Index Manager 设计报告

计算机科学与技术学院数媒 1601 班 王立冬 316010170

一、 模块概述

MiniSQL的整体设计要求对于表的主属性自动建立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）。

Index Manager负责B+树索引的实现，实现B+树的创建和删除（由索引的定义与删除引起）、等值查找、插入键值、删除键值等操作，并对外提供相应的接口。（B+树中节点大小应与缓冲区的块大小相同，B+树的叉数由节点大小与索引键大小计算得到。）

二、 主要功能

1.创建B+树索引：根据索引名和键长创建B+树索引。

2. 删除B+树索引：根据索引名删除索引，若失败输出创建失败的语句。

3. 等值查询：根据索引名和键值查找存有该键值在RecordManager中地址信息的一个整数。若成功，返回该整数，若失败，输出失败原因。

4. 插入键值：根据索引名、键值和整数地址向对应B+树插入一个值。若失败，输出失败原因。

5. 删除键值：根据索引名和键值在对应B+树中删除一个值。若失败，输出失败原因。

三、 对外提供的接口

1. 创建B+树索引：

```
bool CreateIndex(const char* filename, int KeyLength);
//需要API提供索引名以及键长信息
```

2.删除B+树索引：

```
bool DropIndex(const char* filename);
//需要API提供索引名信息
```

3.等值查询：

```
int Find(const char* filename, const char* key);
//需要API提供索引名以及键值信息
```

4.插入键值

```
bool Insert(const char* filename, const char* key, const int value);
//需要API提供索引名、键值以及整数地址信息
```

5.删除键值

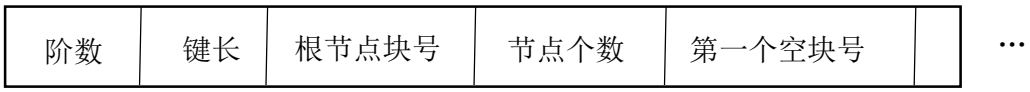
```
bool Remove(const char* filename, const char* key);
//需要API提供索引名以及键值信息
```

四、 设计思路

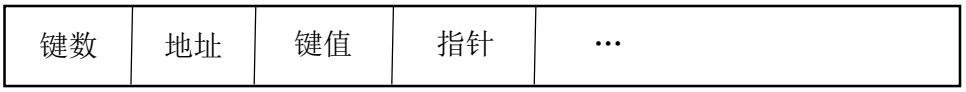
IndexManager模块总共分为三个结构来实现，分别为IndexManager、BpTree、BpTreeNode。IndexManager结构作为IndexManager模块的管理器，提供对外的接口函数，这些接口函数通过对BpTreeNode和BpTree里的成员变量进行操作实现对B+树的操作。

数据结构：一个节点一个块，一个B+树索引一个文件。

Block 0:



Block 1 ~ N:



函数思路：

当创建B+树索引的接口函数被调用时，IndexManager会创建一个以索引名命名的mdb文件，然后通过BufferManager模块读取该mdb文件的第一个块，将文件头信息写入到该块中。文件头信息包含B+树阶数（int，由键长、块的大小以及其他一些数据计算）、键长（int）、根节点的块号（int）、节点个数（int）、第一个空块号（int）。

当删除B+树索引的接口函数被调用时，IndexManager会删除对应索引名的mdb文件。

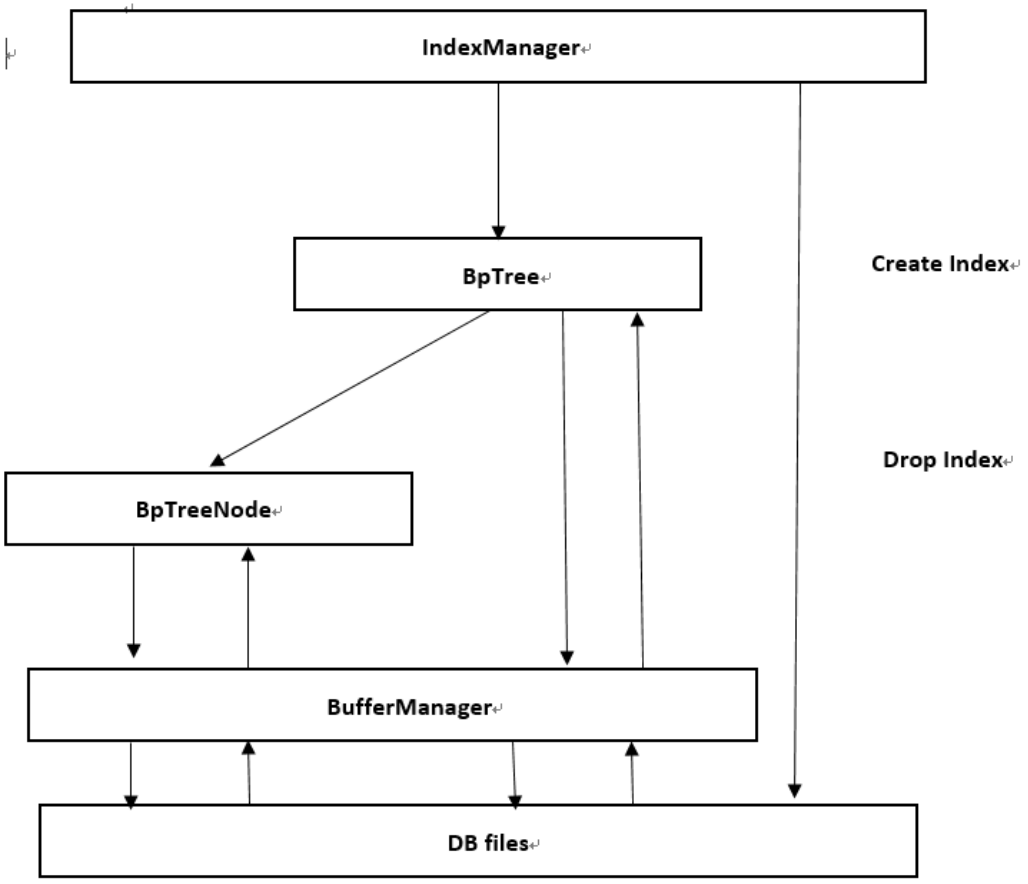
当等值查询的接口函数被调用时，IndexManager会以索引名构造BpTree结构。BpTree构造函数调用时，通过BufferManager模块读取对应mdb文件的第一个块（文件头信息所在的块），将该块里的信息加载到成员变量中。而后BpTree结构会调用

本结构下的等值查询函数，在这里，我们命名它为等值查询k。等值查询k会以本结构下的索引名、键长、块号（第一次为根节点的块号，后面为等值查询k返回的块号，直到结束该查询）构造BpTreeNode结构，BpTreeNode构造函数会通过BufferManager模块读取对应块号的块，然后将块中的数据加载到成员变量中。而后BpTreeNode结构会调用本结构下的等值查询函数，在这里，我们命名它为等值查询m。等值查询m会遍历本结构下的数据，返回结果。

当插入键值的接口函数被调用时，与上述等值查询相似。IndexManager会构造BpTree读取文件头，BpTree根据需求获得对应块的BpTreeNode结构，直到将键值插入，若出现需要分裂节点的情况，BpTree会进行一系列操作完成。

删除键值的思路与插入相同。

五、 整体架构



六、 关键函数和代码

以下贴上了BpTree和BpTreeNode的构造函数、析构函数以及Insert函数。构造、析构函数涉及到了块中数据的读写操作，而Insert函数则是B+树中最典型的操作，其中包含了节点的分裂操作。通过这些函数，能够较快较直观的了解IndexManager模块。

1.BpTree构造函数

```
//Read HeadFile
BpTree::BpTree(const char* filename):_filename(filename)
{
	BufferManager* buffermanager = MiniSQL::get_BM();
	Block* header = buffermanager->get_block(filename, 0);
	char* data = header->data;

	order = *(reinterpret_cast<int*>(data));
	KeyLength = *(reinterpret_cast<int*>(data + 4));
	root = *(reinterpret_cast<int*>(data + 8));
	NodeCount = *(reinterpret_cast<int*>(data + 12));
	FirstEmpty = *(reinterpret_cast<int*>(data + 16));
}
```

2.BpTree析构函数

```
//Update block
BpTree::~BpTree()
{
	BufferManager* buffermanager = MiniSQL::get_BM();
	Block* block = buffermanager->get_block(_filename.c_str(), 0);
	char* data = block->data;

	memcpy(data + 8, &root, 4);
	memcpy(data + 12, &NodeCount, 4);
	memcpy(data + 16, &FirstEmpty, 4);

	block->dirty = true;
}
```

3.BpTree Insert函数

```
//Insert key
bool BpTree::Insert(const char* key, const int value, const int type)
{
	int res;
	bool isleaf;
	BpTreeNode* sp;
	BpTreeNode* node;

	if (root == -1)
	{
		BufferManager* buffermanager = MiniSQL::get_BM();
		Block* block = buffermanager->get_block(_filename.c_str(), FirstEmpty);
		char* data = block->data;

		int size = 0;
		int ptr = -1;
		memcpy(data, &size, 4);
		memcpy(data + 4, &ptr, 4);
		block->dirty = true;

		node = new BpTreeNode(_filename.c_str(), FirstEmpty, KeyLength);
		NodeCount++;
		root = GetFirstEmpty();
		delete node;

		return Insert(key, value);
	}

	res = root;
	vector<int> Value;
	int ValueLength = 0;
	do
	{
		Value.push_back(res);
		ValueLength++;
		node = new BpTreeNode(_filename.c_str(), res, KeyLength);
		res = node->Insert(key,order,value,type);
		if (res == -2)
		{
			sp = node;
			node = NULL;
		}
		delete node;
```

```
} while (res > 0);
	if (res == -3)
	{
		cerr << "Error:[IndexManager]This key already existed." << endl;
		return false;
	}
	else if (res == -2)//Split into two nodes. Return new node
	{
		int ret = GetFirstEmpty();
		if (ValueLength == 1)
		{
			node = sp->Split(ret);
			ret = GetFirstEmpty();
			BufferManager* buffermanager = MiniSQL::get_BM();
			Block* block = buffermanager->get_block(_filename.c_str(), ret);
			char* data = block->data;

			int size = 1;
			int pointer_0 = sp->GetId();
			int pointer_1 = node->GetId();
			char* Key = new char[KeyLength];
			memcpy(Key, node->GetKey(), KeyLength);
			memcpy(data, &size, 4);
			memcpy(data + 4, &pointer_0, 4);
			memcpy(data + 8, Key, KeyLength);
			memcpy(data + 8 + KeyLength, &pointer_1, 4);

			block->dirty = true;

			BpTreeNode* new_node = new BpTreeNode(_filename.c_str(), ret,
KeyLength);
			delete Key;
			delete new_node;
			delete node;
			delete sp;
			NodeCount = NodeCount + 2;
			root = ret;
			return true;
		}
		else if(ValueLength >= 2)
		{
			bool flag;
			node = sp->Split(ret);
```

```
char* Key = new char[KeyLength];
                memcpy(Key, node->GetKey(), KeyLength);
                int pointer = node->GetId();
                flag = Insert(Key, pointer, Value[ValueLength - 2]);
                delete node;
                delete sp;
                delete Key;
                NodeCount++;
                return flag;
            }
        }
        else if (res == -1)
            return true;
}
```

4.BpTreeNode构造函数

```
//Read Block
BpTreeNode::BpTreeNode(const char* filename, int value, int
KeyLength) :length(KeyLength), _filename(filename), id(value)
{
        BufferManager* buffermanager = MiniSQL::get_BM();
        Block* block = buffermanager->get_block(filename, value);
        char* data = block->data;

        size = *(reinterpret_cast<int*>(data));
        ptr.push_back(*(reinterpret_cast<int*>(data + 4)));
        isleaf = ptr[0] < 0;
        int i = 0;
        for (i; i < size; i++)
        {
            char* k = new char[KeyLength];
            memcpy(k, data + 8 + i * (KeyLength + 4), KeyLength);
            keys.push_back(k);
            ptr.push_back(*(reinterpret_cast<int*>(data + 8 + KeyLength + i * (KeyLength +
4))));
        }
}
```

5.BpTreeNode析构函数

```
//Update Block
BpTreeNode::~BpTreeNode()
{
	BufferManager* buffermanager = MiniSQL::get_BM();
	Block* block = buffermanager->get_block(_filename.c_str(), id);
	char* data = block->data;

	if(size == -1)
		memcpy(data, "ptr", 4);
	else
		memcpy(data, &size, 4);
	memcpy(data + 4, &ptr[0], 4);

	int i = 0;
	for (i; i < size; i++)
	{
		memcpy(data + 8 + i * (length + 4), keys[i], length);
		memcpy(data + 8 + length + i * (length + 4), &ptr[i + 1], 4);
		delete keys[i];
	}

	block->dirty = true;
}
```

6.BpTreeNode Insert函数

```
int BpTreeNode::Insert(const char* key, const int order, const int value, const int type)
{
	int i = 0;
	int flag = size - 1;
	if (isleaf)
	{
		for (i; i < size; i++)
		{
			if (strncmp(key, keys[i], length) > 0 && size < order - 1)
			{
				char* k = new char[length];
				memcpy(k, key, length);
				keys.push_back(keys[size - 1]);
				ptr.push_back(ptr[size]);
				for (flag; flag > i; flag--)
				{
					keys[flag] = keys[flag - 1];
```

```
ptr[flag + 1] = ptr[flag];
                        }
                        keys[i] = k;
                        ptr[i + 1] = value;
                        size++;
                        if (type > 0)isleaf = false;
                        return -2;
                    }
                    else if (strncmp(key, keys[i], length) == 0)return -3;
                }
                if (size < order - 1)
                {
                    char* k = new char[length];
                    memcpy(k, key, length);
                    keys.push_back(k);
                    ptr.push_back(value);
                    size++;
                    if (type > 0)isleaf = false;
                    return -1;
                }else if (size == order - 1)                    //Need to create new node
                {
                    char* k = new char[length];
                    memcpy(k, key, length);
                    keys.push_back(k);
                    ptr.push_back(value);
                    size++;
                    if (type > 0)isleaf = false;
                    return -2;
                }
            }else
            {
                if (type > 0 && type == id)
                {
                    isleaf = true;
                    return Insert(key, order, value, type);
                }
                for (i; i < size; i++)
                {
                    if (strncmp(key, keys[i], length) > 0)return ptr[i];
                }
                return ptr[i];
            }
}
```