پروژه پایانی پایگاه داده (بخش ETL)

امیرحسین رجبی (۹۸۱۳۰۱۳)

نصب

```
cd etl/src
python3 -m venv env
source env/bin/activate
pip install -r requirements.txt
python main.py -h
```

نحوه استفاده

به کمک کتابخانه argparse یک رابط CLI طراحی شده است که میتوان صفحه help را به کمک دستور python main.py -h آن را مشاهده کرد. به طوری کلی ورودی های ضروری حداقل مشخصات لازم برای اتصال به پایگاه داده مبدا و مقصد است. (یعنی نام، نام کاربر پایگاه داده و رمز آن) اما در صورت محلی [۱] نبودن پایگاه داده، آدرس سرور میزبان [۲] و درگاه [۳] می توانند مشخص شوند. همچنین میتوان اسکیما [۴] که جداول در آن قرار دارند را مشخص کرد. مثلا دستور زیر یک دستور معتبر است:

```
python main.py -s db1 user1 pass1 -d db2 user 2 pass2 -ss schema1 -ds schema2 -sh host1 -sp
  port1 -dh host2 -dp port2
```

ساختار کد

کلاس های DAG و LinkedList پیاده سازی ساختمان داده های مورد نیاز هستند. همچنین کلاس DAG شامل متدی برای تبدیل گراف به topological_order است. نام کلاس DAG بهتر بود DirectedGraph میبود زیرا بررسی وجود دور در گراف در آن [۵] متد (در واقع property) انجام میشود ولی در کد ها انتظار می رود استفاده هایی که از DAG می شود در واقع گراف جهت داری بدون دور باشند. البته متد topological_order در صورت مشاهده دور آن را اعلام کرده و برنامه با نمایش پیامی مبنی بر cyclic dependencies between tables به کاربر خاتمه می یابد. در بخش باگ ها درباره این موضوع توضیح داده میشود. کلاس DB یک wrapper برای

[۱] non local or remote
[۲] host
[۳] port
[۴] schema
[۵] topological_order

استفاده از درایور psycopg2 است. البته این موضوع در کلاس pipeline تا حدودی نقض شده است زیرا انجام wrapping در این کلاس به دلیل استفاده از named cursor ها یا server-side cursor موجب پیچیدگی مضاعف میشد. علت استفاده از cursor سمت سرور دیتابیس وجود کوئری برای دریافت حجم بالایی از رکورد ها است. کلاس pipeline نیز عملیات اصلی را به کمک کلاس های بالا انجام می دهد. مانند اتصال به پایگاه داده ها، بررسی شرایط لازم برای انجام فرآیند ETL (در طراحی ما) ساخت گراف DAG و topological order به کمک کوئری ها از جداول اسکیمای pg_catalog و در نهایت بستن کانکشن به دیتابیس ها.

منطق گراف ها

گراف DAG ما براساس کلید های خارجی [۶] بین جداول است. یعنی در ابتدا رئوس گراف جداول در اسکیمای مد نظر خواهند بود. سپس یالی جهت دار از جدول A به جدول B در گراف قرار دارد اگر A حداقل یک کلید خارجی به کلید اصلی [۷] جدول B داشته باشد. (در بخش شرایط خواهیم گفت که اجازه وجود کلید خارجی به یک خصیصه یکتا [۸] نه اصلی را نمی دهیم) در صورتی که این dependency graph یک DAG باشد، یعنی هیچ وابستگی دوری [۹] بین جداول وجود ندارد. یعنی میتوان عملیات های delete را در ترتیب topological order عملیات های insert و update در عکس این ترتیب اجرا شوند. زیرا برای insert نیاز داریم تا همه کلید های خراجی معتبر باشند. پس از جدولی شروع به کار میکنیم که هیچ کلید خارجی نداشته باشد. برای حذف نیز نیاز داریم تا هیچ جدولی به رکورد مد نظر کلید خارجی نداشته باشد.

اما کدامیک بر دیگری اولویت دارند؟ میخواهیم نشان دهیم که insert و update ها را میتوان قبل از delete ها انجام داد همچنین ترتیب بین insert و update ها مهم نخواهد بود. (یعنی در **یک** رابطه مهم نیست که اول رکورد x را اضافه کنیم یا y را) فرض کنید برای insert یا update رکورد a نیاز به حذف رکورد b باشد. برای اضافه کردن a باید کلید های خارجی آن در دیتابیس موجود باشند. چون کلید خارجی باید به کلید اصلی اشاره کند (جز شرایط ما)، وابستگی های رکورد a باید یا در دیتابیس موجود باشند یا اضافه شوند، زیرا در عملیات update کلید اصلی تغییر نمیکند (طبق فرض) (اگر کلید خارجی به خصیصه های یکتا مجاز بود چنین گزاره‌ای غلط بود. زیرا در آپدیت امکان تغییر آنها وجود دارد.) اولین عملیات حذف در گراف وابستگی های رکورد a را در نظر بگیرید. پس یک insert مستقیما نیازمند یک حذف بوده است اما وابستگی یا وجود کلید خارجی است که با حذف ارضا نمیشود یا وجود رکوردی با کلید اصلی که امکان duplicate شدن آن وجود ندارد. اما در صورتی که بخواهیم رکوردی را اضافه کنیم که کلید اصلی آن در جدول موجود است و درنتیجه نیاز به حذف آن باشد، این عملیات insert نخواهد بود بلکه update است. به سادگی می توان دید که با پیمایش برعکس topological order و انجام عملیات های insert و update با هر ترتیبی در هر جدول به دلیل شرایط نامبرده (عدم تغییر کلید اصلی در آپدیت و نبودن کلید خارجی به خصیصه یکتا) دیتابیس را در حالت stable قرار میدهد. سپس با پیمایش topological order و انجام delete ها فرآیند ETL خاتمه می یابد.

[۶] foreign key
[۷] primary key
[۸] unique attribute
[۹] cyclic dependency

شرایط لازم برای انجام فرآیند ETL

۱. **نداشتن `unique constraints` و درنتیجه نبودن کلید خارجی به آنها** : به دو دلیل این شرط نیاز است. تصور کنید کلید خارجی (در رابطه A) به یک خصیصه یکتا (در رابطه B) باشد. با انجام آپدیت ها در جدول B دیگر نمیتوان مطمئن بود این کلید حذف نشود و در نتیجه امکان تجاوز `integrity` وجود دارد. همچنین وجود یک خصیصه یکتا موجب می شود تا عملیات های حذف و اضافه و آپدیت در یک رابطه تحت ترتیبی انجام شوند. زیرا امکان اضافه رکورد با مقدار تکراری در خصیصه یکتا مگر با حذف یا آپدیت رکورد پیشین وجود ندارد.

۲. **نداشتن `self reference`** : وجود چنین کلید خارجی نه تنها موجب ایجاد دور در گراف جهت دور میکند بلکه باعث میشود ترتیب `insert` ها در رابطه مدنظر به طور خاص انجام شود. تصور کنید نیاز به اضافه کردن رکوردی باشد که هنوز کلید اصلی متناظر با کلید خارجی آن وجود نداشته باشد. البته امکان حل این موضوع در کد فعیل وجود دارد. تابع `topological order` باید این دور ها رو به صورت آرایه‌ای از صفر و یک ها که یک معادل وجود خود_ارجاعی در آن جدول است در اختیار `pipeline` قرار دهد. سپس خط لوله با تشکیل یک DAG میان عملیات های `insert` و کلید های اصلی مورد نیاز آنها [۱۰] میتواند در هر رابطه جداگانه این قضیه رو مدیریت کند.

۳. **نداشتن وابستگی دوری و DAG بودن گراف** : دلیل آن واضح است ولی راه حل آن مانند حالت بالا است. (گرافی از کلید های اصلی اما با یال هایی میان جدول ها)

[۱۰] یک گراف جهت دار با رئوس کلید های اصلی که وجود هر کلید اصلی موجب انجام آپدیت یا اضافه شدن کلید اصلی دیگر میشود (یال های خروجی) و کلید های اصلی دیگری برای آپدیت یا اضافه کردن این کلید مورد نیاز هستند (یال های ورودی)