**2023 Fall** 데이터베이스 프로젝트 **1-3** 레포트

2018-15001 이수혁

**0.** 개발 환경 및 라이브러리

- Python 3.8.18 version (pyenv 가상환경 사용)
- lark 라이브러리 사용 (Lark, Transformer class)
- berkeleyDB 18.1.6 version 라이브러리 사용
- json 모듈 사용

**1.** 핵심 모듈과 알고리즘에 대한 설명

PROJECT 1-1)
MyTransformer class : lark Transformer class를 상속하여 다양한 쿼리 별 처리 로직을 구현
print_with_prompt : 프롬프트 메시지(DB_2018-15001)를 포함한 출력 구현
split_input_include_semicolon : 세미콜론을 기준으로 쿼리 시퀀스를 구분하여 리스트 형식으로 반환
input_until_semicolon_followed_enter : 프롬프트 입력 함수 구현, \r, \n 등의 개행 문자가 있는 경우에도 입력이 계속될 수 있게끔 처리, 세미콜론 입력 후 \n 입력 시 구문 종료
main : lark 라이브러리를 통해 parser를 생성하고 exit; 명령이 입력되기 전까지 쿼리 입력을 진행함

PROJECT 1-2)
create: 테이블명, 컬럼명, 제약조건을 인자로 받아 조건에 맞는 테이블을 생성하여 저장합니다. “table_list” 값에 테이블명을 추가하며 “table_list/{table_name}/…”에 테이블 메타데이터 및 컬럼 정보를 저장합니다.
drop: 테이블명을 인자로 받아 관련된 데이터를 모두 삭제합니다.
explain/desc/describe: 테이블명을 인자로 받아 테이블 스키마를 출력합니다.
show: “tables”를 인자로 받아 데이터베이스 상의 모든 테이블명을 출력합니다.
insert: 테이블명 및 컬럼 값을 인자로 받아 해당 테이블에 새로운 레코드를 추가합니다.
select: “*” 및 테이블명을 인자로 받아 해당 테이블의 모든 레코드를 출력합니다.

PROJECT 1-3)
insert: insert query를 파싱하여 메타데이터, 삽입할 레코드 정보로 분리합니다. 기본키 및 외래키 제약 조건을 포함한 에러 처리를 진행한 후 해당 테이블에 새로운 레코드를 추가합니다.
select: select query를 파싱하여 메타데이터, 레코드 정보, 조건절로 분리합니다. 만약 from 절에 포함된 테이블이 1개를 초과할 경우 cartesian product를 통해 새로운 레코드를 생성합니다. 알고리즘은 from ~ where ~ select 순서로 진행되며 해당 시점에 맞게 에러를 검사하여 처리합니다. 에러 처리가 끝나면 조건절 및 출력할 컬럼 정보에 맞게 필터링하여 리스트에 담은 후 print_table_select 함수를 통해 터미널에 출력합니다.
delete: delete query를 파싱하여 메타데이터, 조건절로 분리합니다. 외래키 제약 조건을 포함한 에러 처리를 진행한 후 해당 테이블에서 조건에 맞는 레코드를 삭제합니다.
print_table_schema: desc, explain, describe 쿼리의 결과 출력 형식을 맞추기 위해 해당 함수들에서 호출됩니다. 테이블명, 컬럼명, database를 인수로 받아 출력 문자열을 반환합니다.
print_table_select: select query 결과 테이블의 출력 형식을 맞추기 위해 select_query 함수에서 호출됩니다. 컬럼명, 레코드, 컬럼 문자열 최대 폭을 인수로 받아 출력 문자열을 반환합니다.
check_predicate_condition: where 절에서 파싱된 predicate과 레코드를 인수로 받아 조건을 검사한 후 boolean 값을 반환합니다.
print_with_prompt: 주어진 문자열을 prefix와 함께 단순히 출력하던 기존의 방식에서 output list를 입력으로 받아 boolean factor에 따라 prompt 동반 출력 여부를 결정하는 방식으로 수정하였습니다.

**2.** 구현한 내용에 대한 간략한 설명

PROJECT 1-1) 먼저 main 함수 내에서 parser를 생성하고 while 문을 통해 exit 명령이 들어오기 전까지 사용자 입력을 계속 받습니다. 입력된 쿼리는 입력 처리 함수를 이용해 세미콜론을 기준으로 문자열 형태로 나뉘어지고, 이는 Mytransform 클래스의 transform 함수를 통해 처리됩니다. 이때 try ~ except 문을 통해 정상적으로 파싱되지 않았을 경우 오류 메시지를 출력하도록 합니다.

PROJECT 1-2) transform 함수를 통해 각각의 쿼리 파싱 함수로 트리가 전달되면 주어진 트리를 알맞게 파싱하여 지정된 동작을 수행합니다. 또한 main 함수 내에서 프로그램 시작 시 데이터베이스 저장 경로를 설정하고 이를 transformer 생성에 반영합니다. SQL 명령에 따른 동작이 실행된 후 동작 완료 메시지를 반환하며 이를 쿼리 별로 모아 출력합니다. 만약 SQL이 제약조건에 의해 정상적으로 실행될 수 없다면 주어진 메시지 유형에 따른 에러 메시지를 출력합니다.

berkeleyDB를 활용해 키-값 쌍으로 테이블의 메타데이터 및 레코드 정보를 저장합니다. 하나의 DB 내에 모든 테이블 스키마를 관리하며 JSON 모듈을 활용해 리스트나 딕셔너리 형태의 정보를 바이트 시퀀스로 변환해 DB에 저장합니다.
주어진 오류 메시지 이외에 다음 두 가지 오류 유형 및 메시지를 정의하여 사용하였습니다.
SelfReferenceError | 같은 테이블 내의 컬럼을 참조할 때
"Create table has failed: foreign key references itself"
SelectDBExistenceError | select 명령 시 데이터베이스 파일이 존재하지 않을 때
"Selection has failed: database does not exist"

PROJECT 1-3) insert, select, delete 모두 쿼리 파싱, 에러 처리, 명령 수행의 순서로 이루어집니다. 조건에 맞는 에러가 발생한 경우 transformer의 result 애트리뷰트에 리스트 형태로 결과 문자열을 담아 early return을 진행합니다. result[0]은 prompt와 함께 출력하는지 여부를 결정하는 boolean factor로, desc, explain, describe 및 select query의 결과물은 해당 값이 false에 해당합니다. 디버깅의 편의성을 위해 데이터베이스에 값을 저장하는 함수의 경우 "things to put in database" 주석에 해당 값들을 표시해 두었습니다. 테이블 별로 레코드는 고유 식별자(uuid)로 구분하며 테이블 메타데이터에 현재 존재하는 레코드들의 uuid 정보가 존재합니다. insert, select, delete 모두 해당 uuid를 통해 레코드에 접근하여 명령을 실행합니다. 구현 스펙 및 추가 과제에 해당하는 에러 및 명령 수행 로직을 모두 포함하였습니다.

**3.** 프로젝트를 하면서 느낀 점 및 기타사항

PROJECT 1-1) EBNF 문법이 생소했으나 프로젝트를 진행하며 익숙해지게 되어 앞으로 유용하게 쓸 수 있을 것 같습니다. lark 라이브러리에서 제공되는 tree 및 token 객체에 대해 잘 알지 못하는데 앞으로의 프로젝트에서 어떻게 사용할 수 있을 지 기대됩니다. 또한 제공해 주시는 테스트 케이스가 조금 더 많다면 디버깅에 큰 도움이 있을 것 같아 혹시 가능한지 여쭤봅니다.

PROJECT 1-2) lark tree를 인덱싱하는 과정에서 많은 어려움을 겪었습니다. 트리 파싱에 대한 자세한 예시를 프로젝트 안내 시간에 조금 더 보여주신다면 큰 도움이 될 것 같습니다. 또한 berkeleyDB API에 대한 간단한 사용 예제를 라이브 코딩과 함께 보여주신다면 해당 DB 라이브러리 작동 원리에 대해 훨씬 쉽게 이해할 수 있을 것 같아 가능할지 여쭤봅니다. 직접 데이터베이스를 만드는 과정이 복잡하고 어려운 부분도 있었지만 구현된 이후의 성취감과 구현 과정에서의 재미를 동시에 느낄 수 있었습니다.

PROJECT 1-3) 주어진 과제 및 추가 과제를 구현하면서 주석 달기, 모듈화, 일관된 변수명 지정, 예외 처리 등의 중요성을 느꼈습니다. 코드 길이에 비례하여 디버깅이 어려워지는 현상을 방지하기 위해 상기 언급한 내용들을 철저히 지키는 것이 필수적이라고 생각했습니다. 또한 특정 에러 처리를 위해 데이터베이스에 여러 번 접근하는 코드를 구현하며 실제 DBMS라면 block transfer와 seek time을 줄이기 위해 어떠한 방식을 택할지 고민해 보기도 했습니다. 같은 DBMS를 구현하더라도 내부 코드의 데이터 처리 방식에 따라 throughput이 매우 달라질 수 있다고 느꼈습니다. 또한 구현해야 할 함수가 많아지면서 고려할 테스트 케이스 역시 매우 방대해진 결과 예상치 못했던 쿼리 시퀀스가 오류를 발생시킬 수 있다는 것을 느껴 테스트 주도 개발(TDD)의 중요성 역시 절감했습니다. 가능한 모든 입력을 커버할 수 있는 테스트 케이스를 사전에 준비하고 이에 맞춰 코드를 구현해 나가는 것이 디버깅의 편의성 및 시간 절약에 큰 도움을 준다고 생각했습니다.
과제 스펙에는 명시되지 않았으나 'where 1 = 1'과 같이 MySQL에서 작동하는 조건절 논리식 역시 구현하려 하였지만 시간 상 실패하였습니다. 다음 과제 수행 이전에 혹시 시간이 된다면 해당 부분도 구현해 보도록 노력하겠습니다.