Fakulta informatiky a informačných technológií STU v Bratislave

Zadanie 1 – Import dát

29.9.2022

Matej Delinčák

Opis algoritmu a postupu pri importe

Postup pri importe som započal naštudovaním problému, ktorý by mohol vzniknúť pri importe veľkého množstva dát. A na základe tejto analýzy som sa rozhodol pre danú technológiu, o ktorej poviem neskôr. Proces importovania dát z gzipu spočíval z otvorenia oboch súborov postupne. Najprv som spracoval autorov a po importnutí všetkých som začal spracovávať konverzácie. Gzip som nerozbaľoval do súborov, ale použil som libku *"gzip"*, ktorá otvorila tento archív ako bežný súbor, z ktorého sa dá čítať riadok po riadku. A teda riadok po riadku som čítal súbor a pomocou libky *"orjson"* som z formátu *json* dostal klasický *python dict*. Pre autorov si ukladám aj set id-čiek, ktoré som už spracoval, aby som mohol odfiltrovať duplicitné záznamy autorov. Rovnaký prístup som použil aj pre konverzácie. Spracovanie jedného riadku zo súboru *conversations* bolo trocha zložitejšie ako pri *authors*, pretože bolo treba sa viacej pohrať s dátami, ktoré tento json ponúkal. Bolo potrebné vytvoriť viacero entít, ktoré sa vkladali do svojich tabuliek. Pre číselníky *context_domains* a *context_entities* som si držal set id-čiek, ktoré sú už boli spracované, aby som ich znova nevkladal do databázy. Pre číselník *hashtags* som si udržiaval *dict*, kde kľúč bol tag a hodnota priradené id. Pamätám si to preto, aby som nemusel robiť zbytočný select do db. Zároveň si držím aj counter, ktorý používam pre vygenerovanie id pre t[to entitu (t.j. nenechám db vytvoriť id automaticky ako *autoincrement*). Ostatne tabuľky sú v podstate jednoduché na spracovanie a nebudem ich opisovať bližšie. Podobne ako pri *authors,* kde si udržiavam všetky id-čka v pamäti, tak si udržiavam aj všetky id-čka pre *conversations*. Jednak pre duplicitu záznamov ale aj preto, aby som mohol spracovať *conversation_references*. Túto tabuľku napĺňam úplne na konci celého importu, dátami, ktoré získam druhým prejdením súboru *conversations*. Pre každú referenciu sa pozriem do setu id-čiek *conversation*, ci už *parent_id* existuje. Ak áno, tak všetky správne referencie vložím do db. Čo sa týka UTF-8 NULL znaku použil som

```
string.replace("\x00", "\uFFFD")
```

na odstránenie tejto chyby. Samozrejme, pre entitu *links* som spravil podmienku, aby sa táto entita nevložila, ak má url väčšiu ako *constraint* stĺpca. Dáta som vkladal bulk insertom po spracovaní 100_000 záznamov a v tomto momente som aj zapísal timestamp do .csv súboru. Pre 100_000 som sa rozhodol preto, lebo mi to davalo omnoho lepšie časy, ako keď som bulk insertoval len po 10k.

Pre zrýchlenie celého importu som modely tabuliek spravil bez *constraintov* na foreign keys. Tieto pridám na konci pomocou ALTER TABLE. Integritu dát zabezpečujem kontrolovaním dát v RAM. Čiže sa mi nemôže stať, že by následne po vložení fk, databáza vyhodila chybu. A takisto som spravil aj multithreading, kde pre každú tabuľku, ktorá má v programe dostatočne veľa záznamov (t.j. veľkosť batchu), vytvorím thread so samostatným connection do db a tieto dáta vložím.

Použité technológie

Pre svoj script na import som si vybral jazyk Python. Osobne tento jazyk neobľubujem, kvôli absencii typov. Ale rad som využil jeho silu pri písaní scriptov v krátkom časovom horizonte. Zároveň som našiel veľa StackOverflow riešení na bulk insert do databázy

(https://stackoverflow.com/questions/3659142/bulk-insert-with-sqlalchemy-orm). Pre prácu s db som si vybral libku „SqlAlchemy Core. Tento spôsob podľa daného materiálu je najrýchlejší ak nerátam písanie raw query. Zvyšné libky na spracovanie timestampu, json-u a podobne som spomenul vyššie. Vytvoril som teda pár súborov na pracovanie aj s db.

- Ako *drop.py*, ktorý dropne všetky tabuľky schémy, s ktorou pracujeme.
- *Migrate.py,* ktorý zmigruje tabuľky zo súboru *models.py* bez foregin keys
- *Import.py*, ktorý importne už dane dáta a na konci pridá foreign keys

Napísané a vysvetlené každé SQL, ktoré program vykoná a zhodnotená jeho efektívnosť

Pomocou libky som v súbore *migrate.py*, zmigroval tabuľky do db. Tu sa vykonávajú príkazy ako CREATE TABLE, čo nie je časovo náročné. A v samotnom importe nepoužívam žiadne SELECTY, ktoré by spomalili import zbytočným čítaním z disku a všetky potrebne veci si držím v RAM. Jedine SQL query, ktoré používam je INSERT MANY. Kód pre SQL Alchemy v python vyzerá nasledovne:

```
connection.execute(table.insert(),data_list)
```

a tento sa retransformuje na query:

```
BEGIN (implicit)
INSERT INTO authors (name, fullname) VALUES (?, ?)
[...] (('sandy', 'Sandy Cheeks'), ('patrick', 'Patrick Star'))
COMMIT
```

Commit sa teda urobí po každom tomto insert many. Samotný batch, ktorý sa vloží pri spracovaní 100k záznamov sa mení na základe samotného obsahu json zo súboru. Čiže nie vždy bude mat batch rovnaký počet záznamov vkladaných do db naraz, ale vždy bude mať aspoň 100k. Záleží od dát. Na konci importu ešte pridávam constraint na všetky foreign keys v schéme, čo znamená, že sa všetky dáta musia skontrolovať, ale je to rýchlejšie ako keby tam boli od začiatku a mali sa kontrolovať pri každom inserte. Integrita dát je zabezpečená kontrolovaním týchto vzťahov v RAM.

Dĺžka trvania importu a časový opis priebehu

Timestampy z behu prikladám v súbore timestamps.csv po každých 100 000 spracovaných záznamoch. Môj import trval 2 hodiny a 20 minút. Importovanie samotných *authors* trvalo 5 minút, kde batch trval okolo 5-6s. Pri spracovávaní *conversations* dĺžka importu batchu sa pohybuje okolo 20 sekúnd. Tento čas veľmi skáče, podľa toho, či som pre danú tabuľku naplnil potrebných 100k na bulk insert. Pri druhom prechode, kde spracovávam iba references, je tento čas stály a pohybuje sa okolo 3-4 sekúnd. Vloženie foreign keys po úplnom importe dát trvá 11 minút.

Počet a veľkosť záznamov v každej tabuľke ako screenshot

Pre zistenie tohto udajú som si pripravil query, ktorá mi pre každú tabuľku vyráta počet záznamov aj jej veľkosť. A k týmto údajom aj veľkosť celej db a percentuálne zhodnotenie ako ďaleko sa v importe nachádzam.

```
SELECT '0' as count, pg_size_pretty( pg_database_size('pdt2') ) as size, '_size_all' as table_name UNION
SELECT count(*)::varchar(255) as count, pg_size_pretty( pg_table_size('annotations') ) as size, 'annotations' as table_name FROM annotations UNION
SELECT count(*)::varchar(255) as count, pg_size_pretty( pg_table_size('authors') ) as size, 'authors' as table_name FROM authors UNION
SELECT count(*)::varchar(255) as count, pg_size_pretty( pg_table_size('context_annotations') ) as size, 'context_annotations' as table_name FROM context_annotations UNION
SELECT count(*)::varchar(255) as count, pg_size_pretty( pg_table_size('context_domains') ) as size, 'context_domains' as table_name FROM context_domains UNION
SELECT count(*)::varchar(255) as count, pg_size_pretty( pg_table_size('context_entities') ) as size, 'context_entities' as table_name FROM context_entities UNION
SELECT count(*)::varchar(255) as count, pg_size_pretty( pg_table_size('conversation_hashtags') ) as size, 'conversation_hashtags' as table_name FROM conversation_hashtags UNION
SELECT count(*)::varchar(255) as count, pg_size_pretty( pg_table_size('conversation_references') ) as size, 'conversation_references' as table_name FROM conversation_references UNION
SELECT count(*)::varchar(255) as count, pg_size_pretty( pg_table_size('conversations') ) as size, 'conversations' as table_name FROM conversations UNION
SELECT '0' as count, (round(count(*)/32000000.0 * 100))::varchar(255) as count, '_percentage' as table_name FROM conversations UNION
SELECT count(*)::varchar(255) as count, pg_size_pretty( pg_table_size('hashtags') ) as size, 'hashtags' as table_name FROM hashtags UNION
SELECT count(*)::varchar(255) as count, pg_size_pretty( pg_table_size('links') ) as size, 'links' as table_name FROM links ORDER BY table_name
```

	count character varying	size text	table_name text
1	0	100	_percentage
2	0	30 GB	_size_all
3	19458972	1304 MB	annotations
4	5895176	902 MB	authors
5	134285948	7716 MB	context_annotations
6	88	48 kB	context_domains
7	29438	3288 kB	context_entities
8	54613745	2718 MB	conversation_hashtags
9	27917087	1801 MB	conversation_references
10	32347011	8025 MB	conversations
11	773865	40 MB	hashtags
12	11540704	1775 MB	links

Github classroom: https://github.com/FIIT-DBS/zadanie-pdt-mateju25