Database Project #1-1 Report

2019-18499 김준혁

핵심 모듈과 알고리즘

run.py를 짜는 데에 있어 다음과 같이 알고리즘을 구성했다.
마지막이 ';'인 온전한 query까지 받은 이후 다음과 같이 수행하도록 하였다.

```
================================================================
for query(except last element) in query_list:
        parse query
        if parsing success:
                MyTransformer.transform → if exit: exit()
        else:
                Syntax error, break
================================================================
```

구현한 내용

grammar.lark는 워드 파일에 주어진 definition에 대해 부합하도록 작성하였기 때문에 생략한다.
run.py는 크게 세 부분으로 이루어진다.

1. printPrompt/printRequest는 반복적인 prompt 및 입력받은 request에 대한 출력을 위한 함수다.

2. MyTransformer class와 내부 함수는 각 입력 query에 대해 각각의 명령을 수행하도록 하는데, 현재 PRJ 1-1에 따라 printRequest를 통해 "... Requested"를 출력한다. exit_program의 경우는 sys.exit()을 통해 종료하도록 하였다.

3. 함수로 이루어지지 않은 부분은 먼저 grammar.lark를 불러온다. 이후 while문을 돌면서 입력된 쿼리를 ';'로 스플릿하여 마지막 요소가 없거나 공백(isspace)이 나올 때까지 입력을 받은 이후, for문에서 각 부분별로 parsing을 한다. 이때 split을 통해 나온 리스트의 맨 마지막 요소는 공백이므로 무시한다. 나머지의 경우 parsing된 내용을 확인, try~except 명령을 통해 파싱이 안되는 경우 'Syntax error'를 띄우도록 한다. 이후 MyTransformer를 통해 query를 실행한다.

느낀 점 및 기타사항

질문 게시판에 답변이 나오기 전까지 입력의 형식에 있어서 '다수의 세미콜론을 포함한 쿼리가 입력될 때, 마지막 쿼리가 마무리되지 않는 경우' 에 대해서 고민을 많이 했다. 이런 경우, 앞의 쿼리가 정상적으로 구동될 때, 뒤의 입력을 추가적으로 기다려야 하는지가 애매했다. 하지만 이것에 대해 살리는 처리를 할 경우, 처리에 대한 모순점이 있다고 생각이 들었다. 앞쪽의 쿼리가 Syntax Error 가 나는 경우다. 앞쪽의 쿼리가 에러나는 경우, 뒤의 쿼리는 무시하는 것으로 되어있다. 그렇다면, 이 마무리되지 않은 입력을 살리는 쪽으로 할 경우, 에러 이후에 뒤의 입력을 무시하는 행동과는 상충되는 행동이라고 생각이 들어서 처음 제출할 때 그렇게 처리했었다.

하지만 내 생각과는 달리 query가 여러 개 들어와도 ';'로 마무리 될 때까지 입력을 받아야한다는 것을 알고 수정하게 되어 다행이라 생각한다. 내가 너무 어렵게 생각했는지, 헷갈리면 질문을 해야했다는 것에 대해 반성의 시간을 가지고, 이에 따라 코드를 수정해서 다시 올릴 수 있었다.

lark의 문법과 재귀적인 방식의 기능에 대해서는 약간 신기했다. 지금까지 알아온 재귀적인 방법들은 모두 termination 조건을 정해야하는데, 그런 조건 없이도 parsing이 잘 되는 것에 대해 여러 생각을 해보

게되었다. termination 조건에 기본적으로 공백이 포함되는건지, 혹시 parsing syntax를 이상하게 할 경우 리소스를 크게 잡아먹을 수 있는지 등에 대해 생각을 해보았다.

이번 프로젝트를 통해 그런 복잡한 쿼리에 대한 파싱과 Syntax error를 어떻게 알아내어 처리하는지를 깨달을 수 있었고, lark라는 좋은 parser를 알아내서 좋았다.