Project 1-1: SQL Parser

본 프로젝트는 Python 용 파싱 라이브러리인 Lark 를 사용하여 간단한 SQL 파서를 구현하는 것이 목표이다.

핵심 모듈

- **`grammar.lark` 파일**: EBNF (Extended Backus-Naur Form) 형식으로 SQL 문법을 정의한다. Lark API 를 사용하면 이 파일에서 정의된 문법을 기준으로 SQL 쿼리를 파싱하고 그 결과를 AST (Abstract Syntax Tree)로 변환할 수 있게 된다.
- **`run.py` 파일**: Lark 라이브러리로부터 `Lark` 와 `Transformer` 클래스를 가져와, `Lark` 클래스로는 grammar.lark 파일에서 정의된 문법을 기준으로 하는 파서를 생성하고, `Transformer` 클래스로는 이를 상속한 `SQLTransformer` 클래스를 만들어 파서에서 생성한 AST 를 처리한다. 구체적으로, `Transformer` 클래스는 쿼리의 AST 를 전달받았을 때 해당하는 구문을 탐색하며 사용자가 정의한 액션을 수행한다. 사용자가 SQL 쿼리들을 입력할 수 있고, 파서가 쿼리들을 정확히 해석하여 어떤 종류의 쿼리들이 요청되었는지를 출력한다. 만약 쿼리가 올바른 문법으로 입력되지 않았다면 Syntax error 를 출력한다.

구현 내용 및 알고리즘

- **`grammar.lark`**: 대부분 요구사항에 입각하여 구문들을 입력했다. 이때 "=" 기호는 다양한 구문의 predicate 에서 비교연산자 ("같다")로도 사용되고, UPDATE 문에서 할당연산자로도 사용되기 때문에 별도의 키워드로 명시해줄 필요가 있다. 따라서 `ASSIGNMENT: "="`로 해당 기호를 정의하고, predicate 에서 비교하는 연산이 필요할 때는 다른 비교연산자들에 해당 기호를 추가하였다.
- `run.py`
  - **`SQLTransformer` 클래스**: Lark API 의 `Transformer` 클래스를 상속하며, 여러 종류의 SQL 쿼리를 식별하는 메소드를 구현했다. 쿼리별 메소드 내에서는 해당 종류의 쿼리가 요청되었다는 메시지를 반환한다. 예를 들어, `select_query` 메소드에서는 'SELECT' requested 이라는 메시지를 반환된다. 이때 쿼리는 `grammar.lark` 에서 정의한 바에 따르면 command 내의 `query_list` 내의 query 에 속하는데, `SQLTransformer` 가 쿼리를 하나씩만 처리하도록 구현했다. `SQLTransformer` 가 쿼리의 시퀀스를 처리하게 되면, 중간에 잘못된 구문의 쿼리가 들어와도 이를 무시하고 끝까지 처리하는데, 이는 요구사항에 어긋나기 때문이다.
  - **`parse` 함수**: 파서가 Lark API 의 `parse` 메소드를 사용하여 쿼리를 `grammar.lark` 에 따라 파싱하는데, 틀린 구문이면 SyntaxError 를 띄운다. 올바른 구문이면 `SQLTransformer` 인스턴스가 Lark API 의 `transform` 메소드를 사용하여 파싱된 쿼리를 클래스 내 정의된 메소드들에 따라 처리한다.
  - **`process_query_sequence` 함수**: 입력된 쿼리(시퀀스)의 오른쪽 공백을 모두 제거했을 때 입력값이 세미콜론으로 끝나는지 확인하고, 그렇지 않을 경우 입력값을 추가로 받으며 이 과정을 반복한다. 이후 쿼리(시퀀스)를 세미콜론을 기준으로 분할한 뒤 각 쿼리의 앞뒤 공백을 제거하고 세미콜론을 덧붙여 각 쿼리 구문을 완성한다.
  - **`main` 함수**:
    1. `grammar.lark` 파일에서 정의된 문법을 기준으로 `Lark` 로부터 파서를 생성한다.
    2. `SQLTransformer` 의 인스턴스를 생성한다.
    3. `exit` 변수가 참이 아닐 때,
       i. 쿼리(시퀀스)를 입력받고 `process_query_sequence` 함수에서 이를 쿼리의 목록으로 처리한다.
       ii. (목록 내 각) 쿼리에 대해 `parse` 함수를 실행하여 쿼리 종류를 식별한 메시지를 받는다.
       iii. 메시지가 "exit"일 경우에는 `exit` 변수를 거짓으로 바꾸고 루프를 종료하고, 그렇지 않을 경우에는 메시지를 출력한다.
       iv. `parse` 함수를 실행하는 과정에서 틀린 구문이 발견될 경우 에러 메시지를 출력하고, 쿼리들이 남아있을 경우에는 더이상 처리하지 않는다.
       v. A 부터 반복한다.

느낀 점 및 기타사항

프로그래밍 언어 문법을 정의하고 이것에 맞게 파싱하는 방법을 실습하는 것은 처음이어서 신선했다.