Catalog Manager 设计报告

计算机科学与技术学院数媒 1602 班 彭昊 3160104383

一、 模块概述

MiniSQL整体设计要求需要一个管理器记录储存所有表和索引的模式信息，并且通过该管理器能够访问这些模式信息并进行相应操作。这个管理器与数据库的各种增删改查的操作都息息相关，它为数据库提供正确的模式信息以及操作接口，并拥有储存记录这些信息的能力。

Catalog Manager则是负责管理数据库的所有模式信息的管理器，并提供访问以及操作相应信息的函数接口。

管理的模式信息有：

1.数据库中所有表的定义信息，包括表的名称、表中字段（列）数、主键、定义在该表上的索引。

2.表中每个字段的定义信息，包括字段类型、是否唯一等。

3.数据库中所有索引的定义，包括所属表、索引建立在那个字段上等。

二、 主要功能

本数据库Catalog Manager实现的接口功能有：

1.判断字段类型是否唯一、是否是主键。

2.添加或删除表的信息。

3.通过表名获取表的信息。

4.添加或者删除索引。

5.通过索引名或者通过表名以及字段名获取索引信息。

6.通过表名获取所有表上的索引名。

7.通过表名获取表上字段数。

另外，本数据库中Catalog Manager在析构时将所有模式信息通过buffer区写入磁盘文件，构造时将所有模式信息通过buffer区从磁盘文件读出，从而实现信息文件的磁盘存取与读写。

二、 对外提供的接口

1.判断字段是否唯一

```
public bool isUnique(const string attrName, const string tableName)
```

2.判断字段是否为主键

```
public bool isPK(const Attribute attrName)
```

3.添加新表

public bool addtable(const string tableName, const Attribute* attrName ,const int attrCount)

4. 删除已有表

public bool deletetable(const string tableName)

5.获取表信息

public Table* gettable(const string tableName)

6.添加新索引

public bool addIndex(const string indexName, const string tableName, const string attrName)

7.删除已有索引

public bool deleteIndex(const string indexName)

8.获取索引信息（通过索引名）

public Index* getIndex(const string indexName)

9.获取表上字段数

public int getattr_count(const string tableName)

10.获取表上所有索引名

public const IndexName* GetIndexName(const string tableName)

11.获取索引信息（通过表名和字段名）

public Index* getIndexByTableCol(const string tableName, const string attrName)

四、 设计思路

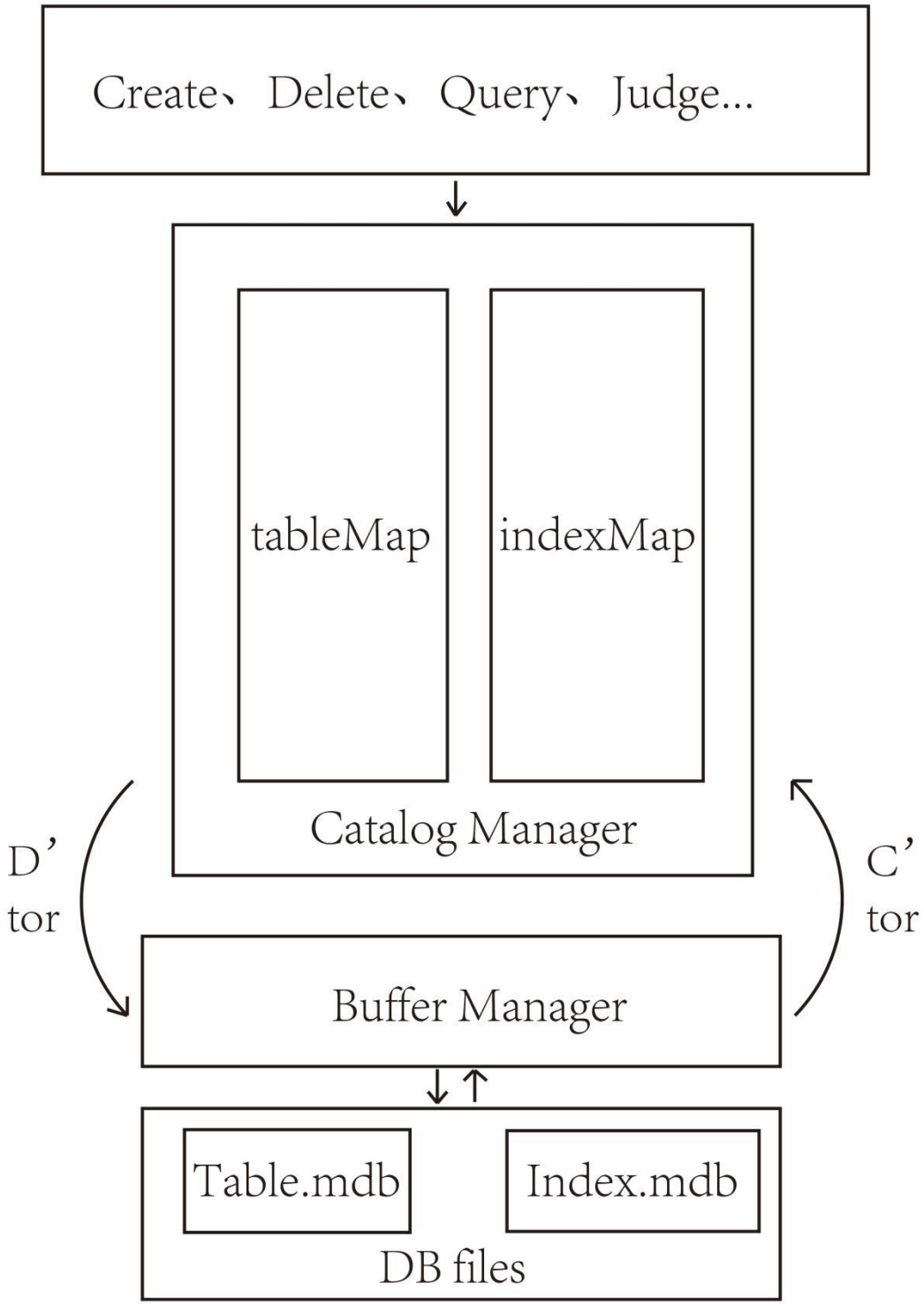
管理和记录所有的表和索引的模式信息可以通过建立两张unordered_map<string, T*>（T为Table或者Index）实现。通过表名或者索引名进入Map中可直接搜寻到对应的表或者索引的结构指针，进而可以访问表或者索引结构里的模式信息。

对于任何如增添、删除、查询、判断等上层的操作命令都可以在catalogManager类中反映到对于tableMap以及indexMap的操作。再将这些操作封装成接口函数提供给上层程序。

五、 整体架构

设计类 catalogManager，类中有两个unordered_map分别在内存中记录表以及索引的模式信息，并且对外提供访问操作这两个map的接口函数。如增添、删除、查询、判断等操作的接口都在catalogManager类中提供。

Catalog Manager析构时将内存中的两张Map存入buffer区，由BufferManager将信息存入磁盘上的Table.mdb以及Index.mdb两个文件中。而Catalog Manager构造时通过BufferManager将磁盘文件中储存的信息重新读入内存的两张Map中。即可实现所有模式信息的磁盘存取与读写。


Create、Delete、Query、Judge...
tableMap
indexMap
Catalog Manager
D'
tor
C'
tor
Buffer Manager
Table.mdb
Index.mdb
DB files

六、 关键函数和代码

CatalogManager 的构造函数：

```
CatalogManager::CatalogManager()
{

  string Tablefilename = "Table";
  string Indexfilename = "Index";
  //load table
  BufferManager* buffermanager = MiniSQL::get_BM();
  Block* block = buffermanager->get_block(("catalog/"+Tablefilename).c_str(), 0);
  int recordCount = *(reinterpret_cast<int*>(block->data));

  memset(block->data, 0, MAX_BLOCK_SIZE);
  block->dirty = false;
  for(int i = 1; i <= recordCount;i++){
     block = buffermanager->get_block(("catalog/"+Tablefilename).c_str(), i);
     char* tableData = block->data;
     Table* table = new Table;
     table->table_name = tableData;
     table->pk = tableData + 31;
     table->attr_count = *(reinterpret_cast<int*>(tableData + 62));
     for(int j=0 ; j < table->attr_count ; j++){
        table->attrs[j].attr_name= tableData +66+j*31+4*j*4;
        table->attrs[j].attr_type=*(reinterpret_cast<int*>(tableData +66+(j+1)*31+4*j*4));
        table->attrs[j].attr_key_type=*(reinterpret_cast<int*>(tableData
+66+(j+1)*31+(4*j+1)*4));
        table->attrs[j].attr_len=*(reinterpret_cast<int*>(tableData +66+(j+1)*31+(4*j+2)*4));
        table->attrs[j].attr_id=*(reinterpret_cast<int*>(tableData +66+(j+1)*31+(4*j+3)*4));
     }
     memset(block->data, 0, MAX_BLOCK_SIZE);
     block->dirty = false;
     tableMap[table->table_name] = table;
  }

  //load index
  block = buffermanager->get_block(("catalog/"+Indexfilename).c_str(), 0);
  recordCount = *(reinterpret_cast<int*>(block->data));

  memset(block->data, 0, MAX_BLOCK_SIZE);
  block->dirty = false;
 for(int i = 1; i <= recordCount;i++){
     block = buffermanager->get_block(("catalog/"+Indexfilename).c_str(), i);
     char* indexData = block->data;

     Index* index = new Index;
     index->index_name=indexData;
     index->table_name=indexData + 31;
     index->attr_name=indexData + 62;

     memset(block->data, 0, MAX_BLOCK_SIZE);
     block->dirty = false;
     indexMap[index->index_name] = index;
  }
}
```

构造函数中，分别对Table.mdb以及Index.mdb进行了读取，将其中的数据通过调用BufferManager放入block中，再将block中的信息按顺序存入内存中的tableMap以及indexMap中，以供后续接口函数使用。

CatalogManager 的析构函数：

```
CatalogManager::~CatalogManager()
{
   string Tablefilename = "Table";
   string Indexfilename = "Index";
   int i=1;
   BufferManager* buffermanager = MiniSQL::get_BM();
   Block* block;
   for (auto table : tableMap){
      if (table.second->table_name.empty()||table.second->attr_count == 0||table.second-
>pk.empty()) continue;
      block = buffermanager->get_block(("catalog/"+Tablefilename).c_str(), i);
      char* data = block->data;
      memcpy(data, table.second->table_name.c_str(), 31);
      memcpy(data+31, table.second->pk.c_str(), 31);
      memcpy(data+62, &table.second->attr_count, 4);
      for(int j =0 ;j < table.second->attr_count;j++){
         memcpy(data+66+j*31+4*j*4, table.second->attrs[j].attr_name.c_str(), 31);
         memcpy(data+66+(j+1)*31+4*j*4, &table.second->attrs[j].attr_type, 4);
         memcpy(data+66+(j+1)*31+(4*j+1)*4, &table.second->attrs[j].attr_key_type, 4);
         memcpy(data+66+(j+1)*31+(4*j+2)*4, &table.second->attrs[j].attr_len, 4);
         memcpy(data+66+(j+1)*31+(4*j+3)*4, &table.second->attrs[j].attr_id, 4);
       }
      i++;
      block->dirty = true;
      delete table.second;
   }
   i=i-1;
   block = buffermanager->get_block(("catalog/"+Tablefilename).c_str(), 0);
   char* data = block -> data;
   memcpy(data , &i, 4);
   block->dirty = true;

   i=1;
   for (auto index : indexMap){
      if(index.second->table_name.empty() || index.second->index_name.empty() ||
index.second->attr_name.empty()) continue;
      block = buffermanager->get_block(("catalog/"+Indexfilename).c_str(), i);
      char* data = block -> data;
      memcpy(data, index.second->index_name.c_str(), 31);
      memcpy(data+31, index.second->table_name.c_str(), 31);
      memcpy(data+62, index.second->attr_name.c_str(), 31);
         i++;
      block->dirty = true;
      delete index.second;
   }
   i=i-1;
   block = buffermanager->get_block(("catalog/"+Indexfilename).c_str(), 0);
   data = block -> data;
   block->dirty = true;
   memcpy(data, &i, 4);
}
```

析构函数中，分别对tableMap和indexMap中的信息进行遍历，将这些信息通过Buffer Manager存入对应文件的block中，当Buffer Manager析构的时候则会将这些block的信息存入对应磁盘文件，实现模式信息的磁盘存储。

添加新表：

```
bool CatalogManager::addtable(  const string tableName, const Attribute* attrName ,const int attrCount)
{
    if(tableExists(tableName)){
        cerr << "ERROR: [CatalogManager::addtable] Table `" << tableName << "` already exists!" << endl;
        return false;
    }
    if(tableName.empty()){
        cerr << "ERROR: [CatalogManager::addtable] Tablename cannot be empty!" << endl;
        return false;
    }
    int i=0;
    int p=0;

    for (i=0;i<attrCount;i++){
        if(isPK(attrName[i])) break;
    }
    if(i>=attrCount){
        cerr << "ERROR: [CatalogManager::addtable] Cannot find primary key !" << endl;
        return false;
    }
    else{
        p=i;
    }

    for(int i=0;i<attrCount-1;i++)
        for(int j=i+1;j<attrCount;j++){
            if(attrName[i].attr_name==attrName[j].attr_name){
                cerr << "ERROR: [CatalogManager::addtable] Duplicate column name `" << attrName[i].attr_name << "`!" << endl;
                return false;
            }
        }

    Table* table = new Table;
    table->table_name = tableName;
    table->pk = attrName[p].attr_name;
    table->attr_count = attrCount;
    for(i=0;i<attrCount;i++)
        table->attrs[i] = attrName[i];
    tableMap[tableName] = table;

    return true;
}
```

先检查新表名是否已经在tableMap中存在、新表名是否为空、传入字段是否有主键、传入字段是否重名等错误，相应地返回错误信息。若无错误，则新建一个table结构将信息一一储存，再将table放入tableMap，返回true。

删除已有表：

```
bool CatalogManager::deletetable(const string tableName)
{
  if(!tableExists(tableName))
  {
    cerr << "ERROR: [CatalogManager::deletetable] Table `" << tableName << "` does not exist!" <<
endl;
    return false;
  }
  else{
    delete tableMap[tableName];
    tableMap.erase(tableName);

    return true;
  }
```

先检查应删除表是否存在于tableMap，若不存在则返回错误信息。若无错误，则将相应table从tableMap中删去，返回true。

获取表信息：

```
Table* CatalogManager::gettable(const string tableName) //const
{
  if(!tableExists(tableName))
  {
    cerr << "ERROR: [CatalogManager::gettable] Table `" << tableName << "` does not exist!" <<
endl;
    return NULL;
  }
  else return tableMap.at(tableName);
}
```

先检查该查找表是否存在于tableMap，若不存在则返回错误信息。若无错误，则将相应table从tableMap中找出，返回该表结构。

添加新索引：

```
bool CatalogManager::addIndex(  const string indexName, const string tableName, const
string attrName)
{
   if(!tableExists(tableName))
   {
       cerr << "ERROR: [CatalogManager::addIndex] Table `" << tableName << "` does not
exist!" << endl;
      return false;
   }

   if(tableName.empty()){
      cerr << "ERROR: [CatalogManager::addIndex] Tablename cannot be empty!" << endl;
      return false;
   }
   if(indexName.empty()){
      cerr << "ERROR: [CatalogManager::addIndex] Indexname cannot be empty!" << endl;
      return false;
   }
   if(indexExists(indexName)){
         cerr << "ERROR: [CatalogManager::addIndex] Index `" << indexName << "` already
exists!" << endl;
         return false;
   }
   if(!isUnique(attrName,tableName)){
       cerr << "ERROR: [CatalogManager::addIndex] Attribute `" << attrName << "` is not
unique!" << endl;
       return false;
   }

   Index* exist = NULL;
   for (auto item : indexMap)
   {
      Index* index0 = item.second;
      if (index0->table_name==tableName && index0->attr_name==attrName)
         exist=index0;
   }
   if(exist!=NULL)
   {
      cerr << "ERROR: [CatalogManager::addIndex] Index with table name `" << tableName <<
"` and attribute name `" << attrName << "` already exists(Index name `" << indexName << "`)!"
<< endl;
      return false;
   }


   Index* index = new Index;
   index->index_name=indexName;
   index->table_name=tableName;
   index->attr_name=attrName;
   indexMap[indexName] = index;

   return true;
}
```

先检查对应表名是否不存在于tableMap、对应表名是否为空、新索引名是否为空、新索引名是否已经在indexMap中存在、对应字段是否属性唯一、对应表上的对应字段是否已经存在索引等错误，相应地返回错误信息。若无错误，则新建一个index结构将信息一一存入，再将index放入indexMap，返回true。

删除已有索引：

```
bool CatalogManager::deleteIndex(const string indexName)
{
   if(!indexExists(indexName))
   {
      cerr << "ERROR: [CatalogManager::deleteIndex] Index `" << indexName << "` does not
exist!" << endl;
      return false;
   }
   else{
      delete indexMap[indexName];
      indexMap.erase(indexName);
      return true;
   }
}
```

先检查应删除索引是否存在于indexMap，若不存在则返回错误信息。若无错误，则将相应index从indexMap中删去，返回true。

获取索引信息（通过索引名）：

```
Index* CatalogManager::getIndex(const string indexName) //const
{
   if(!indexExists(indexName))
   {
      cerr << "ERROR: [CatalogManager::deleteIndex] Index `" << indexName << "` does not exist!"
<< endl;
      return NULL;
   }
   else return indexMap.at(indexName);
}
```

先检查该查找索引是否存在于indexMap，若不存在则返回错误信息。若无错误，则将相应index从indexMap中找出，返回该索引结构。

获取索引信息（通过表名和字段名）：

```
Index* CatalogManager::getIndexByTableCol(const string tableName, const string attrName)
{
   if(!tableExists(tableName))
   {
      cerr << "ERROR: [CatalogManager::getIndexByTableCol] Table `" << tableName << "` does
not exist!" << endl;
      return NULL;
   }
   Index* exist;
   for (auto item : indexMap)
   {
      Index* index0 = item.second;
      if (index0->table_name==tableName && index0->attr_name==attrName)
         exist=index0;
      return exist;
   }
   if(exist == NULL)
   {
      cerr << "ERROR: [CatalogManager::getIndexByTableCol] Index with table name `" <<
tableName << "` and attribute name `" << attrName << "` does not exist!" << endl;
      return NULL;
   }
}
```

先检查该查找表是否存在于tableMap，若不存在则返回错误信息。再遍历表上字段，寻找索引，若能找到则返回该索引，若无法找到，则返回错误信息。