데이터베이스 프로젝트 1-2 보고서

컴퓨터공학부
2012-11258
유근국

● 핵심 모듈과 알고리즘에 대한 설명

프로젝트1-1과 마찬가지로 JavaCC를 이용하여 구현된 SQL 파서에 스키마 정보를 DB에 저장 하거나 DB에서 삭제하는 부분을 구현하였다. 모든 구현은 SimpleDBMS_Parser.jj 파일에 구현하였으며 이를 컴파일하여 자동으로 java 파일들을 얻었다.

구현한 쿼리는 show tables, drop table, create table, desc 네 가지이다. 네 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

■ 이번 과제에서 가장 중요한 부분은 Berkeley DB를 이용하여 DB를 구축할때 key와 value를 어떻게 정의할것인가 였다. 본인은 key로는 table name (String)을, value로는 직접 정의한 Table class의 오브젝트를 담았다. 또한 key의 중복을 허용하지 않았다.

■ Table class 오브젝트를 value로 담기위해 table class를 serializable로 정의 했으며 다음과 같이 entryBinding (SerialBinding) 을 사용하였다.

```
scc = new StoredClassCatalog(myClassDb);
entryBinding = new SerialBinding(scc,Table.class);
```

오브젝트 직렬화

```
entryBinding.objectToEntry(obj,data);
```

오브젝트 역직렬화

```
(Table) entryBinding.entryToObject(foundData);
```

■ Value로 담기는 Table class 및 내부 멤버 클래스 구조는 다음과 같다.

```
class Table implements Serializable
{
	String tableName;
	ArrayList<Attribute> attributes;
	ArrayList<String> referencedBy; // 해당 Table을 reference하는 table들의 name 리스트
	ArrayList<ForeignKey> FKs; // FK와 PK는 Attribute class 내부에서 구분하지않고 Table class에서 이와같이 리스트형식으로 가지고있게 하여 구분하였다.
	ArrayList<String> PK;

}

class Attribute implements Serializable
{
	String tableName; // 해당 Attribute가 포함된 테이블의 이름이다.
	String attrName;
	String type;
	Boolean nullable;

}

class ForeignKey implements Serializable  // ForeignKey가 single attribute to attribute로 정의되는것이 아니라 attribute list to attribute list로 정의되므로 이와같이 직접 class를 정의하였다.
{
	ArrayList<String> referencingAttrs;
	String referencedTable;
	ArrayList<String> referencedAttrs;

}
```

■ Syntax가 다음과 같이 primary key definition이 column definition 보다 먼저 나온경우를 에러로 처리하지 않는다. 따라서 key constraint와 column definition이 어떤 순서로 나오더라도 처리할 수 있게 만들어야 했다. 이를위해 다음과 같이 TableElementList class, ColumnDef class, TableConstraintDef class를 정의하여 우선적으로 table element를 모두 버퍼링 한 후 column definition들을 먼저 처리 하고 그 후에 table constraint들을 적용하는 방식을 사용했다.

```
class TableElementList
{
    ArrayList<ColumnDef> columnDefList;
    ArrayList<TableConstraintDef> tableConstraintDefList;

}

class ColumnDef
{
    String columnName;
    String dataType;
    Boolean nullable;

}

class TableConstraintDef
{
    String type; // "FK" or "PK"
    ArrayList<String> attrs;
    String referencedTable; // for only FK
    ArrayList<String> referencedAttrs; // for only FK

}
```

- 쿼리에 에러가 있는경우 DB가 업데이트 되지 않도록 하기위해 모든 DB update operation은 항상 에러체크가 모두 완료된 후에 이루어지도록 하였다. 또한 프로그램이 exit query로 정상적으로 끝나지 않고 비정상적으로 종료될 경우에도 DB가 저장되도록 하기위해서 DB update operation 이후에는 항상 sync()를 실행하여 DB를 저장하도록 하였다.

- Table name과 Attribute name이 case insensitive 하도록 하기위해 프로그램 내 table name, attribute name의 비교는 항상 case insensitive하게 이루어지도록 각 경우마다 함수를 만들어 사용하였다.

  tableExist(), getTableFromDB(), containsCaseInsen() 등

- Show tables

  Cursor.getFirst(), cursor.getNext() 함수를 이용하여 간단히 DB의 모든 레코드를 한번 훑어가며 key(table name)를 출력하도록 하였다.

- Desc

  해당 테이블에대한 객체를 DB에서 get 해온 후 Table 객체에 저장된 attribute 정보들을 출력토록 하였다. 이때 출력은 printf를 이용하여 column name, type, nullable, key 순으로 19,10,10,10 칸의 문자 포멧에 맞추어 출력되도록 하였다.

- Drop table

  간단히 해당 테이블의 database entry를 DB에서 찾아온 후 삭제하도록 하였다. 이때 해당 테이블이 reference 하고있던 테이블들의 referencedBy 리스트에서 해

당 테이블을 지워주어 더이상 reference 하지않음을 반영토록 하였다. 또한 DB에서 삭제가 일어난 후에도 sync()를 호출해 즉시 파일에 저장되도록 하였다.

- Create table

  새로운 Table 객체를 만들어 table element들을 읽고 column들을 생성한 후 table constraint를 적용하는 순으로 구현하였다. 완성된 table객체는 입력된 table name을 key로 하여 DB에 저장된다.

  Table element들을 읽고 column들을 생성, table constraint를 적용하는 과정에서 모든 에러 케이스들을 체크하게 된다.

  만일 쿼리를 처리하는 과정중 에러를 발견하게되면 Exception에 스펙에 정의된 에러 메시지를 담아 throw를 통해 던지게 된다. 던져진 exception은 바깥의 catch 문에서 잡히며 catch문에서는 exception 메세지를 출력한 후 reinit()을 통해 새로운 쿼리 입력을 받게된다.

가정한 것들

- 한 column이 여러 table의 attribute을 reference 하는경우, 즉, 같은 column에 대한 foreign key 정의가 여러번 나왔을 경우 스펙에서는 에러로 처리하지 않았지만 일반적인 DBMS에서 충분히 에러로 볼 수 있는 경우이므로 자체적으로 에러처리를 해주었다. 이 경우 출력되는 메세지는 "Create table has failed: the column references multiple tables" 이다.

- 스펙문서에 나와있는

  "Primary key로 지정된 컬럼은 따로 not null로 지정하지 않더라도 자동적으로 not null인 것으로 본다."

  "null 값을 가질 수 있는 컬럼은 primary key가 될 수 없다."

  라는 제약의 해석을 첫번째 문장의 의미에 맞추었다. 첫번째 문장의 의미를 not null로 지정하지 않은, 즉 nullable한 (기본값이 nullable이므로) 칼럼을 primary key로 지정하면 not null로 자동으로 설정되어야 한다는것으로 해석하고, 두번째 문장의 의미를 두번째 문장의 의미를 nullable한 column은 primary key가 될 수 없다는 것으로 해석하면 두 문장이 서로 모순을 만들기 때문에 본인은 첫번째 문장에 맞추어 primary key로 지정된 칼럼은 이전의 nullable값에 상관없이 not null로 덮어쓰도록 하였다. 즉, not null로 따로 지정하지 않아 nullable한 칼럼이 primary key가 될 수 있도록 하였다.

● 컴파일과 실행 방법

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

혹은 PRJ1-2_2012-11258.jar 파일을 바로 java –jar PRJ1-2_2012-11258.jar 명령을 통해 실행할 수 있다.

● 프로젝트를 하면서 느낀 점

본인이 했던 과제들 중 손꼽을 정도로 잘 수행하지 못한 과제였다고 생각한다. 처음 구현을 시작하기 전에 구조에 대한 설계를 치밀하게 해놓고 시작했어야 했는데 그렇지 못했던게 나중에 너무 큰 고통으로 다가왔다. 에러 케이스를 처리하는 과정에서 계속해서 하나씩 구조에 더 필요한것이나 바꾸어야할 점들이 드러났고 그것을 수정하는데 너무 많은 시간을 투자해야 했다. 특히 제일 마지막에 수정한 case insensivity 같은 경우는 앞서 지금까지 어디어디에서 table name과 column name을 비교했는지 알 수없을 정도로 많이 진행된 상태였기때문에 더욱 어려웠다. 처음부터 처리해야하는 에러케이스들을 쭉 한번 훑고 구조를 정하고 시작했더라면 하는 아쉬움이 크게 남는다.

힘든 과제였지만 교훈도 얻었고 그래도 벌써 simple DBMS의 절반이상을 완성했다는것이 위안이 된다. 이번에 얻은 교훈을 토대로 마지막 남은 프로젝트를 잘 해낼 수 있도록 하겠다.