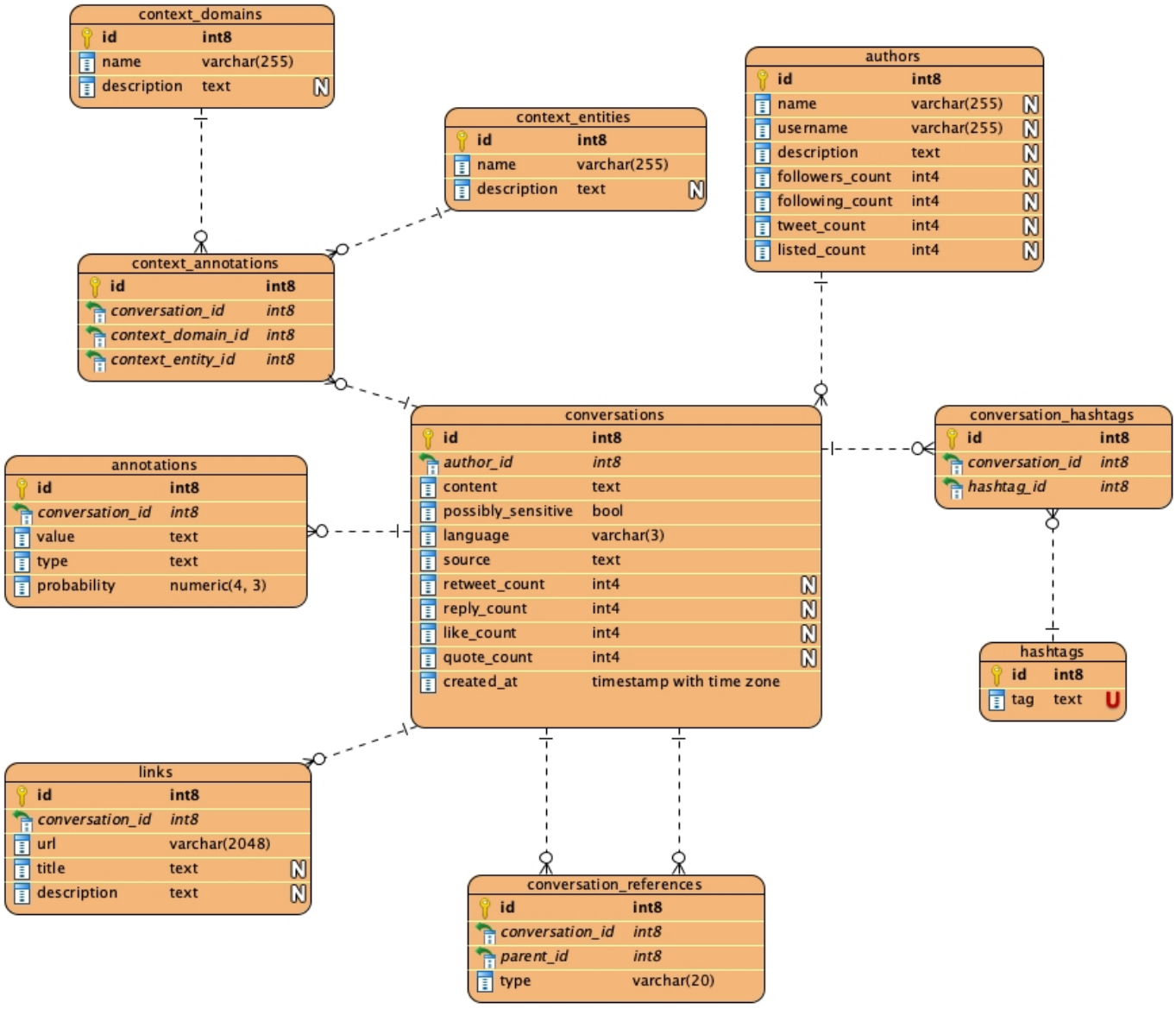
Zadanie 1 - Import tweetov do PostgreSQL

Marek Sýkora, 103141
https://github.com/Marek-FIIT/PDT_2022

Našou úlohou bolo spracovať záznamy o autoroch tweetov v súbore *authors.jsonl* (4GB) a záznamov o tweetoch samotných zo súboru *conversations.jsonl* (58GB).

Vzniknúť musí **PostgreSQL** databáza s nasledujúcou štruktúrou:



1. Opis a zdôvodnenie riešenia

Mali sme slobodu voľby prístupu aj z hľadiska voľby jazyka aj prístupu k insertovaniu do db. Rozhodol som sa riešenie postaviť na príkaze **COPY**. Ten načíta dáta zo súboru a insertne ich hromadne do databázy. Tento prístup som zvolil pretože sa jedná o najefektívnejší spôsob ako robiť bulk insert v prípade, že sú dáta predpripravené napríklad v .csv súbore.

1.1. Príprava dát

Dáta je potrebné najprv spracovať do .csv súboru čitateľného príkazom COPY. Na transformáciu (ale aj celý proces) som sa rozhodol použiť jazyk python, hlavne kvôli skúsenostiam a knižnici pydantic, ktorá umožňuje prehľadne načítavať a validovať json objekty.

Oba .jsonl súbory načítavam po riadku a postupne z nich vytváram .csv súbory. Zo súboru authors.jsonl vznikne súbor authors.csv a zo súboru conversations.jsonl vznikne ďalších 9 súborov, ktoré korešpondujú s ostatnými tabuľkami v dátovom modeli.

Tieto už obsahujú zvalidované dáta pripravené na prenos do databázy a následne sa pomocou COPY postupne importujú.

Súbor authors.csv (cca 6 000 000 záznamov) bolo možné naraz importovať do databázy. Do niektorých ostatných tabuliek však potrebujeme vložiť aj výrazne vyššie množstvo záznamov, ktoré už prekračovalo limity príkazu COPY. Preto som sa rozhodol každý .csv súbor, ktorý vytváram po 5 000 000 záznamoch zalomiť (okrem authors.csv, keďže ten sa vošiel naraz) a generovať tak súbory *table-number*.csv, ktoré je potrebné všetky importovať do databázy.

1.2. Import do databázy

Všetky SQL dopyty sú vykonávané **python** scriptom, ktorý pomocou **sqlalchemy** posiela raw SQL do lokálne bežiacej databázy.

Najprv do databázy pošlem SQL script na vytvorenie všetých tabuliek dátového modelu v takej podobe aká je požadovaná. Všetky id stĺpce v tabuľkách, okrem tabuliek *authors*, *conversations*, *hashtags*, *context_domains* a *context_entities*, sú definované ako autoincrement a identity, takže tie sa plnia automaticky pri napĺňaní týchto tabuliek.

Následne sa pomocou COPY (osodlaného ako raw SQL) do databázy importujú všetky čiastkové súbory všetkých tabuliek okrem *conversation_references*. Väzby vo forme foreign key v tabuľke *conversation_references* nie je možné validovať pri predspracovaní, takže tú nie je možné priamo importovať do cieľovej tabuľky. Na riešenie tohto problému je pri importovaní referencii vytvorená pomocná tabuľka *_conversation_references*, do ktorej sú nakopírované všetky záznamy. Z tejto pomocnej tabuľky sú následne za pomoci už naplnenej tabuľky *conversations* do cieľovej tabuľky *conversation_references* insertnuté len záznamy s platnými väzbami cez cudzie kľúče.

Keďže záznamy pred importom validujem a teda pri importe ich považujem za spŕavne a platné, samotný čas trvania importu som skrátil vypnutím triggrov na tabuľkách, teda aj kontrolovania primary key a foreign key. Tieto opäť po naplnení všetkých tabuliek zapnem (vypnutie aj zapnutie je odoslané ako raw SQL).

2. Použité SQL

Uvedené SQL je v tejto podobe posúvané sqlalchemy, ktoré každé jednotlivé volanie obalí do transakcie.

2.1. Inicializácia databázy

Vytvorenie všetkých finálnych tabuliek aj so všetkými constraints.

```
CREATE TABLE IF NOT EXISTS public.authors
(
    id bigint NOT NULL,
```

```
    name character varying(255) COLLATE pg_catalog."default",
    username character varying(255) COLLATE pg_catalog."default",
    description text COLLATE pg_catalog."default",
    followers_count integer,
    following_count integer,
    tweet_count integer,
    listed_count integer,
    CONSTRAINT authors_pkey PRIMARY KEY (id)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.authors
    OWNER to postgres;

-----------------------------------------------------------

CREATE TABLE IF NOT EXISTS public.conversations
(
    id bigint NOT NULL,
    author_id bigint NOT NULL,
    content text COLLATE pg_catalog."default" NOT NULL,
    possibly_sensitive boolean NOT NULL,
    language character varying(3) COLLATE pg_catalog."default" NOT NULL,
    source text COLLATE pg_catalog."default" NOT NULL,
    retweet_count integer,
    reply_count integer,
    like_count integer,
    quote_count integer,
    created_at timestamp with time zone NOT NULL,
    CONSTRAINT conversations_pkey PRIMARY KEY (id),
    CONSTRAINT author_id FOREIGN KEY (author_id)
        REFERENCES public.authors (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.conversations
    OWNER to postgres;

-----------------------------------------------------------

CREATE TABLE IF NOT EXISTS public.hashtags
(
    id bigint NOT NULL,
    tag text COLLATE pg_catalog."default" NOT NULL UNIQUE,
    CONSTRAINT hashtags_pkey PRIMARY KEY (id)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.hashtags
```

```
    OWNER to postgres;

-----------------------------------------------------------

CREATE TABLE IF NOT EXISTS public.conversation_hashtags
(
    id bigint NOT NULL GENERATED ALWAYS AS IDENTITY,
    conversation_id bigint NOT NULL,
    hashtag_id bigint NOT NULL,
    CONSTRAINT conversation_hashtags_pkey PRIMARY KEY (id),
    CONSTRAINT conversation_id FOREIGN KEY (conversation_id)
        REFERENCES public.conversations (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT hashtag_id FOREIGN KEY (hashtag_id)
        REFERENCES public.hashtags (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.conversation_hashtags
    OWNER to postgres;

-----------------------------------------------------------

CREATE TABLE IF NOT EXISTS public.context_domains
(
    id bigint NOT NULL,
    name character varying(255) COLLATE pg_catalog."default" NOT NULL,
    description text COLLATE pg_catalog."default",
    CONSTRAINT context_domains_pkey PRIMARY KEY (id)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.context_domains
    OWNER to postgres;

-----------------------------------------------------------

CREATE TABLE IF NOT EXISTS public.context_entities
(
    id bigint NOT NULL,
    name character varying(255) COLLATE pg_catalog."default" NOT NULL,
    description text COLLATE pg_catalog."default",
    CONSTRAINT context_entities_pkey PRIMARY KEY (id)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.context_entities
    OWNER to postgres;
```

```
-----------------------------------------------------------

CREATE TABLE IF NOT EXISTS public.annotations
(
    id bigint NOT NULL GENERATED ALWAYS AS IDENTITY,
    conversation_id bigint NOT NULL,
    value text COLLATE pg_catalog."default" NOT NULL,
    type text COLLATE pg_catalog."default" NOT NULL,
    probability NUMERIC(4,3) NOT NULL,
    CONSTRAINT annotations_pkey PRIMARY KEY (id),
    CONSTRAINT conversation_id FOREIGN KEY (conversation_id)
        REFERENCES public.conversations (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.annotations
    OWNER to postgres;

-----------------------------------------------------------

CREATE TABLE IF NOT EXISTS public.links
(
    id bigint NOT NULL GENERATED ALWAYS AS IDENTITY,
    conversation_id bigint NOT NULL,
    url character varying(2048) COLLATE pg_catalog."default" NOT NULL,
    title text COLLATE pg_catalog."default",
    description text COLLATE pg_catalog."default",
    CONSTRAINT links_pkey PRIMARY KEY (id),
    CONSTRAINT conversation_id FOREIGN KEY (conversation_id)
        REFERENCES public.conversations (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.links
    OWNER to postgres;

-----------------------------------------------------------

CREATE TABLE IF NOT EXISTS public.context_annotations
(
    id bigint NOT NULL GENERATED ALWAYS AS IDENTITY,
    conversation_id bigint NOT NULL,
    context_domain_id bigint NOT NULL,
    context_entity_id bigint NOT NULL,
    CONSTRAINT context_annotations_pkey PRIMARY KEY (id),
    CONSTRAINT conversation_id FOREIGN KEY (conversation_id)
        REFERENCES public.conversations (id) MATCH SIMPLE
```

```
            ON UPDATE NO ACTION
            ON DELETE NO ACTION,
        CONSTRAINT context_domain_id FOREIGN KEY (context_domain_id)
            REFERENCES public.context_domains (id) MATCH SIMPLE
            ON UPDATE NO ACTION
            ON DELETE NO ACTION,
        CONSTRAINT context_entity_id FOREIGN KEY (context_entity_id)
            REFERENCES public.context_entities (id) MATCH SIMPLE
            ON UPDATE NO ACTION
            ON DELETE NO ACTION
    )

    TABLESPACE pg_default;

    ALTER TABLE IF EXISTS public.context_annotations
        OWNER to postgres;

    -----------------------------------------------------------

    CREATE TABLE IF NOT EXISTS public.conversation_references
    (
        id bigint NOT NULL GENERATED ALWAYS AS IDENTITY,
        conversation_id bigint NOT NULL,
        parent_id bigint NOT NULL,
        type character varying(20) COLLATE pg_catalog."default" NOT NULL,
        CONSTRAINT conversation_references_pkey PRIMARY KEY (id),
        CONSTRAINT conversation_id FOREIGN KEY (conversation_id)
            REFERENCES public.conversations (id) MATCH SIMPLE
            ON UPDATE NO ACTION
            ON DELETE NO ACTION,
        CONSTRAINT parent_id FOREIGN KEY (parent_id)
            REFERENCES public.conversations (id) MATCH SIMPLE
            ON UPDATE NO ACTION
            ON DELETE NO ACTION
    )

    TABLESPACE pg_default;

    ALTER TABLE IF EXISTS public.conversation_references
        OWNER to postgres;
```

2.2. Kopírovanie súborov do databázy

V tejto verzii príkazu COPY špecifikujeme dabuľku, do ktorej kopírujeme, zdrojový súbor a parmetre súvisiace s týmto súborom.

```
    COPY public.conversations
    FROM 'D:\FIIT\Inzinierske_studium\1__zimny\PDT\Zadanie_1\csvs\conversations-
06.csv'
    WITH (DELIMITER '|', ESCAPE '~', FORMAT CSV, HEADER TRUE);
```

2.3. Vytvorenie pomocnej tabuľky pri importovaní *conversation_references*

Tabuľka neobsahuje ani primárny kľúč ani cudzie kľúče a po spracovaní bude zmazaná (drop table).

```
    CREATE TABLE IF NOT EXISTS public._conversation_references
    (
        conversation_id bigint NOT NULL,
        parent_id bigint NOT NULL,
        type character varying(20) COLLATE pg_catalog."default" NOT NULL
    )

    TABLESPACE pg_default;

    ALTER TABLE IF EXISTS public._conversation_references
        OWNER to postgres;
```

2.4. Zapísanie len platných záznamov do tabuľky *conversation_references*

Keď tabuľku *_conversation_references* joinneme (INNER JOIN) s tabuľkou *conversations* dostaneme len riadky v ktorých sú platné *conversation_id* a *parent_id*, ktoré môžu byť použité ako foreign key väzby. Všetky dáto z takto prefiltrovanej tabuľky *_conversation_references* môžeme insertnúť co cieľovej *conversation_references*.

```
    INSERT INTO public.conversation_references (conversation_id, parent_id, type)
    SELECT _conversation_references.* FROM public._conversation_references
    JOIN public.conversations AS conversations_1 ON
_conversation_references.conversation_id = conversations_1.id
    JOIN public.conversations AS conversations_2 ON
_conversation_references.parent_id = conversations_2.id;
```

Tento prístup má úplne identický execution plan ako napríklad alternatíva kde z *_conversation_references* vyberiem len tie záznamy kde *parent_id* aj *conversation_id*, ktoré sú obsiahnuté v *conversations.id*:

```
    SELECT * FROM public.conversation_references
    WHERE conversation_id IN (select id from public.conversations)
    AND   parent_id       IN (select id from public.conversations);
```

2.5. Enable/disable triggers

Dissable:

```
    ALTER TABLE IF EXISTS public.authors DISABLE TRIGGER ALL;
    ALTER TABLE IF EXISTS public.conversations DISABLE TRIGGER ALL;
    ALTER TABLE IF EXISTS public.hashtags DISABLE TRIGGER ALL;
    ALTER TABLE IF EXISTS public.conversation_hashtags DISABLE TRIGGER ALL;
    ALTER TABLE IF EXISTS public.context_domains DISABLE TRIGGER ALL;
```

```
    ALTER TABLE IF EXISTS public.context_entities DISABLE TRIGGER ALL;
    ALTER TABLE IF EXISTS public.annotations DISABLE TRIGGER ALL;
    ALTER TABLE IF EXISTS public.links DISABLE TRIGGER ALL;
    ALTER TABLE IF EXISTS public.context_annotations DISABLE TRIGGER ALL;
    ALTER TABLE IF EXISTS public.conversation_references DISABLE TRIGGER ALL;
```

Enable:

```
    ALTER TABLE IF EXISTS public.authors ENABLE TRIGGER ALL;
    ALTER TABLE IF EXISTS public.conversations ENABLE TRIGGER ALL;
    ALTER TABLE IF EXISTS public.hashtags ENABLE TRIGGER ALL;
    ALTER TABLE IF EXISTS public.conversation_hashtags ENABLE TRIGGER ALL;
    ALTER TABLE IF EXISTS public.context_domains ENABLE TRIGGER ALL;
    ALTER TABLE IF EXISTS public.context_entities ENABLE TRIGGER ALL;
    ALTER TABLE IF EXISTS public.annotations ENABLE TRIGGER ALL;
    ALTER TABLE IF EXISTS public.links ENABLE TRIGGER ALL;
    ALTER TABLE IF EXISTS public.context_annotations ENABLE TRIGGER ALL;
    ALTER TABLE IF EXISTS public.conversation_references ENABLE TRIGGER ALL;
```

2.6. Adding autogeneated prmary keys (unused)

V prípade, že by sme chceli dodatočne doplniť autogenerated a identity pre primárne kľúče tabuliek, ktorých id stĺpec sme mali daný, by sme museli vykonať pre každú z nich ešte takýto príkaz:

```
    ALTER TABLE public.context_entities
    ALTER COLUMN id ADD GENERATED ALWAYS AS IDENTITY;
    SELECT setval('context_entities_id_seq',  (SELECT MAX(id) + 1 FROM
public.context_entities), false);
```

3. Trvanie spracovania

Bolo úlohou logovať ako dlho trvalo insertnúť do databázy každých 10 000 záznamov. Príkaz COPY však spracuje všetky záznamy naraz (celý čiastočný súbor naraz) a teda nie je možné logovať každých N záznamov. Zároveň je potrebné brať do úvahy predspracovanie, ktoré zabera väčšinu času behu programu. Nie je teda možné povedať ako dlho trval celý proces pre každých 10 000 záznamov. Rozhodol som sa teda logovať trvanie týchto jednotlivých blokov:

Block	Current datetime	Overall durration	Block duration
authors.jsonl conversion	2022-10-05T08:32Z	01:42	01:42
conversations.jsonl conversion	2022-10-05T10:03Z	92:13	91:30
database initialization	2022-10-05T10:03Z	92:09	0:56
disabling triggers	2022-10-05T10:03Z	92:09	0:59
table: hashtags	2022-10-05T10:04Z	93:59	0:50

Block	Current datetime	Overall durration	Block duration
table: context_domains	2022-10-05T10:04Z	93:59	0:59
table: context_entities	2022-10-05T10:04Z	93:59	0:59
table: authors	2022-10-05T10:04Z	93:06	0:06
table: conversations	2022-10-05T10:12Z	101:58	07:52
table: context_annotations	2022-10-05T10:22Z	111:32	10:34
table: annotations	2022-10-05T10:23Z	112:02	01:30
table: links	2022-10-05T10:25Z	114:56	01:53
table: conversation_hashtags	2022-10-05T10:28Z	117:29	03:32
table: conversation_references	2022-10-05T10:35Z	124:48	06:19
enabling triggers	2022-10-05T10:35Z	124:43	0:54

4. Objem dát

Table	Records	Size
annotations	19458972	1721 MB
authors	5895176	1068 MB
context_annotations	134285948	10 GB
context_domains	88	64 kB
context_entities	29438	4168 kB
conversation_hashtags	54613745	3888 MB
conversation_references	27917087	2399 MB
conversations	32347011	8628 MB
hashtags	773865	88 MB
links	11540704	2022 MB