Record Manager 设计报告

计算机科学与技术学院数媒 1602 班 陈逸雪 3160102516

一、 模块概述

Record Manager 是与 Catalog Manager 和 Index Manager 并列的管理数据文件的模块。它为 API 提供与数据记录操作相关的函数接口，并调用 Buffer Manager 的函数实现对内存和磁盘文件的读写。Record Manager 负责管理记录表中数据的数据文件。

根据实验要求，数据文件暂时只支持定长记录的存储。Record Manager 可支持的数据类型有 int，float 和 char。数据文件由一个或多个数据块组成，块大小与缓冲区块大小相同。

二、 主要功能

1. 数据文件的创建（由表的定义与删除引起）

如果语句执行成功，则返回 true；否则，返回 false。

2. 数据文件的删除

如果语句执行成功，则返回 true；否则，返回 false。

3. 记录的插入

能够根据所给的记录去插入到相应的表中，并返回插入记录的位置。

4. 记录的删除

该操作能够支持不带条件的删除和带一个条件的删除（包括等值查找、不等值查找和区间查找）。

5. 记录的查询

该操作能够支持不带条件的查找和带一个条件的查找（包括等值查找、不等值查找和区间查找）。

6. 提供相应接口

提供相应的接口供 API 和其它函数调用。

三、 对外提供的接口

1.数据文件的创建

`+ createTable(Table* table)：bool` 按照传入的 table 指针建表，创建 mdb 文件。若表已存在或者创建失败，返回 false 并输出错误信息。

2.数据文件的删除

`+ dropTable(Table* table)：bool` 删除对应的 mdb 文件，若表不存在或者删除失败，返回 false 并输出错误信息。

3.记录的插入

`+ insert(Record& record)：int` 首先判断表是否存在，然后对记录进行检验，是否存在主键相同的问题，然后插入记录并返回记录所存储位置的 id。

4.记录的删除

`+ deleteValue(Table* table, Condition_list clist)：bool` 按照条件删除表中的对应记录，成功返回 true，失败返回 false 并输出错误信息。

`+ deleteValue(Table* table)：bool` 删除整张表中的记录。

5.记录的查询

`+ select(Table* table, Condition_list clist)：Data*` 按照条件查找对应记录，返回存有记录的 data 数据结构。

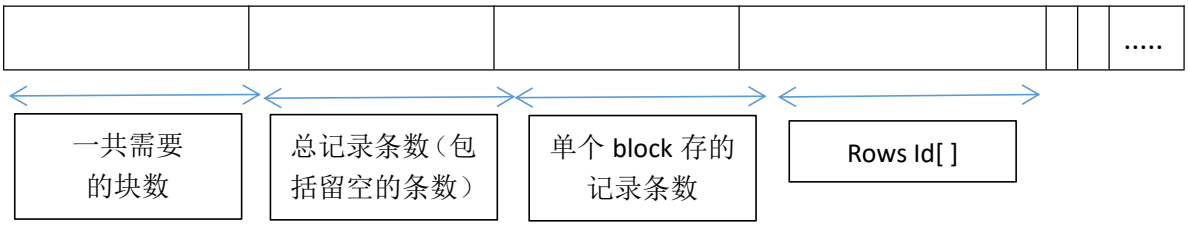
`+ select(Table* table)：Data*` 首先判断表是否存在，存在的话返回整个 data 数据结构。

四、设计思路

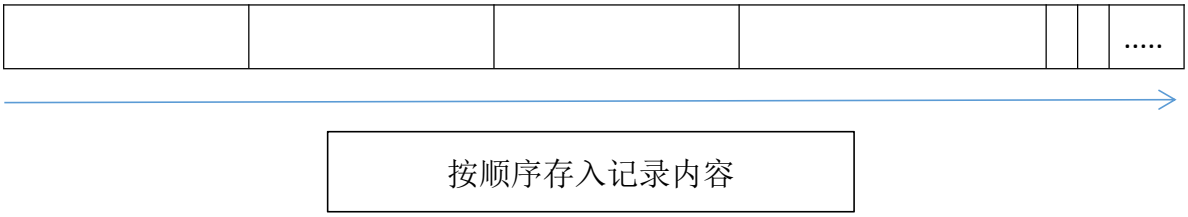
Record Manager 模块包含了两个 cpp 文件，recordManager.cpp 主要是对操作行为的管理，而 data.cpp 是对对象进行的实际操作。

首先是对 data 数据结构的设计，它包含了表名（tableName），以记录元组（tuples）为元素的容器，还有记录了每条记录位置的数组 rowsId（rowId [0]表示所储存的总记录条数，rowId [i]表示第 i 个位置所储存的记录序号，若该位置要留空，则值为 -1）。

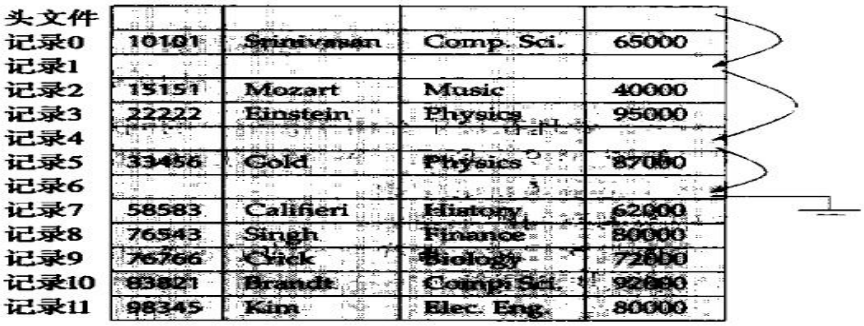
块头信息 Block[0]：



子块：

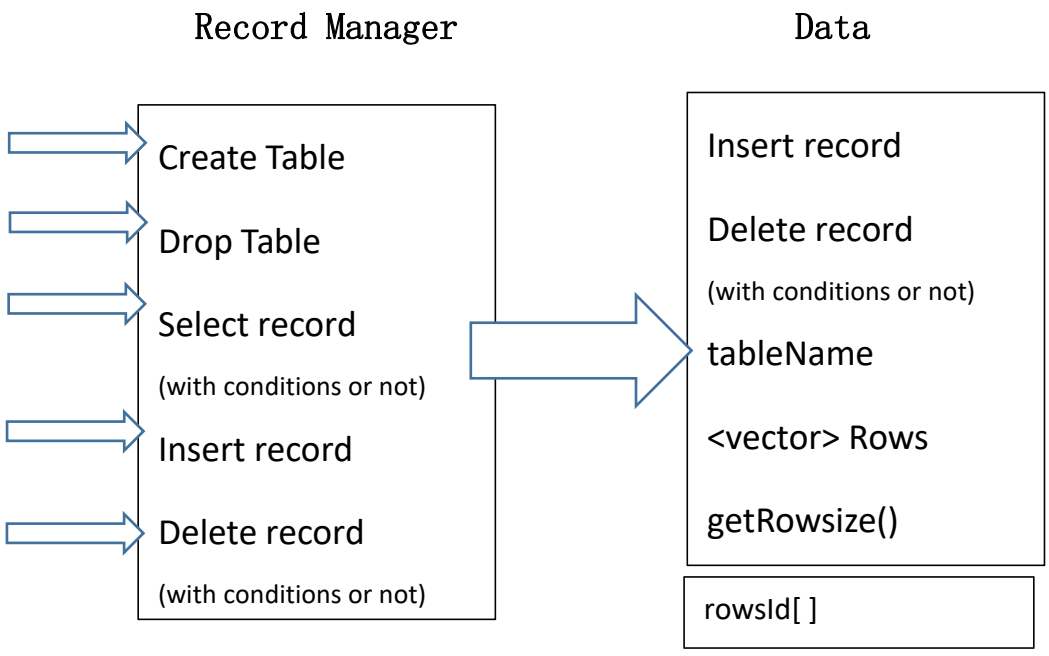


然后对于 record Manager 来说，只是对 data 数据结构进行相应函数的调用。每次调用完对 data 进行析构，



其中为了节省空间和方便记录 index，记录的插入删除操作都符合空闲列表的定义，只不过在代码中将链表简化为数组来操作。

五、整体架构



设计类 record Manager，由其提供对外接口。在每一个函数的实现过程中，临时构建数据结构 data，在函数实现过程中该任务由类 Data 承担。更新成功后，将这个数据结构的信息存回内存中进行管理。在 Data 内，承担插入、删除的具体操作。而查找主要在 record Manager 内临时构建一个 data 将符合条件的记录传出来。

六、 关键函数和代码（伪代码）

1.Record Manager 查找函数

返回不带条件的查找则直接返回 data 的数据结构，若是带条件的则构建一个新的 data，将符合条件的记录数存进去返回。

带条件：

```
Data* RecordManager::select(Table* table, Condition_list clist) {
    string filename = table->table_name;
    Data* data0 = new Data(filename.c_str());
    Data* data1 = new Data;
    data1->tableName = filename;
    int count = clist.size();
    int j;
    for (int i = 0; i < data0->rows.size(); i++) {
        list<Condition>::iterator v = clist.begin();
        for (j = 0; j < count; j++ ) {
            int id = table->searchAttrId((*v).attr_name);
            string b = data0->rows[i].attr_values[id];
            if ((*v).attr_type == STRING) { //char
                if (data0->charCmp((*v).cmp_value, b, (*v).op_type) == false)
                    break;
            }
            if ((*v).attr_type == INTNUM) {
                if (data0->intCmp((*v).cmp_value, b, (*v).op_type) == false)
                    break;
            }
            if ((*v).attr_type == FLOATNUM) {
                if (data0->floatCmp((*v).cmp_value, b, (*v).op_type) == false)
                    break;
            }
```

```
                *v++;
        }
        if (j == count) {
            Tuple tuple;
            tuple = data0->rows[i];
           data1->rows.push_back(tuple);
        }

    }


    return data1;
}
```

不带条件：

```
Data* RecordManager::select(Table* table) {

    string filename = table->table_name;

    Data* datas = new Data(filename.c_str());

    return datas;
```

2.Data 构造函数

```
Data::Data(const char* filename) :_filename(filename) {
    BufferManager* buffermanager = MiniSQL::get_BM();
    Block* block = buffermanager->get_block(filename, 0);
    int blockCount = *(reinterpret_cast<int*>(block->data));
    int count = *(reinterpret_cast<int*>(block->data + 4));
    int recordcount = *(reinterpret_cast<int*>(block->data + 8));
    for(int m=0;m<=recordcount;m++)
        rowsId[m]= *(reinterpret_cast<int*>(block->data + 8+ m*4));
    block->dirty = false;
    tableName = _filename;
    CatalogManager* manager = MiniSQL::get_CM();
    int attrcount = manager->getattr_count(tableName);
    Table* table = manager->gettable(tableName);
    int i;
    for (i = 1; i <= blockCount; i++) {
        Block* block = buffermanager->get_block(filename, i);
        char* recordData = block->data;
        int k;
       int bias = 0;
```

```
if (i != blockCount) {
            for (int j = (i - 1)*count + 1; j <= i*count; j++) {
                if (strcmp(recordData + bias, "EMPTY") != 0) {
                  Tuple tuple;
                    for (k = 0; k < attrcount; k++) {
                        if (table->attrs[k].attr_type == INT) {
                            int a= *(reinterpret_cast<int*>(recordData + bias));
                            tuple.attr_values[k] = to_string(a);
                        }
                        else if (table->attrs[k].attr_type == FLOAT) {
                            float a = *(reinterpret_cast<float*>(recordData + bias));
                            tuple.attr_values[k] = to_string(a);
                        }
                        else {
                            char ch[table->attrs[k].attr_len+1];
                            strncpy(ch , recordData + bias, table->attrs[k].attr_len);
                            ch[table->attrs[k].attr_len] = '\0';
                            tuple.attr_values[k] = ch;
                        }
                        bias += table->attrs[k].attr_len;
                    }
                  rows.push_back(tuple);
                }

            }
        }
        if (i == blockCount) {
            cout << "LastBlockCount:" << i << endl;
            for (int j = (blockCount - 1) * count+1; j <= recordcount; j++) {
                if (strcmp(recordData + bias, "EMPTY") != 0){
                  Tuple tuple;
                    for (k = 0; k < attrcount; k++) {
                        if (table->attrs[k].attr_type == INT) {
                            int a = *(reinterpret_cast<int*>(recordData + bias));
                            tuple.attr_values[k] = to_string(a);
                        }
                        else if (table->attrs[k].attr_type == FLOAT) {
                            float a = *(reinterpret_cast<float*>(recordData + bias));
                            tuple.attr_values[k] = to_string(a);
                        }
                        else {
                            char ch[table->attrs[k].attr_len+1];
                            strncpy(ch , recordData + bias, table->attrs[k].attr_len);
                            ch[table->attrs[k].attr_len] = '\0';
```

```
                                tuple.attr_values[k] = ch;
                            }
                            bias += table->attrs[k].attr_len;
                        }
                    rows.push_back(tuple);
                }
                else{
                    bias += table->length();
                }
            }
        }
        block->dirty = false;
    }
}
```

3. Data 析构函数

```
Data::~Data() {
    printf("~data\n");
    int k,i;
    int j = 1;
    int blockcount;
    CatalogManager* manager = MiniSQL::get_CM();
    BufferManager* buffermanager = MiniSQL::get_BM();
    Table* table = manager->gettable(tableName);
    int attrcount = manager->getattr_count(tableName);
    int count = MAX_BLOCK_SIZE /table->length();
    if (rowsId[0] <= count)
        blockcount = 1;
    else if (rowsId[0] % count == 0)
        blockcount = rowsId[0] / count;
    else
        blockcount = rowsId[0] / count + 1;
    for (i = 1; i <= blockcount; i++) {
       Block* block = buffermanager->get_block(_filename.c_str(), i);
       int counts = 0;//记录一个 block 存的记录数
       int bias = 0;
       for (j; j <= rowsId[0]; j++) {
           if (rowsId[j] != -1) {
               for (k = 0; k < attrcount; k++) {
                   if (table->attrs[k].attr_type == INT) {
                       int a=stringToNum<int>(rows[rowsId[j] - 1].attr_values[k]);
```

```
                            memcpy(block->data + bias, &a, table->attrs[k].attr_len);

                        }
                    else if (table->attrs[k].attr_type == FLOAT) {
                        float a = stringToNum<float>(rows[rowsId[j] -
1].attr_values[k]);
                        memcpy(block->data + bias, &a, table->attrs[k].attr_len);
                    }
                    else {
                        char s[table->attrs[k].attr_len+1];
                        strncpy(s, (rows[rowsId[j] - 1].attr_values[k]).c_str(),
table->attrs[k].attr_len);
                        s[table->attrs[k].attr_len] = '\0';
                        memcpy(block->data + bias, s, table->attrs[k].attr_len);
                    }
                    bias += table->attrs[k].attr_len;
                    //leave null space
                }
                counts++;
            }
            else {
                string str = "EMPTY";
                memcpy(block->data + bias, str.c_str(), table->length());

                bias += table->length();
                counts++;
            }
            if (counts == count) {
                break;
            } //a block is full, come to next block
        }
        block->dirty = true;
    }
    i--;

    Block* block = buffermanager->get_block(_filename.c_str(), 0);
    memcpy(block->data, &i, 4); //i block
    memcpy(block->data + 4, &count, 4);//one block stores count records
    for(int m=0;m<=rowsId[0];m++)
        memcpy(block->data + 8+ 4*m, &rowsId[m], 4);
    block->dirty = true;
}
```

4. Data 插入函数

```
int Data::insert(Record& record) {
    CatalogManager* manager = MiniSQL::get_CM();
    Table* table = manager->gettable(record.table_name);
    int prid = table->getPrimaryKeyId();
    for (int i = 0; i < rows.size(); i++) {
        if (rows[i].attr_values[prid] == record.attr_values[prid]) {
            cerr << "ERROR: [recordManager::insert] Primarykey `" << table->pk << "` is
not unique!" << endl;
            return -1;
        }
    }
    Tuple tuples;
    for (int i = 0; i < record.num_values; i++) {
       tuples.attr_values[i] = record.attr_values[i];
    }
    rows.push_back(tuples);
    int m;
    for (m = 1; m <= rowsId[0]; m++) {
        if (rowsId[m] == -1) {
            rowsId[m] = rows.size();
            return m;
        }
    }
    if (m == rowsId[0] + 1) {
        rowsId[m] = rows.size();
        rowsId[0]++;
        int left = stringToNum<int>(rows[0].attr_values[2]);
        return m;
    }
}
```

5. Data 删除函数

带条件：

```
bool Data::deleteValue(Table* table,Condition_list clist) {
    int count = clist.size();
    int i=0;
    vector<Tuple>::iterator m;
    for (m = rows.begin(); m!=rows.end(); *m++) {
        list<Condition>::iterator v;
       int j=0;
        for (v = clist.begin(); v != clist.end(); *v++) {
            int id = table->searchAttrId((*v).attr_name);
            string b = (*m).attr_values[id];
            if ((*v).attr_type == STRING) {
                if (charCmp((*v).cmp_value, b, (*v).op_type) == false)
                    break;
            }
            if ((*v).attr_type == INTNUM) {
                if (intCmp((*v).cmp_value, b, (*v).op_type) == false)
                break;
            }
            if ((*v).attr_type == FLOATNUM) {
                if (floatCmp((*v).cmp_value, b, (*v).op_type) == false)
                    break;
            }
            j++;
        }
        i++;
        if (j == count) {
           for (int k = 1; k <= rowsId[0]; k++) {
                if (rowsId[k] > rowsId[i])
                    rowsId[k]--;
            }
            rowsId[i] = -1;
           m = rows.erase(m);
            *m--;
        }
    }
    return true;
}
```