데이터베이스 프로젝트 1-3

2015-16227 김혜수

1. 데이터베이스 저장 형식

   각각의 table은 db 폴더 안에 table의 이름과 같은 .db 파일로 저장된다고 가정하였습니다.
   각각의 파일 내에 (5+ column의 개수) 개의 key-value 데이터가 저장됩니다.
   각각의 key는 string을 byte로, value는 리스트 혹은 딕셔너리를 byte로 인코딩하여 저장하였습니다.

   - primary_key : primary key들의 리스트를 값으로 갖습니다.
   - foreign_key : foreign key 들의 리스트를 값으로 갖습니다.
   - columns : 각 column의 이름들의 리스트를 값으로 갖습니다
   - referenced : 이 테이블을 참조하는 다른 테이블의 이름들의 리스트를 값으로 갖습니다.
   - referencing : 이 테이블이 참조하는 다른 테이블의 이름들의 리스트를 값으로 갖습니다.
   - 각 column의 이름 : column의 정보를 가진 딕셔너리를 값으로 갖습니다.
     딕셔너리가 가진 key-value는 다음과 같습니다
     - type : column의 data type 값을 갖습니다
     - null : null 값을 가질 수 있으면 'Y', 그렇지 않으면 'N' 값을 갖습니다.
     - Key : Primary 키 혹은 Foreign 키 인지 정보를 저장합니다.

   프로젝트 1-2에서 구현했던 위 사항에 더하여, 각 데이터의 primary key 값들을 순서가 있는 리스트에 저장하여 바이트로 인코딩한 값을 key로 하고, value로는 각각의 column 값에 대한 value를 저장한 딕셔너리를 값으로 갖도록 데이터 하나하나를 저장하였습니다.

2. 알고리즘에 대한 간략한 설명

   - 프로젝트 1-1 에서 구현한 Parser와 프로젝트 1-2 에서 구현한 ddl을 사용했습니다
   - Transformer내의 create_table_query , drop_table_query, show_tables_query, desc_query, insert_query, delete_query, select_query 함수에서 각각의 쿼리에 대한 동작을 처리하도록 하였습니다.
   - Insert_query 에서는 순차적으로 테이블 존재 여부, 컬럼 수와 타입 체크, 중복 여부에 대한 에러 체크를 한 후 에러가 없다면 딕셔너리에 각각의 칼럼에 대한 값들을 저장하고, primary key의 리스트를 key로 하여 berkeley db에 저장합니다. 이 때, 해당 데이터가 참조하는 foreign key data가 있다면 참조하는 데이터에 이 데이터의 정보를 저장하여 추후에 삭제 여부를 결정할 때에 사용합니다
   - Delete_query 에서는 테이블과 where 절 내부의 칼럼 존재 여부에 대한 에러 체크를 한 후, 삭제하려는 데이터가 참조 당하는 데이터들을 방문하여 이 데이터를 삭제해도 되는지 확인합니다. 삭제해도 된다면, 삭제하려는 데이터가 참조하는 데이터와 참조 당하는 데이터 각각에 방문하여 해당 데이터의 정보를 삭제하고 이 데이터를 Berkeley db에서 삭제합니다.
   - Select_query 에서는 테이블과 칼럼의 존재 여부에 대한 에러 체크를 한 후, 명시된 테이블에서 가져올 수 있는 모든 데이터를 우선 가져옵니다. Where 절이 없다면 모두 출력하고, where 절이 존재하면 그 중 조건에 맞는 데이터만 남겨서 출력합니다.

Where 절에서 어떤 데이터가 사용될지 모르기 때문에 우선 모든 칼럼에 대한 정보를 가져온 후, 결과를 출력할 때에만 선별된 칼럼을 출력합니다.

3. 가정한 것들
   A. Select query의 결과를 프린트할 때에 파이썬의 prettytable이라는 모듈을 사용하였습니다. 채점하시는 조교님께서도 번거로우시겠지만 prettytable 모듈을 설치 후 채점해주시면 감사하겠습니다. (https://www.howtoinstall.me/ubuntu/18-04/python3-prettytable/)
   B. Select query에서 column 이름에 별칭을 사용하는 경우, 해당 별칭 앞에 테이블 명을 붙여서 사용하는 경우가 없다고 가정하였습니다.
   예를 들어,
   select col1 as C from t_name where t_name.C is null;
   위와 같은 query는 사용하지 않고, 대신 아래와 같은 query를 사용한다고 가정하였습니다.
   select col1 as C from t_name where C is null;
   select t_name.col1 as C from t_name where C is null;
   C. Select query에서 결과를 출력할 때에, column들의 이름은 테이블명.칼럼명 의 형태로 테이블명을 모두 명시하였습니다
   D. Insert query에서는 항상 해당 테이블의 칼럼 수와 같은 수의 값들이 들어온다고 가정하였습니다. 즉 nullable한 값을 명시적으로 적어주지 않으면 기본으로 null로 설정하지 않고, 들어온 값의 수가 적다고 에러를 반환합니다.
   E. 제공된 skeleton lark file에 정의된 것에 따라서 들어오는 데이터의 종류는 int와 char, date 만 있다고 가정하였습니다.
   F. 외부에서 db 폴더 내의 파일을 임의로 변경하거나 추가, 삭제할 수 없다고 가정하였습니다. Input command를 통해서가 아닌 임의로 데이터베이스 파일을 추가하거나 삭제하는 경우 에러가 발생할 수 있습니다.

4. 프로젝트를 하면서 느낀점
   이런 프로그램을 직접 짜지 않아도 사용할 수 있는 sql 프로그램이 배포되어 있음에 감사함을 느꼈습니다.
   간단한 가정들을 삭제하고 복잡한 동작들이 모두 가능하게 만든다면 아주 복잡하고 어려운 프로그래밍 과정이 될 것이라고 느꼈습니다.
   특히나 Join, Group by, sorting 등을 구현한다면 현재 제가 만든 프로그램에서는 시간복잡도가 매우 나쁘게 나올 것이라고 생각하였습니다. 현재 제가 사용하고 있는 파이션 딕셔너리에 의존한 방식이 아닌, 시간 복잡도를 최소화 하는 방식이 무엇일지 고민해보게 되었습니다. 간단한 sql 문으로 쉽게 사용하고 있는 데이터베이스를 직접 구현하는 데에는 너무 많은 수고가 든다는 것을 알게 되었습니다. 아주 잘 구현된 데이터베이스들이 상용화된 시대에 살고 있는 것에 감사하게 되었습니다.