Project 2 Report

1. Overview

In project 1, we've already finished ***RecordBasedFileManager*** component, which provides record paged file access facilities. On top of that, we are going to build Relation Manager, which could provide tuple read/write operations in terms of table name.

2. Design & Implementation

- ***Tombstone***

One method remaining in RBFM component requiring some attention is updateRecord(). When this may make new content's length exceeding previous hold space, we have to move it to other places, but leaving a tombstone, a forward pointer there.

Our solution to differentiate this is to use record_length field in the entry of each record as the indicator,

1. ***record_len[i] > 0***, the slot points to the immediate address
2. ***record_len[i] == 0***, the slot is empty
3. ***record_len[i] < 0***, the slot is a tombstone, the forward address (pageNum, slotID) = (-record_len[i], offset[i])

Of course, the reading/deleting record operation in Project 1 needs corresponding modification: first find immediate address by following along the tombstone.

- ***Catalog & Attributes Metainfo***

For each table, we have data like this: `table_name [field_1(name, type), field2(name, type)... ]`. We use catalog and attributes two file to store these information. Catalog and Attributes themselves are also two table with following definitions:

```
Catalog (table-id, table-name, file-name)
Attributes (table-id, column-name, column-type, column-length)
```

For simplicity concern, we use the name of table as the name of corresponding file name. Of course, to make it flexible, instead of hardcoding that, we use Macro to do conversion.

- ***Methods Implementation***

If we look at most methods in Relation Manager, we could easily find each one we could find corresponding mirror in RBFM, only different is the parameters. Eg:

```
RC Relations Manager::readTuple(const string &tableName, const RID &rid, void *data);

RC RecordBasedFileManager::readRecord(FileHandle &fileHandle,
                                      const vector<Attribute> &recordDescriptor,
                                      const RID &rid, void *data)
```

For Relation Manager, we only require tableName to conduct tuple-based operation. So we use two map data structures, mapping tableName to corresponding table ID and descriptor.

Another thing unexpected before implementation is that, instead of naively always return success, we do have to do sanity check for each operation and only return success when it take place and succeed. For example, you could not delete a non-exist record. RBFM change for this correspondingly.

3. Tests & Debugging

The implementation pass all 16 tests after some debugging.

During that process, I find gdb is quite useful. Instead of using handwriting print method to binary search root of bug, we could run it by gdb and use backtrace command to see the calling context when it crash.

And each time to rerun the tests, it’s important to remove catalog and attributes two meta file.