Database Project 1-2

2013-11408 신동빈

1. 핵심 모듈 및 알고리즘

버클리 DB의 경우 주어진 자료처럼 key - value를 String - String으로만 저장 가능한 줄 알았으나, byte[] 형태로 변환할 수 있을 경우, class 파일을 정의해서 value값에 class 파일을 넣을 수 있었다.

```
EnvironmentConfig tableDbEnvConfig = new EnvironmentConfig();
EnvironmentConfig tableClassEnvConfig = new EnvironmentConfig();
tableDbEnvConfig.setAllowCreate(true);
tableClassEnvConfig.setAllowCreate(true);
tableDbEnv = new Environment(new File("db/"), tableDbEnvConfig);
tableClassEnv = new Environment(new File("db/"), tableClassEnvConfig);

//Open Database or if not, create one.
DatabaseConfig tableDbConfig = new DatabaseConfig();
DatabaseConfig tableClassConfig = new DatabaseConfig();
tableDbConfig.setAllowCreate(true);
tableDbConfig.setSortedDuplicates(true);
tableClassConfig.setAllowCreate(true);
tableClassConfig.setSortedDuplicates(false);
tableDb = tableDbEnv.openDatabase(null, "tableDb", tableDbConfig);
tableClass = tableClassEnv.openDatabase(null, "tableClass", tableClassConfig);

DatabaseEntry value = new DatabaseEntry();
DatabaseEntry newTableName = new DatabaseEntry(tableName.getBytes("UTF-8"));
//to save and load in Berkley DB for class Table
StoredClassCatalog tableClassCatalog = new StoredClassCatalog(tableClass);
EntryBinding bind = new SerialBinding(tableClassCatalog, Table.class);
```

따라서 위와 같이 tableClass라는, tableDb를 byte 단위로 넣기 위한 Database를 만든 후, StoredClassCatalog와 EntryBinding이라는 클래스를 이용해 직접 정의한 class 파일을 value값에 넣을 수 있도록 했다. (여기서 tableClass에 해당하는 config는 duplicate를 false로 설정해 주어야 한다.)

```
class Table implements Serializable{
    private static final long serialVersionUID = 201311408L;
    //primary key, foreign key list.
    ArrayList<String> pkList;
    ArrayList<String> fkList;

    //same attribute has same factor. columnList[i], dataTypeList[i], nullableList[i] is one set.
    ArrayList<String> columnList;
    ArrayList<String> dataTypeList;
    ArrayList<String> nullableList;

    public Table(ArrayList<String> pkList,
    ArrayList<String> fkList,
    ArrayList<String> columnList,
    ArrayList<String> dataTypeList,
    ArrayList<String> nullableList) {
        this.pkList = pkList;
        this.fkList = fkList;
        this.columnList = columnList;
        this.dataTypeList = dataTypeList;
        this.nullableList = nullableList;
    }
}
```

2. 가정 사항 및 구현

여러 가정 사항은 주어진 파일을 그대로 적용하였다.

```
//return columnDefinition String like "*colDef:(columnName):(dataType):(nullable)"
String columnDefinition() :
{
  String colDef;
  String cn;
  String dt;
  String nullable = "Y";
}
{
  cn = columnName()
  dt = dataType()
  (
    < NOT_NULL >
    {
      nullable = "N";
    }
  )?
  {
    colDef = "*colDef:" + cn + ":" + dt + ":" + nullable;
    return colDef;
  }
}
```

```
//return pk_list like "*pkConstraint:(PK1):(PK2):...:(PK_N)"
String primaryKeyConstraint() :
{
  String cnList;
}
{
  < PRIMARY_KEY >
  cnList = columnNameList()
  {
    return "*pkConstraint" + cnList;
  }
}

//return fk_list like "*fkConstraint:(fk1):(fk2):...:(fk_n):*ref:(tableName):..."
String referentialConstraint() :
{
  String fkList;
  String refList;
  String tn;
}
{
  < FOREIGN_KEY >
  fkList = columnNameList()
  < REFERENCES >
  tn = tableName()
  refList = columnNameList()
  {
    return "*fkConstraint" + fkList + ":*ref:" + tn + refList;
  }
}

//return columnNmaeList like ":(cn1):(cn2):...:(cn_n)"
String columnNameList() :
{
  String cnList = "";
  String cn;
}
{
  < LEFT_PAREN >
  cn = columnName()
  {
    cnList += ":" + cn;
  }
  (
    < COMMA >
    cn = columnName()
    {
      cnList += ":" + cn;
    }
  )*
  < RIGHT_PAREN >
  {
    return cnList;
  }
}
```

특징적인 구현 부분으로는, pk와 fk, 그리고 column Definition을 String으로 전달하기 위해, 위와 같이 한 String으로 보내는 방법을 다음과 같이 정의하였다.

ex) create table account(
primary key(account_number, branch),
account_number int not null,
branch char(13),
tmp char(5)
);

의 경우, column definition은 *colDef:account_number:int:N
primary key는 *pkConstraint:account_number:branch
foreign key는 *fkConstraint:column1:column2:...:columnN:*ref:refTableName:....
와 같은 방법을 사용한 뒤, java 파일에서 split을 통해 분류해서 정리했다.

3. 컴파일 및 실행 방법

   eclipse의 경우 jj파일을 저장하면 자동으로 패키지에 java 파일들이 컴파일 후 생성되며, jj파일 안에서 정의한 class명의(dbms라 가정)에 따라 dbmsParser라는 java파일에서 테스트를 해 볼 수 있다.

4. 느낀 점

   처음에 key - value 삽입을 오직 String to String으로만 하려 했다면 Project 1-2 뿐만 아니라 이후 데이터 삽입/삭제 등을 구현해야 하는 Project 1-3에서 엄청난 노가다 작업이 필요했을 것을, 빠르게 Serializable에 관한 정보를 얻어 이렇게 구현한 것에 대해 다행이라는 생각이 들었다. 또한, 이번 프로젝트를 할 때는 시간이 없어서 구현해보지는 못했지만, 가이드에서 지정해 준 에러 이외에도 생각해 볼 에러가 순간순간 번뜩 지나감을 느꼈는데 이후 시간이 된다면 그 부분을 구현해보고 싶다.