浙江大学

数据库系统实验报告

作业名称：	MiniSQL 设计
姓　　名：	朱理真
学　　号：	3190101094
电子邮箱：	3190101094@zju.edu.cn
联系电话：	19817862976
指导老师：	孙建伶

2021 年 7 月 4 日

MiniSQL 设计(索引部分)

一、 实验目的

设计并实现一个精简型单用户 SQL 引擎(DBMS)MiniSQL，允许用户通过字符界面输入 SQL 语句实现表的建立/删除；索引的建立/删除以及表记录的插入/删除/查找。

通过对 MiniSQL 的设计与实现，提高学生的系统编程能力，加深对数据库系统原理的理解

就该报告而言，主要处理索引的建立/删除和搜索问题。

二、 系统需求

数据类型

只要求支持三种基本数据类型：int，char(n)，float，其中 char(n)满足 1 <= n <= 255 。

表定义

一个表最多可以定义 32 个属性，各属性可以指定是否为 unique；支持 unique 属性的主键定义。

索引的建立和删除

对于表的主键自动建立 B+树索引，对于声明为 unique 的属性可以通过 SQL 语句由用户指定建立/删除 B+树索引（因此，所有的 B+树索引都是单属性单值的）。

查找记录

可以通过指定用 and 连接的多个条件进行查询，支持等值查询和区间查询。

插入和删除记录

支持每次一条记录的插入操作；

支持每次一条或多条记录的删除操作。(where 条件是范围时删除多条)

就该报告而言，主要处理索引的需求。具体如下：

1.索引具备对 int，char(n),float 的建立，删除，查找（单值和范围查找）功能，其中字符个数限制在 255 以内。

2.每一个索引值都是单属性，且唯一的。

3.实现结果为 B+树

4.考虑到记录结构未实现按主索引键顺序存储，故所有索引均需采用辅助索引

三、 实验环境

操作系统：Windows10

语言：C++ （C++11 以上标准）

编译工具：Visual Studio 2019

四、 模块设计

1. Index Manager

i. 功能描述

实现并提供以下功能的接口：

1.给定索引键类型，建立索引
2.删除整个索引
3.给定键值和索引键所在记录在硬盘中存储的位置，插入一个索引键
4.给定键值，删除一个索引键
5.给定键值，查找一个索引键，返回其所在的记录
6.给定键值，查找范围内的索引键，返回它们所在的记录

ii. 主要数据结构

算法数据结构

B+树：
1.每个节点占用一个块(4KiB)大小
2.根据键值的大小自动计算 B+树的叉数 n
3.除根节点，非叶子节点最大孩子个数为 n,最少孩子个数为 cell(n/2)
4.叶子节点最大孩子个数为 n-1,最少孩子个数为 cell((n-1)/2)

实现数据结构

BptNode:
代表 B+树的一个节点，主要提供以下功能：
1.节点数据在磁盘中读取或写入
2.块内键值二分搜索
3.块内插入
4.块内删除
5.与其他块合并
6.与其他块再分配
7.找到子节点
8.作为叶子节点，找到邻接的下一个叶子节点
9.提供块内数据读取和修改的接口

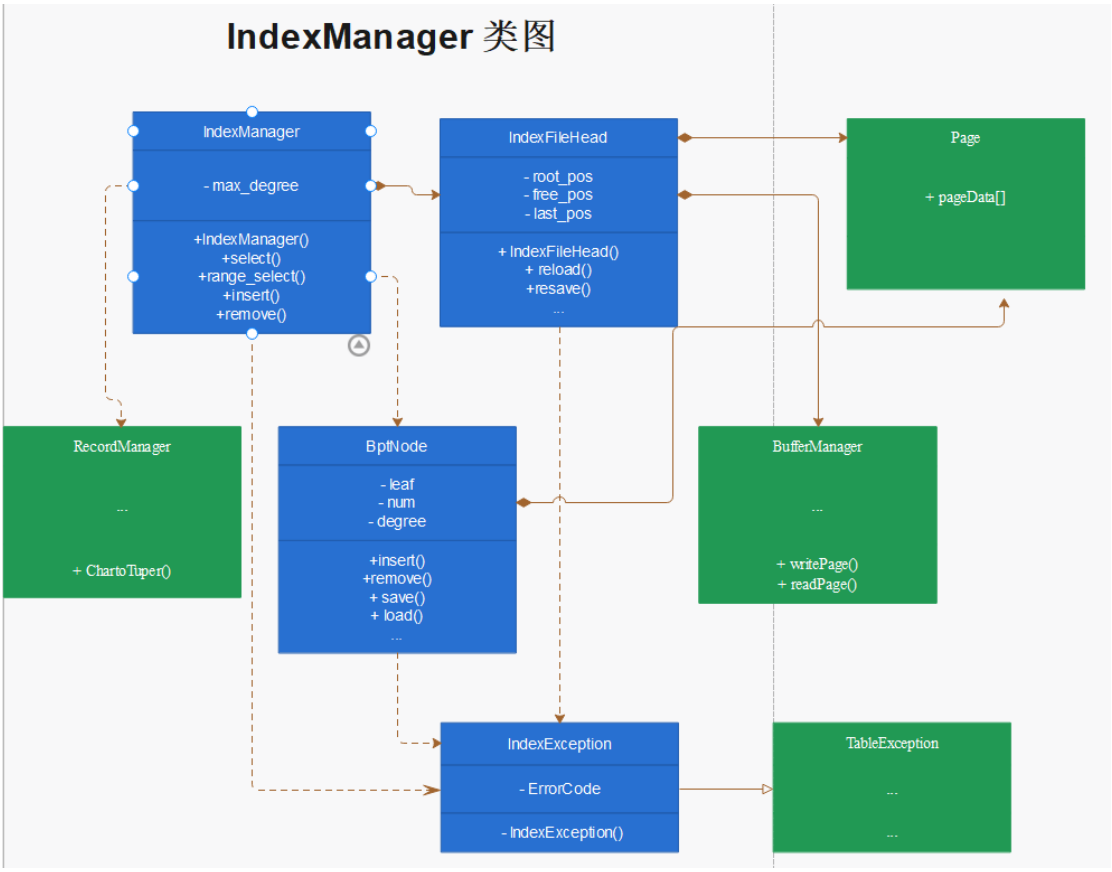
IndexFileHead:
索引文件头，实现索引文件空间的管理:
1.打开索引文件，获取索引文件信息
2.读取，修改和保存索引文件信息
2.格式化索引文件
3.找到索引根节点
4.释放索引节点空间
5.分配索引节点空间
索引节点的分配用到了空闲链表结构

IndexException:
继承 TableException，
提供异常信息

IndexManager:

提供外部接口，同时调用其他数据结构实现索引的功能

iii. 给出类图(如果有)，以及类间关系



类间关系如上图，其中蓝色部分是报告包括的，绿色部分则是他人的成果（具体细节不表，仅显示需要的部分）。（绿色的类互相之间可能也有连接，但是这不是 Index 所要关系的。）

IndexManger 包含类 IndexFileHead，同时在过程中主要使用类 BptNode，实现搜索功能时，使用类 RecordManager 的一个静态（图中未表现出来）函数

IndexFileHead 与物理存储交互，因此包含 Page 类和 BufferManager 类, BptNode 也是同理。

同时，所有 Index 部分的类都会使用类 IndexException,以在异常时抛出

五、 模块实现

1. 阐述该模块所使用的数据结构，说明实现各个功能的核心代码(给出简化后的代码截图)，包括重要的函数、提供给 API 模块的接口(如果有)等。

IndexManager:

实现并提供以下功能的接口：

1.给定索引键类型，建立索引

2.删除整个索引

3.给定键值和索引键所在记录在硬盘中存储的位置，插入一个索引键

4.给定键值，删除一个索引键
5.给定键值，查找一个索引键，返回其所在的记录
6.给定键值，查找范围内的索引键，返回它们所在的记录

核心代码：
1.建立索引（API 接口）

```
IndexManager::IndexManager(
    const Table& t,
    int attr,
    int new_index
):index_head(t.index.IndexName[attr],new_index) {
    DataInt tmp(0);
    int key_size = tmp.getSize(t.attr.flag[attr]);
    max_degree =
        (PAGE_SIZE - 2 * sizeof(int)-sizeof(Pos))
        / (sizeof(Pos) + key_size)
        + 1;
}
```

建立索引需要设置参数 new_index = true

2.删除索引，同上，设置 new_index=true 即格式化索引文件
3.插入索引
API 接口：

```
void IndexManager::insert(
    const Data& key,
    int block_index,
    int offset
) {
    BptNode* proot,*p;
    if (index_head.getRootPos() == index_head.NULL_POS) {
        proot = new BptNode(index_head.getFilename(), index_head.getNewPage(), max_degree);
        proot->setState(true, 0);
        proot->save(index_head.bm);
        index_head.setRootPos(proot->getPos());
    }
    else proot = new BptNode(index_head.getFilename(), index_head.getRootPos(), max_degree);

    vector<BptNode*> ans;
    vector<int> his_pos;
    Pos pos = find(key, *proot, ans,his_pos,findUNIQUE);
    Pos insert_pos;
    insert_pos.pos[0] = block_index;
    insert_pos.pos[1] = offset;
    pure_insert(ans, his_pos, key, insert_pos, pos.pos[1]);
    while (!ans.empty()) {
        p = ans.back();
        ans.pop_back();
        delete p;
    }
}
```

其中比较重要的函数：

find：找到所在键值的叶子节点及在其中的位置

```
Pos IndexManager::find(// judge if key exists and be unique
    const Data& key,
    BptNode& root,
    vector<BptNode*>& ancestor,
    vector<int>& his_pos,
    int flag
) {
    Pos pos;
    BptNode* node = &root;
    BptNode* next;
    int i;
    while (1) {
        node->load(index_head.bm);
        ancestor.push_back(node);
        pos = node->binary_search(key);

        if (node->IsLeaf()) break;
        i = pos.pos[1];
        if (pos.pos[1] == pos.pos[0]) i++;

        his_pos.push_back(i);

        next = new BptNode(*node, i);
        node = next;
    }
    if (flag == findUNIQUE && pos.pos[1] == pos.pos[0]) {
        throw IndexException(KEY_NOT_UNIQUE, "key not unique");
    }
    else if (flag == findEXIST && pos.pos[1] != pos.pos[0]) {
        throw IndexException(KEY_NOT_EXIST, "key not exist");
    }
    return pos;
}
```

pure_insert：给定插入位置，进行键值插入

```cpp
void IndexManager::pure_insert(
    vector<BptNode*>&ancestor,
    vector<int>& his_pos,
    //BptNode& t,
    const Data& key,
    const Pos& pos,
    int index
) {
    BptNode& t = *(ancestor.back());
    ancestor.pop_back();
    if (t.getNum() < max_degree - 1) {
        t.leaf_insert(key, pos, index);
        t.save(index_head.bm);
        delete& t;
    }
    else {
        BptNode ano(index_head.getFilename(), index_head.getNewPage(), max_degree);
        Data* p_new_key = key.new_copy();
        t.leaf_split(ano, key, pos, index, *p_new_key);
        ano.save(index_head.bm);
        t.save(index_head.bm);

        //t.PrintNode(key);
        //ano.PrintNode(key);

        parent_insert(ancestor,his_pos, t, *p_new_key, ano);
        delete p_new_key;
        delete& t;
    }
}
```

上个函数的子函数 parent_insert：进行非叶子节点的插入

```cpp
void IndexManager::parent_insert(
    vector<BptNode*>& ancestor,
    vector<int>& his_pos,
    BptNode& child,
    const Data& key,
    BptNode& new_child) {
```

```
    if (ancestor.empty()) {
        BptNode root(index_head.getFilename(), index_head.getNewPage(), max_degree);
        root.setState(0,1, root.NO_CHANGE);
        root.setKey(key, 0);
        root.setPos(child, 0);
        root.setPos(new_child, 1);
        root.save(index_head.bm);
        index_head.setRootPos(root.getPos());
    }
    else {
        BptNode& P = *(ancestor.back());
        ancestor.pop_back();
        int tmp_pos = his_pos.back();
        his_pos.pop_back();
        if (P.getNum() < max_degree - 1) {
            P.nonleaf_insert(key, new_child, tmp_pos);
            P.save(index_head.bm);
            delete& P;
        }
        else {
            BptNode ano(index_head.getFilename(), index_head.getNewPage(), max_degree);
            Data* pnew_key = key.new_copy();
            P.parent_split(
                ano,
                key,
                tmp_pos,
                new_child,
                *pnew_key
            );
            ano.save(index_head.bm);
            P.save(index_head.bm);
            parent_insert(ancestor,his_pos, P, *pnew_key, ano);
            delete pnew_key;
            delete& P;
        }
    }
}
```

4.删除索引键

API 接口：

```
void IndexManager::remove(const Data& key) {
    BptNode* proot, * p;
    if (index_head.getRootPos() == index_head.NULL_POS) {
        throw IndexException(DEFAULT, "index empty");
    }
    else
        proot = new BptNode(
            index_head.getFilename(),
            index_head.getRootPos(),
            max_degree
        );

    vector<BptNode*> ans;
    vector<int> his_pos;
    Pos pos = find(key, *proot, ans, his_pos, findEXIST);
    his_pos.push_back(pos.pos[1]);
    //if (pos.pos[0] != pos.pos[1])
    //  throw IndexException(DELETE_FAILED, "key not found");
    remove_entry(ans, his_pos, key);
    while (!ans.empty()) {
        p = ans.back();
        ans.pop_back();
        delete p;
    }
}
```

重要函数 remove_entry：

```
void IndexManager::remove_entry(
    vector<BptNode*>& ancestor,
    vector<int>& his_pos,
    const Data& key
) {
    BptNode& t = *ancestor.back();
    int delete_pos = his_pos.back();
    t.remove_in_node(key,delete_pos);
    if (ancestor.empty() && t.getNum() == 0) {
        int lastroot = t.getPos();
        if (t.IsLeaf()) { ... }
        else { ... }
        index_head.freePage(lastroot);
        return;
    }
    if (!ancestor.empty() && t.tooless()) {
        int case1, case2;
        delete_pos = his_pos.back();
        Data&delete_key = *key.new_copy();
        Data&new_delete_key = *key.new_copy();
        BptNode& P = *ancestor.back();
        BptNode* left=nullptr;
        BptNode* right=nullptr;

        if (delete_pos == 0) case1 = 0;
        else { ... }

        if (delete_pos == P.getNum()) case2 = 0;
        else { ... }
        int flag = case2 + (case1 << 4);
        switch (flag) {
        case 0x01:
            if (right) {
                t.coalesce(delete_key, *right);
                remove_entry(ancestor, his_pos, key);
                break;
```

```
            break;
        }
        else throw IndexException(DEFAULT, "pointer null");
    case 0x10:
    case 0x11:
        left->coalesce(delete_key, t);
        remove_entry(ancestor, his_pos, key);
        break;
    case 0x02:
    case 0x12:
        if (right) {
            t.distribute_from(*right, delete_key, new_delete_key);
            P.setKey(new_delete_key, delete_pos);
            break;
        }
        else throw IndexException(DEFAULT, "pointer null");
    case 0x20:
    case 0x21:
    case 0x22:
        P.getKey(delete_key, delete_pos-1);
        left->distribute_to(t, delete_key, new_delete_key);
        P.setKey(new_delete_key, delete_pos-1);
        break;
    default:
        throw IndexException(DEFAULT, "delete entry error");
    }
  }
}
```

5.查找索引键：（API）

```
void IndexManager::select(const Data& key, Table& t) {
    Pos pos, rec_pos;
    Tuper* tp;
    RecordManager rm(NULL);
    int root_pos = index_head.getRootPos();
    if (root_pos == index_head.NULL_POS) return;
    BptNode& root = *new BptNode(index_head.getFilename(), root_pos, max_degree);
    pos = find(key, root);
    if (pos.pos[1] != pos.pos[0]) return;
    rec_pos = root.getPtr(pos.pos[1]);

    Page& p = *new Page;
    p.ofs = rec_pos.pos[0];
    p.pageType = RecordPage;
    p.tableName = t.getName();

    index_head.bm.readPage(p);
    tp = rm.Char_Tuper(t, (char*)(p.pageData + rec_pos.pos[1]));

    t.data.push_back(tp);

    delete& root;
    delete& p;
}
```

6.范围查找

```
void IndexManager::range_select(
    const Data* key1, const Data* key2,
    bool bound1, bool bound2,
    Table& t) {
    int i;
    Pos pos, rec_pos;
    Tuper* tp;
    RecordManager rm(nullptr);
    int root_pos = index_head.getRootPos();
    if (root_pos == index_head.NULL_POS) return;

    BptNode& start =
        *new BptNode(
            index_head.getFilename(),
            root_pos,
            max_degree
        );

    Page& p = *new Page;
    Data* K;
    p.pageType = RecordPage;
    p.tableName = t.getName();
    pos.pos[1] = 0;
    pos.pos[0] = -1;
    if (key1 == nullptr) {
        K = key2->new_copy();
        ReachFirstLeaf(start);
    }
    else {
        K = key1->new_copy();
        pos = find(*key1, start);
    }
    if (pos.pos[1] == pos.pos[0] && bound1 == false) {
        pos.pos[1]++;
    }
```

```
bool loop=true;
do {
    start.load(index_head.bm);
    for (i = pos.pos[1]; i < start.getNum(); i++) {
        start.getKey(*K,i);
        if ((key2)&&(*key2<*K || *key2==*K && !bound2)){
            loop=false;
            break;
        }
        rec_pos = start.getPtr(i);
        p.ofs = rec_pos.pos[0];
        index_head.bm.readPage(p);
        tp = rm.Char_Tuper(t, (char*)(p.pageData + rec_pos.pos[1]));
        t.data.push_back(tp);
    }
    pos.pos[1]=0;
} while (loop && start.being_next());
```

2. 阐述模块的测试代码

测试流程：

建立一个索引，随机插入 2000 个键值，然后保存索引文件；

打开索引文件，输出所有数据，一个一个删除所有索引，保存索引文件。

为更好地体现代码漏洞，将 B+树叉数设为 4

主要代码如下图：

```
int main() {
    int i;
    cin >> i;
    if(i)insert_test();
    else delete_test();
    return 0;
}
```

```
void delete_test() {
    DataInt key(0);
    int a[13];
    int i;
    int j,aj;
    for (i = 0; i < 13; i++) a[i] = rand();

    try {
        IndexManager& im =
            *(new IndexManager(
                "result/index1",
                key.size(),
                false
            )
        );

        for (i = 0; i < sizeof(a) / sizeof(a[0]); i++) {
            j = rand() % (13-i);
            aj = a[j];
            a[j] = a[12 - i];
            key.x = aj;
            im.remove(key);
            cout << "key " << i <<":"<<aj<< " deleted" << endl;
            im.show_leaves();
        }
        delete& im;
    }
    catch (exception& e) {
        cout << e.what() << endl;
    }
}
```

```
void insert_test() {
    DataInt key(3);
    int i;

    fstream file("result/index1.record", ios::out);
    if (file.fail()) {
        cout << "failed to open file";
        return;
    }
    file.close();

    //srand(time(nullptr));

    IndexManager& im =
        *(new IndexManager("result/index1", key.size(), true));
    try {
        for (i = 0; i < 13; i++) {
            key.x = rand();
            im.insert(key, i, 0);
            //cout << "key " << i << " inserted" << endl;
        }
        im.show_leaves();
        //im.show_index_file();
        delete& im;
        cout << "insert finished" << endl;
    }
    //catch (IndexException& ie) {
    //  cout << ie.what()<<endl;
    //  cout << "iteration:" << i << endl;
    //  im.show_index_file();
    //  delete& im;
    //}
    catch (exception& e) {
        cout << e.what() << endl;
    }
}
```

经测试调试，结果基本无误。

关于查找的功能由于涉及到 RecordManager，正确性由小组整体进行测试

六、 遇到的问题及解决方法

1. 代码编译错误

由于直到把全部代码都写好了才进行编译，产生了许多潜在的语法错误，错误与错误交织，无从下手。

解决方法：

采用增量检查编译错误的方法，即一开始只编译一个文件，检查错误；在其基础上再增加一个文件，一起编译，解决新增的错误，从而各个击破。

2. 文件循环包含

编译提示找不到一些类型的定义

解决方法：找到那些类型定义所在的头文件，确认它们确实被编译了。发现头文件 A 包

含了头文件 B，头文件 B 又包含了头文件 A。由于只包含一次的效果，B 中的 A 可能不会展开。因此可能导致 B 中依赖 A 的定义缺失。将一些头文件放在 cpp 文件中包含即可

3.设计索引插入、删除时的各种错位，与考虑不周

解决方法：通过 debug，耐心进行断点、单步调试，一步步找到问题所在。

4.BufferManager 出现的一些问题

虽然不是 Index 负责的模块，但它的问题与 Index 息息相关。因此帮助修正了其中许多的重要漏洞。

七、 总结

MiniSQL 是我遇到的又一个大作业。虽然我对自己 index 的完成度还是比较满意的，但是此次小组合作的作品还是有不少的漏洞。但大程带给我的并不只有挫败感，我还吸取了很多的经验：设计大程时一定要定一个框架，像模具一样慢慢填充，可以进行单元的测试，而不是东拼一块西拼一块，最后不一定能拼成完美的一块；加强小组之间的沟通交流，增加线下见面次数，互相监督，互相交流；利用好调试工具，就大程方面，我认为使用 VS 比起 Vscode 更加便利一些。我认为，吸取了这一次大程的经验，我一定能在以后的专业课以至以后的工作中合作得更好。