa počas behu neprejavilo.

Čas potrebný pre vytvorenie všetkých .csv súborov pre tabuľky bol dokopy okolo 1 hodiny hodín a 53 minút pre poskytnutý testovací beh.

Súbor insertTables.csv obsahuje čas potrebný na import z takto vytvorených .csv súborov. Poskytuje časový náhľad na trvanie importu každých 10 000 záznamov(pre tabuľku conversation a authors) alebo pre toľko záznamov, koľko bolo vyčítaných z 10 000 converastion záznamov(v tabuľkách hashtags, links a podobne). V určitých časových intervaloch je vidieť značné spomalenie, toto znamená že sa do tabuľiek pridávajú constraint-y, č isa mažú duplikáty.

Čas potrebný pre nasledovný import všetkých záznamov do databázy bol 58 minút a 50 sekúnd, pre poskytnutý testovací beh.

Dokopy čas potrebný pre naplnenie databázy bol v poskytnutom prípade približne **3 hodiny**.

Počet záznamov a veľkosť tabuliek

Velkosť:

	annotations text	authors text	context_annotations text	context_domains text	context_entities text	conversation_hashtags text	conversation_references text	conversations text	hashtags text	links text
1	1721 MB	1202 MB	10 GB	64 kB	4168 kB	8774 MB	2437 MB	8666 MB	88 MB	2040 MB

Veľkosť databázy je dokopy okolo 34 GB.

Počet záznamov:

	annotations bigint	authors bigint	context_annotations bigint	context_domains bigint	context_entities bigint	conversation_hashtags bigint	conversation_references bigint	conversations bigint	hashtags bigint	links bigint
1	19458993	5895176	134286096	88	29438	54613811	27917121	32347011	773865	11540715

Niektoré tabuľky môžu obsahovať záznamy navyše, toto je spôsobené tým, že po vymazání záznamov s duplikátnym ID v conversations, bolo nemožné určiť ktoré záznamy k takýmto vymazaným konverzáciám patrili, keďže mali rovnaké ID. Niektoré duplikáty sa podarilo vymazať pri vytváraní súborov csv. (pokial boli v rovnakom 10 000 bloku), no to neboli všetky.