데이터베이스 프로젝트 1-3 보고서

컴퓨터공학부
2012-11258
유근국

● 핵심 모듈과 알고리즘에 대한 설명

프로젝트1-2에선 JavaCC를 이용하여 구현된 SQL 파서에 스키마 정보를 DB에 저장 하거나 DB에서 삭제하는 부분을 구현하였다면 이번에는 실제 레코드를 DB에 저장하고 삭제, 탐색 하는 부분을 구현하였다. 모든 구현은 SimpleDBMS_Parser.jj 파일에 구현하였으며 이를 컴파일하여 자동으로 java 파일들을 얻었다.

구현한 쿼리는 insert into, delete from, select 세 가지이다. 세 가지 쿼리 각각의 세부구현은 차이가 있으나 공통된 알고리즘은 다음과 같다.

1. 사용자로부터 쿼리를 입력받는다.
2. 입력된 쿼리가 Syntax에 맞는지 우선 판단한다.
3. Syntax에 맞는 쿼리가 들어온 경우에만 쿼리에 따라 DB를 조작한다.
4. 단, 쿼리에 스펙에 명시된 에러가 있을경우 에러를 출력하며, 변경내용은 DB에 반영되지 않도록 한다. 또한 에러를 출력한 후 프로그램은 멈추지 않고 다시 입력을 받도록 한다.

DB는 스펙에 나와있는 대로 Berkeley DB 라이브러리를 활용하여 파일 형태로 내부 db폴더 안에 저장된다. Berkeley DB 라이브러리는 내부 lib 폴더에 포함하였다.

● 구현한 내용에 대한 간략한 설명

■ 본인은 이번 과제의 뼈대코드로 본인이 작성한 프로젝트 1-2 소스를 사용하였다. 본인은 프로젝트 1-2 에서 Berkeley DB의 key로 table name을, value로 직접 정의한 Table class 오브젝트를 담았기때문에 레코드를 담기위해 Table class에 records 라는 멤버를 추가하여 구현하였다.

```
class Table implements Serializable
{
    String tableName;
    ArrayList<Attribute> attributes;
    ArrayList<String> referencedBy; // name of table that reference
```

```
this table.
    ArrayList<ForeignKey> FKs;
    ArrayList<String> PK;
    ArrayList<ArrayList<String>> records;

    ...
```

■ 본인이 정의한 레코드의 타입은 ArrayList<String>이다. 해당 레코드가 포함된 테이블의 attribute 순으로 각 attribute에 해당하는 값이 String 타입으로 담긴다. 이런 레코드들이 다시 ArrayList로 모여 각 테이블의 relation(멤버 records)을 형성한다.

■ 또한 구현의 편의를 위해 직접 Pair class를 정의하였다.

```
class Pair<E,F> implements Serializable
  {
        E left;
        F right;
        Pair(E l, F r)
        {
          left = l;
          right = r;
    ...
```

■ 쿼리에 에러가 있는경우 다음과 같이 throw문으로 Exception에 메세지를 담아 던지고 가장 윗단인 main 함수에서 catch하여 에러문을 출력하도록 하였다. 또한 에러문을 출력한 후에는 프로그램이 멈추지 않도록 파서를 재실행 시켜주었다.

```
throw new Exception("Where clause try to reference non existing
                    column");

...

catch (Exception e)
{
  System.out.println(e.getMessage());
  //e.printStackTrace();
  System.out.print("DB_2012-11258> ");
  SimpleDBMSParser.ReInit(System.in);
}
```

■ Insert into

우선 insertColumnsAndSource()에서 Pair 형태로 목표 column 리스트와 value 리스트를 받아온다. column리스트가 없으면 DB에서 해당 table의 attribute 리스트를 직접 가져온다. 그 후 column리스트와 value리스트의 타입 비교, nullable 체크, primary key constraint, referential constraint 체크 등 에러가 존재하는지 확인한다.

Primary key constraint를 확인할때 해당 테이블이 primary key가 정의되지 않

은 테이블일 경우 모든 attribute를 묶은 set을 primary key로 가정하고 구현하였다.

- Delete from

  Referential integrity를 지키기 위해 우선 1.해당 테이블을 reference하는 테이블이 있는지 확인하고 2.있다면 그 테이블의 referencing attribute중 not nullable인 attribute가 있는지 확인했으며 3.현재 삭제의 대상이 된 레코드를 referencing하는, 값이 같은 레코드가 있는지 확인하였다. 이와 같이 확인 후 삭제가 가능하다면 삭제한 후 referencing attribute들의 값을 null으로 바꾸어 주었다.

- Where clause

  기본적으로 whereClause()가 레코드들의 리스트를 받아서 3 value boolean expression 값들의 리스트를 반환하도록 구현하였다. 3 value boolean expression 값들의 리스트는 where절의 가장 leaf 구조인 predicate()에서 만들어진 후 위로 올라오면서 and, or, not의 계산을 거치게 된다. 이때 unknown 값의 계산은 “not unknown = unknown”, “unknown or true = true”, “unknown or (non true) = unknown”, “unknown and false = false”, “unknown and (non false) = unknown” 의 규칙을 따라 계산하였다.

- Select

  where절의 구현은 delete와 동일하며 다만 그 적용이 delete는 조건을 만족하면 삭제인 반면 select는 조건을 만족하면 남기도록 적용하였다. select문에서 whereClause()에 넘겨주는 레코드들의 리스트는 fromClause()에서 생성된 리스트로, from절에 나타난 테이블들의 카테시안 곱의 결과이다. 카테시안 곱의 정의에 맞게 from절의 테이블중 하나라도 빈 테이블이 존재하면 결과가 빈 테이블이 나오도록 하였다.

● 가정한 것들

- From, where절의 구현에 있어서 다음과 같은 가정들을 적용하였다.
  - from절의 테이블 이름을 리네임할 경우 이전 이름은 잊고 리네임 한 이름만을 고려한다. 즉, 리네임 이전의 옛날 이름을 참조하려 할 경우 WhereTableNotSpecified 에러를 발생시켰다.
  - from절에서 리네임의 결과, from절에 이름이 같은 테이블들이 생길 경우 따로 에러라고 보지 않고 where절에서 만약 참조가 있다면 모호함 체크만 하도록 하였다.

- ◆ where절에서 스트링간의 비교는 사전순으로 앞에 나오는 스트링이 더 작은 스트링이라고 판단하였다.

■ 테이블 및 칼럼 이름들은 case insensitive 하게 구현하였으나 레코드의 값들은 case sensitive 하게 구현하였다.

● 컴파일과 실행 방법

이클립스상에서 프로젝트를 연 후 JavaCC를 통해 컴파일한다. 그후 생성된 Java파일을 통해 SimpleDBMSParser를 메인 class로 하여 실행한다.

혹은 PRJ1-3_2012-11258.jar 파일을 바로 java –jar PRJ1-3_2012-11258.jar 명령을 통해 실행할 수 있다.

● 프로젝트를 하면서 느낀 점

힘든 과제였다. 본인이 이번학기에 수행한 과제들 통틀어서 가장 오랜 시간이 걸린 과제였으며 가장 코드라인이 길어진 과제였다. 분명히 어떻게 구현해야 할지도 알고 어떤 식으로 구조화 하는게 좋을지도 충분히 구상할 수 있는 과제였음에도 불구하고 그것을 코드로 표현해 내는것이 쉽지 않았다.

이번 과제를 하면서 가장 많이 느낀것은 예외처리의 중요성이다. 보통 다른 과제들을 할때는 스펙에도 예외처리를 명시하는 경우가 많지 않고 처리한다 하더라도 꽤 일반적인 처리만 하는게 대부분인데 이번 과제에서는 실제 구현에 쏟은 정성만큼 예외처리에 정성을 쏟아야 했다. 처음에는 수많은 예외처리에 짜증이 나기도 했지만 실제 DBMS는 우리 과제보다 훨씬 정교하게, 더 심한 코너케이스에 대해서도 처리가 다 되어있을거란 생각에 불만없이 과제를 마무리 할 수 있었다.

정말 힘든 과제였지만 모든 구현을 끝마치고나니 굉장히 큰 성취감을 느낄 수 있었다. 정말 심플한 DBMS이긴 하지만 여러 테스트들을 돌리다보니 실제 DBMS 못지않게 잘 만든 프로그램이란 생각과 동시에 그걸 내가 직접 만들었다는 생각이 들어 너무 좋았다.

이번 과제에서 느낀점들을 토대로 앞으로 맡게될 더욱 큰 프로젝트들도 잘 해결할 수 있도록 하겠다.