Database Project #1-2 Report

2019-18499 김준혁

핵심 모듈과 알고리즘

1. DB 내 record의 의미

기본적으로 key-value 방식을 취하는 Berkeley DB를 Relational 하게 이용하기 위해 다음과 같이 key-value를 구성하였다. 아래 표에서 '#number'는 숫자를 의미하며, "None"은 값이 "None"이 아닌 실제 record가 없는 것을 의미한다. 여러 삽입/삭제에서 데이터들을 재배열(ex:인덱스 0~10까지 데이터에서 중간 삭제로 0~9로 align)하는 것은 비효율적이라 판단, 실제 개수와 최대 인덱스 값을 구분하였다.

Key	Value
tables	테이블 최대 인덱스
tablesnum	테이블 개수
table-#number	테이블 이름 (tablename)
tablename	존재할 경우 True, 없으면 None
tablename-columns	컬럼 최대 인덱스 = 컬럼 개수
tablename-primary	해당하는 컬럼들 (columnname[-columnname]*)
tablename-referencedby	referencing 테이블들 (tablename[-tablename]*
tablename-data	데이터 최대 인덱스
tablename-datanum	데이터 개수
tablename-#number	컬럼 이름 (columnname)
tablename-columnname	존재할 경우 True, 없으면 None
tablename-columnname-type	int / char+N / date
tablename-columnname-notnull	True / False
tablename-columnname-primary	해당 컬럼이 primary인 경우 True, 없으면 None
tablename-columnname-foreign	해당 컬럼이 foreign인 경우 없으면 None, 있으면 tablename-columnname[+tablename-column]*
tablename-columnname-#number	실제 데이터값

2. 추가한 주요 함수들 (여러 쿼리에서)

dbput(k, v) : DB에 자동으로 ascii로 byte encode된 record가 들어가도록 하는 함수
dbget(k) : DB에서 자동으로 ascii로 byte decode하여 string 타입의 value를 얻는 함수
byteTupleToStringList(data) : record의 key, value를 모두 string으로 반환하는 함수
printError(error_code) : 에러 종류에 따른 메시지를 출력해주는 함수
getTable(table_name) : 테이블 이름에 따른 테이블의 모든 데이터를 가져오는 함수
printTable(table) : 들어온 테이블 전체를 출력하는 함수

구현한 내용

1. create table

spec에서 제공된 에러 및 referencing 관계를 모두 고려하며, 위에 구성한 표에 맞게 DB에 record를 넣었다.

2. drop table

주어진 조건에 맞게 구현하였으며, drop table을 할 때 referencedby가 None일 때 가능한 것으로 정했기에, drop이 정상적으로 실행되면 다른 table의 'referencedby' 또한 수정해주는 방식으로 구현했다.

3. explain, describe, desc

외부 함수(explainDescribe(items))로 따로 빼, 세 쿼리가 동일한 기능을 수행하도록 구현했다.

4. show tables

1-1에서 이 쿼리를 잘못 이해하여 잘못 작성한 파서를 수정했고, 이번엔 기능을 올바르게 구현했다.

5. insert

기본적으로 spec에서 요구하는 내용에 따라 구현하였으며, 다만 insert into에 들어온 컬럼 이름이 중복된 경우 특별한 에러 요구사항이 없어, 에러는 아니되, 가장 마지막에 들어온 값이 들어가도록 구현했다.

6. select

getTable(table_name), printTable(table)을 이용하여 1-3에 대비할 수 있도록 구현했다.

느낀 점 및 기타사항

사실상 0차원의 점과 같은 key-value 방식의 database를 2차원적이라 볼 수 있는 relational DB로 만드는 것 자체가 과정이 복잡할 것이라 생각은 되었지만, 실제로 코드를 짜면서 많은 수정을 거쳤던 것 같다. 특히, table을 구성하기 위한 create table 문을 작성하는 과정에서 어떻게 key-value pair를 정할지에 대해 효율성을 생각하면서 수정하면 코드를 이에 따라 다시 고치고 하는 과정이 반복되어 피로를 느낄 수밖에 없었다. 다만 create table 문을 구현하고 나서는, DB 구조 자체가 fix되다보니, 나머지 쿼리를 구현하는 과정에 있어서는 큰 어려움은 없었다. 약간 걱정이 되는 부분은 1-3에서 하게 될 select 문의 where clause였다. 요구사항에 한정된 select 문을 구현하면서, 이 where clause를 1-3에서 구현하게 될 때 얼마나 큰 어려움이 있을지에 대해 두려움을 미리부터 느끼고 있었던 것이다.

질문 게시판을 참고하여 코드를 수정하는 과정에 있어서 답변이 바뀐 것은 아무래도 혼란스러울 수밖에 없었다. 본래의 답변에 합당한 코드를 짠 상태에서 이를 다시 수정하는 과정에서 꼬이는 부분이 생길지에 대한 두려움을 가질 수밖에 없었다. 다행히도, 이번에는 미리 그 답변이 번복될 가능성을 생각하여 해당 부분의 코드만 삭제하면 해결되도록 짜놓은 덕분에, 큰 수정 없이 해당 부분만 삭제해서 해결했다. 선견지명을 가진 과거의 나에게 감사의 말을 전하고 싶다.