DB2017

Project 1-3 : Implementing DML

2012-11249 컴퓨터공학부 설재완

1. 핵심 모듈과 알고리즘에 대한 설명

Proj 1-2 와 마찬가지로 <String, String>을 key-value pair 로 갖도록 DB 를 이용하였다. 조교님이 제공해주신 reference code 위에서 작성하였다. 가장 많이 사용하고 주로 사용한 자료구조는 ArrayList<String>과 StringTokenizer 를 이용한 String split 이다. 특히 1-2 에 비해서 추가된 점은 record 를 저장할 때, key 를 "<table name> @records"으로 하여 record 를 저장하였다.

또한 helper function 으로 records query(특정 table 의 record 를 모두 보여주는 query), deleteat query(특정 table 의 특정 index 의 record 를 지워주는 query)를 이용하였다.

```
int RecordsQuery() :
{ String tblName; }
{
  < RECORDS > tblName = TableName()
  {
    showRecords(tblName);
    return 500;
  }
}

void showRecords(String tblName) :
{}
{
  {
    Vector<String> records = myDB.getDB(tblName + " @records");
    System.out.println(tblName + "'s records start");
    for(int i=0; i<records.size(); i++) {
      System.out.println(records.get(i));
    }
    System.out.println("#of records: " + records.size());
    System.out.println(tblName + "'s records end");
    return;
  }
}
```

```
int DeleteatQuery() :
{
  String tblName;
  Token t;
}
{
  < DELETE_AT > tblName = TableName()
  t = < INT_VALUE >
  {
    deleteAt(tblName, t.image.toLowerCase());
    return 500;
  }
}

void deleteAt(String tblName, String index) :
{}
{
  {
    int deleteIndex = Integer.parseInt(index);
    Vector<String> records = myDB.getDB(tblName + " @records");
    myDB.deleteDB(tblName + " @records");
    for(int i=0; i<records.size(); i++) {
      if(i != deleteIndex) {
        myDB.putDB(tblName + " @records", records.elementAt(i));
      }
    }
    System.out.println("delete " + deleteIndex + "th row completed");
  }
}
```

2. 구현한 내용에 대한 간략한 설명

조교님 제공 코드와 Grammar.docx 를 참고하여 작성하였다. Insert, delete, select 각각의 query 는 먼저 오류가 있는지 확인한 후 오류가 없다고 판단되면 실제 insert, delete, select 를 수행하게 하였다. 다음의 그림에서 확인할 수 있다.

```
< INSERT_INTO > tblName = TableName()
{
  if (CheckNoSuchTable(tblName))
  {
    NoSuchTable();
  }
}
colAndSrc = InsertColumnsAndSource()
{
  if ((colName = CheckInsertColumnExistenceError(tblName, colAndSrc)) != null) {
    System.out.println("Insertion has failed: '" + colName + "' does not exist");
    handleDBError(parser);
  }
  else if (CheckInsertTypeMismatchError(tblName, colAndSrc)) {
    System.out.println("Insertion has failed: Types are not matched");
    handleDBError(parser);
  }

  changedColAndSrc = changeColAndSrc(tblName, colAndSrc);
  if ((colName = CheckInsertColumnNonNullableError(tblName, colAndSrc)) != null) {
    System.out.println("Insertion has failed: '" + colName + "' is not nullable");
    handleDBError(parser);
  }
  else {

    if(CheckInsertDuplicatePrimaryKeyError(tblName, changedColAndSrc)) {
      System.out.println("Insertion has failed: Primary key duplication");
      handleDBError(parser);
    }
    else if(CheckInsertReferentialIntegrityError(tblName, changedColAndSrc)) {
      System.out.println("Insertion has failed: Referential integrity violation");
      handleDBError(parser);
    }
  }

  finalInputString = truncate(tblName, changedColAndSrc);
  myDB.putDB(tblName + " @records", finalInputString);
  return 5;
```

Insert 에서는 총 6 가지의 error 를 체크하며 특히 column name 이 존재하는 지를 type mismatch 보다 먼저 확인하게 하였다. 이는 column name 이 존재하지 않을 경우도 사실은 type mismatch 의 경우에 포함될 수 있기 때문이다.

```
{
  if (CheckNoSuchTable(tblName))
  {
    NoSuchTable();
  }
}
(
  bve = WhereClause()
)?
{
  ArrayList<String> tblNames = new ArrayList<String>();
  tblNames.add(tblName);
  // delete later start
  //if(bve != null) { bve.print(); }
  // else { System.out.println("no bve"); }
  // delete later end

  if(CheckWhereTableNotSpecifiedError(tblNames, bve)) {
      System.out.println("Where clause try to reference tables which are not specified");
      handleDBError(parser);
  }
  else if(CheckWhereColumnNotExistError(tblNames, bve)) {
      System.out.println("Where clause try to reference non existing column");
      handleDBError(parser);
  }
  else if(CheckWhereAmbiguousReferenceError(tblNames, bve)) {
      System.out.println("Where clause contains ambiguous reference");
      handleDBError(parser);
  }
  else if(CheckWhereIncomparableError(tblNames, bve)) {
      System.out.println("Where clause try to compare incomparable values");
      handleDBError(parser);
  }

  realDelete(tblNames.get(0), bve);
  return 6;
}
```

Delete 에서는 총 5 가지 에러를 먼저 확인하였으며, 그 이후 실제 delete 가 진행되도록 하였다. DB 의 data 에서 직접 레코드를 지우기는 어려워서 일단 record 를 다 꺼내고, record 중 일부(delete 하지 않을 record)만 다시 넣어주었다.

```
if((s = CheckSelectTableExistenceError(tbe)) != null) {
  System.out.println("Selection has failed: '" + s + "' does not exist");
  handleDBError(parser);
}
else if((s = CheckSelectColumnResolveError(scl, tbe)) != null) {
  System.out.println("Selection has failed: fail to resolve '" + s + "'");
  handleDBError(parser);
}
else if(CheckWhereTableNotSpecifiedError(tblNames, bve)) {
    System.out.println("Where clause try to reference tables which are not specified");
    handleDBError(parser);
}
else if(CheckWhereColumnNotExistError(tblNames, bve)) {
    System.out.println("Where clause try to reference non existing column");
    handleDBError(parser);
}
else if(CheckWhereAmbiguousReferenceError(tblNames, bve)) {
    System.out.println("Where clause contains ambiguous reference");
    handleDBError(parser);
}
else if(CheckWhereIncomparableError(tblNames, bve)) {
    System.out.println("Where clause try to compare incomparable values");
    handleDBError(parser);
}

// selece start
realSelect(scl, tbe);
return 4;
```

Select 에서는 총 6 가지 에러를 확인하였으며, 4 가지는 delete 에서와 마찬가지로 where 절에 대한 것이다. delete 와 select 는 먼저 대상이 되는 records 를 정한 후 각각의 record 에 대해 where test 를 실행하였다.

Where 절의 참, 거짓 여부를 확인하기 위해 BooleanValueExpression, BooleanTerm , BooleanFactor, BooleanTest class 를 정의하였으며, 가장 바닥인 Predicate 부터 tree 형태로 올라와서 계산하도록 하였다. 특히 unknown 에 대하여 처리하기 위해 Boolean type 으로 계산하지 않고 string type 으로 "true", "unknown", "false"를 넘겨 준 후, 최종 결과가 "unknown"일 경우 "false"로 처리하였다.

3. 가정한 것들

메일로 문의한 결과, as 구문은 고려하지 않아도 된다고 하셔서 as 와 관련된 것을 구현하지 않았다.

조교님이 제공해주신 코드를 그대로 사용하였는데, 이 때, create table query 를 수행할 때, primary key 를 반드시 '잘' 지정해 준다고 가정하였다. 이로써 insert 수행 시, primary key columns 에 대한 uniqueness 를 확인할 수 있었다.

Foreign key 는 다른 table 의 primary key(primary key 가 여러개 있을 경우 모두)와 일치하도록 하였다.

Char stirng 의 길이를 계산할 때, quote 를 제외한 길이를 생각하였다.

Select 절에서 *로 projection 을 할 때, 겹치는 column name 이 있다면, 그 table 에 한해서 table name 을 출력하게 하였다.

4. 컴파일과 실행방법

Proj-1, Progj-2 와 마찬가지로 SimpleDBMSGrammar-2.jj 파일을 수정하면 같이 수정되는 파일 중 SimpleDBMSParser.java 파일이 있다. 이곳에 main 함수가 있으며, 기존의 java 프로젝트와 마찬가지로 RUN 버튼을 통해 컴파일 + 실행을 할 수 있다.

Executable jar 파일로 실행하는 경우, 커맨드라인에서 java –jar PRJ1-3_2012-11249.jar 명령어로 실행할 수 있다.

5. 프로젝트를 하면서 느낀 점

Delete 절과 select 절이 비슷한 부분이 많아서 둘다 처리할 수 있도록 where 구문을 작성하려 하다가 시간을 많이 사용하였다. 천천히 한다면 그렇게 생각해서 하는 것이 좋으나, 학기말이라는 시간적 제약때문에 결국 비슷한 역할을 하는 코드를 다소 여러번 사용한 점이 아쉽다.

또한 어떤 경우에도 잘 돌아가도록 robust 하게 코드를 짜는 것이 중요하다고 하여 시도하였으나, 역시 시간의 제약때문에 성공하지 못했다. 특히 굉장히 지저분하게 짰음에도 불구하고 소모한 시간이 너무 커서 아쉬웠다. 혹시라도 시간적 여유가 있다면 다시 한번 정갈스럽게 코드를 작성해보고 싶다.

mysql 을 사용한 경험이 부족하여 특정 상황에 대하여 어떻게 돌아가는지를 명확히 판단할 수 없어서 고민이 많이 됐다. 그래서 가능한 한 가정으로 해결하려 하였다.