INDEX MANAGER实验报告

姓名： 张鑫

学号：3200102809

专业：计算机科学与技术

0 绪论

本模块由我一人完成。在本模块中，我们需要实现一个基于磁盘的动态B+树索引，对数据表索引进行管理，实现了索引的创建和删除，索引键的等值查找，索引键的范围查找（返回对应的迭代器），以及插入和删除键值等操作，并对外提供了相应的接口。

1 B+树数据页

1.1 B Plus Tree Page

B Plus Tree Page是B+树叶节点和中间节点的公共父类，继承自Page，维护了如下属性：

```
[[maybe_unused]] IndexPageType page_type_;
[[maybe_unused]] lsn_t lsn_;
[[maybe_unused]] int size_;
[[maybe_unused]] int max_size_;
[[maybe_unused]] page_id_t parent_page_id_;
[[maybe_unused]] page_id_t page_id_;
```

其中page_type_为枚举类型，有非法节点，叶节点和内部节点三种取值。

size_是当前数据页中的索引键值对的数量，max_size_ 为容量。

parent_page_id_ 是该节点的父节点的逻辑页地址，若该节点是根节点，则此属性为INVALID_PAGE_ID_。

page_id_为该页本身的逻辑页号。

在该类中，我们可以为B+树页节点设置公共方法如各私有属性的读写。

1.2 B Plus Tree Leaf Page

叶结点BPlusTreeLeafPage存储实际的数据，它按照顺序存储个键和个值，其中键由一个或多个Field序列化得到（参考#3.2.4），在BPlusTreeLeafPage类中用模板参数KeyType表示；值实际上存储的是RowId的值，它在BPlusTreeLeafPage类中用模板参数ValueType表示。叶结点和遵循着键值对数量的约束，同样也需要完成对应的合并、借用和分裂操作。

在叶节点中，笔者并没有采用助教建议的std::pair的键值对方式，而是将二者用分离数组实现。

```
page_id_t next_page_id_;
KeyType key_[LEAF_PAGE_SIZE];
ValueType value_[LEAF_PAGE_SIZE];
```

这样的设计可以利用空间局部性原理提升CPU缓存命中率从而使得索引操作更高效。

另外，B+树的叶节点由单向链表组织，由next_page_id_连接，可以线性迭代。

1.3 B Plus Tree Internal Page

B+树的中间节点不存储数据，只存储键值和对应的子节点的逻辑页号，由于子节点的数量比键值多1，于是笔者采用了如下设计。

```
KeyType key_[INTERNAL_PAGE_SIZE];
page_id_t value_[INTERNAL_PAGE_SIZE + 1];
```

和1.2中原理相同，利用局部性原理。

2 B+ Tree Insert

B+树的插入作分类讨论如下：

- 若B+树根节点为叶节点，直接插入根节点
- 若B+树根节点为中间节点，则迭代找到对应插入键值的叶节点插入。
- 若键值已经在叶中存在，结束操作

插入后的操作：

- 若页节点大小小于MaxSize，结束操作
- 否则对叶节点执行分裂操作，传出右边叶子的最小键值，插入父节点维护B+平衡。

插入父节点后的操作：

- 若父节点大小小于MaxSize，结束操作
- 否则对internal page执行分裂操作，额外将middle key分裂出，插入父节点，进行递归调用
- 若该internal page是根节点，则执行分裂后额外分配新的根节点，连接。
- 注意分裂时，要更新子节点的父节点值

3 B+ Tree Delete

B+树的删除操作步骤如下：

- 首先找到删除键值对应的叶节点
- 若键值不存在于节点中，结束操作，否则删除对应键值
- 检查删除后的节点大小，若小于节点最小大小，则检查临近节点

    - 若该节点是父节点的最左节点，选取右侧sibling为临近节点
    - 否则选取左侧节点为临近节点
- 找到临近节点后
    - 若临近节点大小大于MinSize, 从临近节点转移一个键值对到该节点，更新父节点中的键值
    - 若临近节点大小等于MinSize,将该节点和临近节点和父节点取出的对应键值合并成新的节点，递归检查父节点大小。
- 若该节点为根节点大小为0, 则删除该节点。

4 B+ Tree Search

B+树的等值查询通过迭代找到键值对应的叶子节点，线性扫描叶子节点得到值。

对于范围查询，对范围的端点创建迭代器，利用迭代器线性遍历进行范围查询。

5 B+ Tree Iterator

B+树迭代器成员如下：

```
int index_;
MappingType val_;
B_PLUS_TREE_LEAF_PAGE_TYPE* leaf_page_;
BufferPoolManager* buffer_pool_manager_;
```

index_是该迭代器指向内容在叶节点中的下标。

val为std::pair<key,value>，是迭代器指向内容的只读。（B+索引的迭代器是只读的，这是因为B+树的键值的更改必须要通过B+树的插入和删除完成，因而B+树执行插入删除可能使得迭代器失效。）

6 B+ Tree Index

B+树索引是通过 B+ tree index这个类向外部暴露的，另外B+树在根节点改变时要通过存储与INDEX_ROOT_PAGE_ID页的IndexRootPage更改索引Id到索引根节点的映射。

7 Source Code

由于源码过大，见附录。