[DB] Project 1-3 보고서

2010-12794
조 준 상

1. 핵심 모듈과 알고리즘에 대한 설명

이번 과제를 할 때, 프로젝트 2 에서 사용했던 DB 를 그대로 사용하였으며, tuple 을 저장하기 위해 따로 DB 를 추가하지 않았다. 내가 사용한 DB 의 구조는 <key-value> pair 가 <table 이름, table 객체>로 되어 있는데, table 객체에 tuple 들을 충분히 저장할 수 있다고 생각했기 때문에 table 객체에 Record 라는 멤버 변수를 추가하여 실제 tuple 데이터를 관리하였다.

이번 과제의 요구사항은 세 개의 DML 구문인 (insert, delete, select)를 구현하는 것이었다. 세 쿼리에 해당하는 동작을 효과적으로 구현하기 위해서는, 데이터 조작을 편리하게 만들 필요가 있었다. 따라서 자료구조를 다음과 같이 설계하였다. 모든 데이터는 Table 객체를 통해 접근할 수 있고, Table 객체는 Data 객체를 ArrayList 로 가지고 있다. 우리가 다루는 것이 relational database 이기 때문에, 이중(double) ArrayList 를 사용할까 고민해봤지만, 하나의 ArrayList 만으로 구현하는 데는 어려움이 없을 것 같아 하나만 사용했다. 하나만 사용하되, Table 객체가 column 에 대한 정보를 가지고 있기 때문에, column 의 개수만큼 널뛰기를 해서 순회하면 Table 의 첫 번째 줄, 두 번째 줄과 같이 행(row)별로 접근할 수 있어 구현할 때 어려움은 없었다.

까다로웠던 부분은 whereClause 에서 들어온 쿼리를 올바르게 평가하는 일이었다. 어떻게 구현할지 한참을 고민한 끝에, whereClause 의 조건문을 손쉽게 평가하기 위해 트리 자료구조를 선택했다. 구현은 다음과 같다. BooleanExpression 이라는 인터페이스를 하나 정의하고, 이를 구현하는 And, Or, Not, LeafExpression 객체를 두어, 위에서 interpret 명령어를 내리면 leaf 에서부터 evaluate 된 값이 And, Or, Not 을 거쳐 평가될 수 있도록 하였다. 이때 Leaf 는 'id=4' 또는 'name is null'과 같이 하나의 조건만을 평가하도록 구현하였다.

2. 구현한 내용에 대한 간략한 설명

이번에도 저번 과제와 마찬가지로 ErrorHandler 클래스를 적극 활용했다. 각각의 쿼리마다 ErrorHandler 클래스를 두어, 쿼리를 수행하는 도중 에러가 검출되면 errorDetected 메소드를 통해 에러를 등록하고, Parser 가 쿼리를 다 읽은 후에 에러가 등록된 게 있으면 그 에러에 해당하는 에러메시지를 출력하도록 하였다.

에러를 발생시키지 않는 올바른 쿼리가 들어올 때는 Table 객체를 중심으로 구현을 하였다. Insert 나 Delete 같은 경우는 Primary key constraint, Referential Integrity Constraint 를

잘 구현하는게 중요했기 때문에, 각각의 테이블의 primary key, foreign key 값을 효과적으로 비교하기 위해 Table 객체가 아래와 같이 많은 양의 정보를 멤버 변수로 가지고 있게끔 만들었다.

```
public class Table implements Serializable {

    String name;
    Record record;
    ArrayList<columnInfo> columnNT;
    ArrayList<String> referencedBy;
    ArrayList<String> referringTo;
    ArrayList<String> primaryKeys;
    ArrayList<String> foreignKeys;
    ArrayList<String> anotherForeignKeys;
    ArrayList<String> mappedKeys; // foreign key와 map되는 referring table의 columnName
    public int preemptDoublePrimaryKey;
```

위와 같이 primary key, foreign key 정보를 Table 객체가 가지고 있기 때문에, 필요할 때마다 테이블을 DB 에서 꺼내와 값을 비교할 수 있게끔 만들었다. Select 의 경우, 여러 개의 테이블이 fromClause 에 동시에 나오는 어려움이 있었지만, 여러 개의 테이블이 나오면 우선 이를 하나의 거대한 테이블로 합쳐서 주어진 operation 을 수행했더니 편리하게 구현할 수 있었다.

3. 가정한 것들

(1) 어떤 테이블의 char type 의 길이에 대한 제약이 있을 경우, 무조건 truncate 한 값을 비교한다. 이를 테면, char(5)인 컬럼에 dragon 과 dragonfly 라는 data 가 들어온다고 생각하면 이 두 데이터는 길이 제약으로 같은 데이터가 된다.

4. 컴파일과 실행 방법

Eclipse 환경에서 Runnable JAR File 의 형식으로 Export 하여 PRJ1-3_2010-12794.jar 파일을 생성하였다. 실행 방법은 실행하고자 하는 디렉터리에 "**mkdir db**" 명령을 통해 디렉토리를 만들고, 쉘에 "**java –jar PRJ1-3_2010-12794.jar**"를 입력하는 것이다.

5. 프로젝트를 하면서 느낀 점

이번 과제에서 짠 코드가 내가 지금껏 짜본 코드 중에서 가장 긴 코드였다는 점에서 여러가지 배운 점이 많았다. 코딩 라인 수가 2000 줄이 넘어가다 보니, 복잡도가 너무 커져서 중간에 에러가 발생할 경우 이를 찾는 게 굉장히 어려웠다. 스스로 실수를 방지하려고 하다 보니, 주석도 많이 달게 되었고 혹시 모를 에러에 대비해 try ~ catch 문을

많이 깔아 놓았으며, 또한 어떤 value 의 중간 값이 제대로 들어가는지를 확인하기 위해 System.out.print 도 많이 사용했다. 이렇게 방어적인 코딩을 하는 과정에서 의도치 않게 디버깅 실력이 많이 는 듯한 느낌을 받았다.

모듈화의 중요성도 다시 한 번 실감하게 되었다. 이번 과제에서 for 문을 굉장히 많이 사용했는데, 이러한 for 문들이 코드 상에 여러 군데에 나와 있으면 내가 짠 코드 임에도 순간 읽기 어려운 불편함이 있었다. 이를 방지하기 위해 어떤 특정 기능을 수행한다고 생각되는 하나의 코드 덩어리는 웬만하면 하나의 메소드로 묶어서 활용했다.

전역변수에 대한 고민도 하게 되었다. 전역변수를 쓰면 확실히 그때그때 인자를 넘기지 않아도 돼서 좋지만, 무턱대고 사용할 경우 코드의 효율성이 떨어진다는 것을 알게 되었다. 전역 변수 대신 메소드에 인자를 넘기고 지역 변수를 활용하면 비슷한 기능이 다음에 또 필요할 때 재활용할 수 있지만, 전역 변수를 쓸 경우 코드 재활용을 하기가 어려운 문제가 있었다. 앞으로의 코딩에서는 꼭 필요한 변수만 전역 변수로 활용하기로 다짐했다.

String 을 잘 활용해야한다는 것도 느꼈다. 이번 과제에서 String 을 굉장히 많이 사용하다 보니 그 중요성을 실감했다. Regex 를 이용해 String 을 똑똑하게 구분할 수도 있고, concatenate 를 통해 여러 자료를 하나로 합치는 것도 가능하며, 이를 split 을 이용해 다시 쪼갤 수 있다는 것도 알게 되었다. 잘만 활용하면 다양한 영역에서 효과적으로 이용할 수 있을 것 같다. String 을 조금 더 잘 활용하기 위해서는 Regex 를 좀 더 연습해야될 것 같다.

마지막으로 과제 내용(DBMS 구현)의 측면에서 1,2 번 과제보다 조금 더 현실의 DB 와 가까운 것을 만들게 되어 재미있고 유익했다. 또한, 우리가 개념적으로 이해하는 logical 한 모델과 이를 실제로 구현 하는 것 사이에는 어느 정도 괴리가 있다는 것을 다시 한 번 느끼게 되었다.