3 INDEX MANAGER

3.1 实验概述

Index Manager 负责数据表索引的实现和管理，包括：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，并对外提供相应的接口。

在上一个实验中，同学们应该能够发现，通过遍历堆表的方式来查找一条记录是十分低效的。为了能够快速定位到某条记录而无需搜索数据表中的每一条记录，我们需要在上一个实验的基础上实现一个索引，这能够为快速随机查找和高效访问有序记录提供基础。索引有很多种实现方式，如B+树索引，Hash索引等等。在本实验中，需要同学们实现一个基于磁盘的B+树动态索引结构。

3.2 B+树数据页

B+树中的每个结点（Node）都对应一个数据页，用于存储B+树结点中的数据。因此在本节中，你需要实现以下三种类型的B+树结点数据页：

3.2.1 BPlusTreePage

`BPlusTreePage`是`BPlusTreeInternalPage`和`BPlusTreeLeafPage`类的公共父类，它包含了中间结点和叶子结点共同需要的数据：

- `page_type_`: 标记数据页是中间结点还是叶子结点；
- `key_size_`: 当前索引键的长度；
- `lsn_`: 数据页的日志序列号，该模块中不会用到；
- `size_`: 当前结点中存储Key-Value键值对的数量；
- `max_size_`: 当前结点最多能够容纳Key-Value键值对的数量；
- `parent_page_id_`: 父结点对应数据页的`page_id`；
- `page_id_`: 当前结点对应数据页的`page_id`。

你需要在`src/include/page/b_plus_tree_page.h`和`src/page/b_plus_tree_page.cpp`中实现`BPlusTreePage`类。

3.2.2 BPlusTreeInternalPage

中间结点`BPlusTreeInternalPage`不存储实际的数据，它只按照顺序存储m个键和$m+1$个指针（这些指针记录的是子结点的`page_id`）。由于键和指针的数量不相等，因此我们需要将第一个键设置为INVALID，也就是说，顺序查找时需要从第二个键开始查找。在任何时候，每个中间结点至少是半满的（Half Full）。当删除操作导致某个结点不满足半满的条件，需要通过合并（Merge）相邻两个结点或是从另一个结点中借用（移动）一个元素到该结点中（Redistribute）来使该结点满足半满的条件。当插入操作导致某个结点溢出时，需要将这个结点分裂成为两个结点。

你需要在`src/include/page/b_plus_tree_internal_page.h`和`src/page/b_plus_tree_internal_page.cpp`中实现`BPlusTreeInternalPage`类。

Note: 为了便于理解和设计，我们将键和指针以`pair`的形式顺序存储，但由于键和指针的数量不一致，我们不得已牺牲一个键的空间，将其标记为INVALID。也就是说对于B+树的每一个中间结点，我们都付出了一个键的空间代价。实际上有一种更为精细的设计选择：定义一个大小为m的数组连续存放键，然后定义一个大小为$m+1$的数组连续存放指针，这样设计的好处在于，一是没有空间上的浪费，二是在键值查找时CPU缓存的命中率较高（局部性原理）。学有余力的同学可以尝试着使用这种方式去实现。

3.2.3 BPlusTreeLeafPage

叶结点`BPlusTreeLeafPage`存储实际的数据，它按照顺序存储m个键和m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。叶结点和中间结点一样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。

你需要在`src/include/page/b_plus_tree_leaf_page.h`和`src/page/b_plus_tree_leaf_page.cpp`中实现`BPlusTreeLeafPage`类。

3.2.4 Key、Value & KeyManager

`Key`: 索引键是索引列的值序列化后得到的字符串。如BPlusTreeIndexGenericKeyTest中所示，对于一个有三列（id，name，account）的表，索引（id，name）的键即是两列的值（例如27，“minisql”）序列化后的字符串。索引列的长度作为参数在构造BPlusTreeIndex时作为参数传入，保存在各个节点中，方便根据`key_size`确定每个键值对在模板中的位置，从而读写。

> Value`: 值类型可能不同，叶结点存储`RowId`，而非叶结点存储`page_id

`KeyManager`: 负责对`GenericKey`进行序列化/反序列化和比较，注意比较时传入的是`GenericKey*`指针，指针指向的内容可能在插入删除时随着B+树结构变动被修改。

```
TEST(BPlusTreeTests, BPlusTreeIndexGenericKeyTest) {
  DBStorageEngine engine(db_name);
  std::vector<Column *> columns = {new Column("id", TypeId::kTypeInt, 0, false, false),
                                   new Column("name", TypeId::kTypeChar, 64, 1, true,
false),
                                   new Column("account", TypeId::kTypeFloat, 2, true,
false)};
  std::vector<uint32_t> index_key_map{0, 1};
  const TableSchema table_schema(columns);
  auto *key_schema = Schema::ShallowCopySchema(&table_schema, index_key_map);
  std::vector<Field> fields{Field(TypeId::kTypeInt, 27),
                            Field(TypeId::kTypeChar, const_cast<char *>("minisql"), 7,
true)};
  KeyManager KP(key_schema, 128);
  Row key(fields);
  GenericKey *k1 = KP.InitKey();
  KP.SerializeFromKey(k1, key, key_schema);
  GenericKey *k2 = KP.InitKey();
  Row copy_key(fields);
  KP.SerializeFromKey(k2, copy_key, key_schema);
  ASSERT_EQ(0, KP.CompareKeys(k1, k2));
}
```

对于B+树中涉及到的索引键的比较，由于`GenericKey`对象并不是基本数据类型，因此不能够直接使用比较运算符`>`、`<`等进行比较（除非对传入的对象的比较运算符进行重载，但这种设计方式难以应对需要不同比较方式的场景）。为此，我们需要借助`KeyManager`中的`CompareKeys`方法对两个索引键进行比较。以下是一个例子：

```
void Example(GenericKey *k1, GenericKey *k2, KeyManager &KM) {
    if (KM.CompareKeys(k1, k2) > 0) {
        // k1 > k2
    } else if (KM.CompareKeys(k1, k2) < 0) {
        // k1 < k2
    } else {
        // k1 == k2
    }
}
```

`CompareKeys`的实现在框架中已经给出（在`src/include/index/generic_key.h`中定义），其基本原理是，对于两个待比较的索引键`GenericKey`（为了将索引键存储到B+树数据页中，需要将索引键进行序列化，也就是说`GenericKey`内部实际上存储的是索引键序列化后得到的字符串，参考下面代码中`GenericKey`类的定义），首先将其按照索引键定义的模式`key_schema_`进行反序列化，然后对反序列化得到的每一个域`Field`，调用`Field`的比较函数进行比较。`Field`类型的比较函数已经在代码框架中给出，具体细节请同学们自行学习了解。

```
class GenericKey {
    friend class KeyManager;
    // actual location of data, extends past the end.
    char data[0];
}

inline void SerializeFromKey(GenericKey *key_buf, const Row &key) const;

inline void DeserializeToKey(const GenericKey *key_buf, Row &key) const;

inline int GenericComparator::CompareKeys(const GenericKey *lhs, const GenericKey *rhs)
const
{
    uint32_t column_count = key_schema_->GetColumnCount();
    Row lhs_key(INVALID_ROWID);
    Row rhs_key(INVALID_ROWID);
    DeserializeToKey(lhs, lhs_key);
    DeserializeToKey(rhs, rhs_key);

    for (uint32_t i = 0; i < column_count; i++)
    {
      Field *lhs_value = lhs_key.GetField(i);
      Field *rhs_value = rhs_key.GetField(i);
        if (lhs_value->CompareLessThan(*rhs_value) == CmpBool::kTrue)
        return -1;

      if (lhs_value->CompareGreaterThan(*rhs_value) == CmpBool::kTrue)
        return 1;
    }
    // equals
    return 0;
```

```
}
```

3.2.5 Some Tips

- `BPlusTreePage::GetMinSize()` 所返回的值通常情况下为 `max_size_/2`，但它实际上对于叶子结点/非叶结点/根结点/非根结点可能会有所不同。且 `size` 的概念通常情况下表示的是指针的数量（即结点中键值对的数量），换而言之，在中间结点中，包含$k-1$个键和k个指针的 `size` 为k。
- `BPlusTreePage` 中的内容实际上存储于 `Page` 中的 `data_`，每当需要对B+树的数据页进行读写时，首先需要从 `BufferPoolManager` 中获取（`Fetch`）这个页，此时拿到的数据页为 `Page` 类型，但我们需要用到的数据页 `BPlusTreeInternalPage` 和 `BPlusTreeLeafPage` 是 `BPlusTreePage` 类的子类，`BPlusTreePage` 类和 `Page` 类的 `data_` 域在内存分布上是相同的（通俗来说，`data_` 域中 `PAGE_SIZE` 个字节存放的就是 `BPlusTreePage` 对象），因此需要通过 `reinterpret_cast` 将 `Page` 中的 `data_` 重新解释成为我们需要使用的类。最后，在使用完毕后需要将该页释放（`Unpin`），以下是一个使用 `reinterpret_cast` 将 `Page` 类的 `data_` 域重新解释成 `BPlusTreeInternalPage` 对象例子：

```
auto *page = buffer_pool_manager->FetchPage(page_id);
if (page != nullptr) {
    auto *node = reinterpret_cast<BPlusTreeInternalPage *>(page->GetData());
    /* do something */
    buffer_pool_manager->UnpinPage(page_id, true);
}
```

- 在不需要使用数据页时，请务必将其释放，我们将会在测试代码中加入 `CheckAllUnpinned()` 机制检查所有的数据页最终是否被释放。
- 在 `UpdateRootPageId` 函数中，有关root page的定义在 `include/page/index_roots_page.h` 中
- `BPlusTree::BPlusTree` 函数中，如果传入的 `leaf_max_size` 和 `internal_max_size` 是默认值0，即 `UNDEFINED_SIZE`，那么需要自己根据keysize进行计算

3.3 B+树索引

在完成B+树结点的数据结构设计后，接下来需要完成B+树的创建、插入、删除、查找和释放等操作。注意，所设计的B+树只能支持 `Unique Key`，这也意味着，当尝试向B+树插入一个重复的Key-Value键值对时，将不能执行插入操作并返回 `false` 状态。当一些写操作导致B+树索引的根结点发生变化时，需要调用 `BPLUSTREE_TYPE::UpdateRootPageId` 完成 `root_page_id` 的变更和持久化。

Note：在 `**UpdateRootPageId**` 函数中，有关root page的定义在 `**include/page/index_roots_page.h**` 中

你需要在 `src/include/index/b_plus_tree.h` 和 `src/index/b_plus_tree.cpp` 中实现整个 `BPlusTree` 类。其中一些方法如Coalesce、Redistribute根据传入参数类型不同（LeafPage or InternalPage）需要实现两个方法，看起来很多，但大体逻辑是类似的，细微处需要根据是叶子结点还是内部节点作出修改。

在实现 `BPlusTree` 时，你无需考虑 `GenericKey`、`KeyManager` 的实现，与它们相关的类已经实现，位于 `src/include/index/generic_key.h` 中。`KeyManager` 的实例将会随 `BPlusTreeIndex` 一起构造。

3.4 B+树索引迭代器

与堆表`TableHeap`对应的迭代器类似，在本节中，你需要为B+树索引也实现一个迭代器。该迭代器能够将所有的叶结点组织成为一个单向链表，然后沿着特定方向有序遍历叶结点数据页中的每个键值对（这在范围查询时将会被用到）。

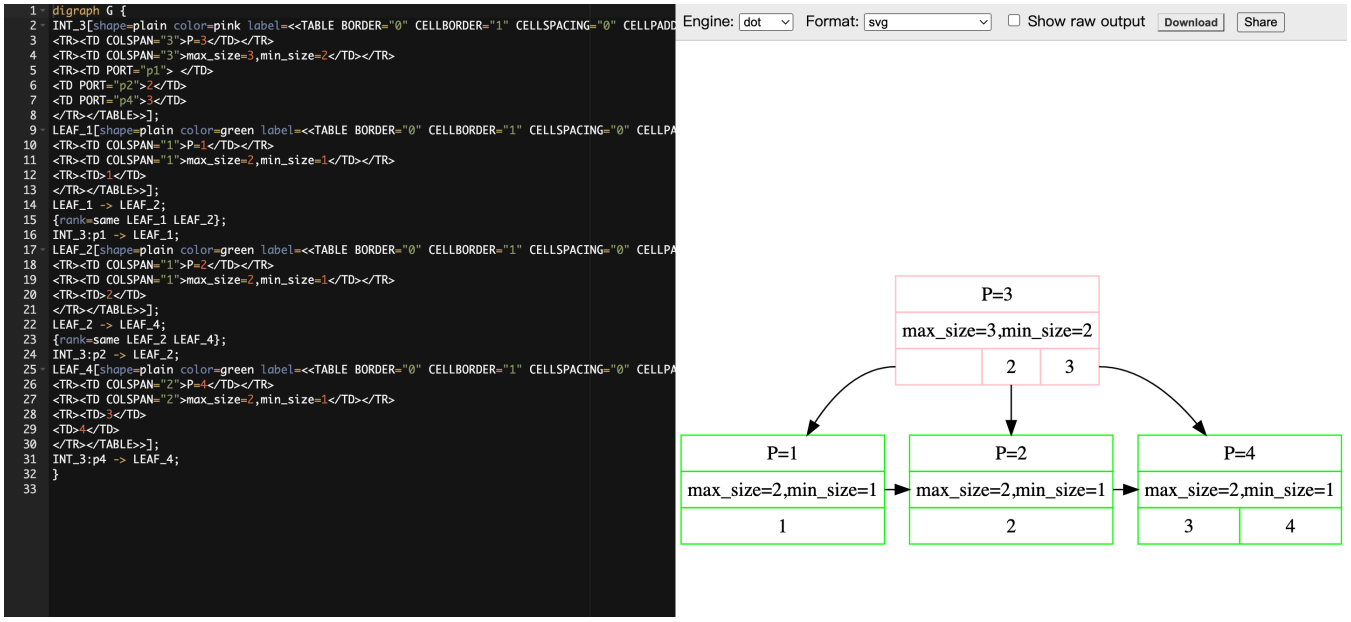
你需要在`src/include/index/index_iterator.h`和`src/index/index_iterator.cpp`中实现B+树索引的迭代器`IndexIterator`。同样地，你需要在`BPlusTree`类中实现`Begin()`和`End()`函数以获取B+树索引的首迭代器和尾迭代器。

3.5 模块相关代码

- `src/include/page/b_plus_tree_page.h`
- `src/page/b_plus_tree_page.cpp`
- `src/include/page/b_plus_tree_internal_page.h`
- `src/storage/page/b_plus_tree_internal_page.cpp`
- `src/include/page/b_plus_tree_leaf_page.h`
- `src/storage/page/b_plus_tree_leaf_page.cpp`
- `src/include/storage/index/b_plus_tree.h`
- `src/storage/index/b_plus_tree.cpp`
- `src/include/storage/index/index_iterator.h`
- `src/storage/index/index_iterator.cpp`
- `test/index/b_plus_tree_index_test.cpp`
- `test/index/b_plus_tree_test.cpp`
- `test/index/index_iterator_test.cpp`

3.6 开发提示

1. 推荐在**夏学期第4周前**完成本模块的设计。
2. 这是一个展现B+树插入和删除操作的可视化网站，可以帮助熟悉B+树的相关操作：链接
3. 在调试时，可以通过`BPlusTree::PrintTree(std::ofstream &out)`将B+树的结构以DOT格式输出到输出流中，然后可以通过一个可视化网站：链接，查看当前B+树的状态。具体的使用方法可以参考测试模块中给出的代码。



3.7 诚信守则

1. 请勿从其它组或在网络上找到的其它来源中复制源代码，一经发现抄袭，成绩为`0`；
2. 请勿将代码发布到公共Github存储库上。

3.8 评论

熊儒海2022-05-15 20:42

在b_plus_tree_leaf_page.cpp中 MoveLastToFrontof等函数并没有像b_plus_tree_internal_page.cpp中要求的那样更新parent的separation key,我觉得这有些不妥，并且两个cpp文件中的MoveLastToFrontof等函数参数表不一致，这似乎会影响到模板的使用

fakeyyy2022-05-18 18:44

leaf里面的RemoveAndDeleteRecord要求return page size after deletion 是不是写错了？应该返回key & value pair 的数量?

Brucecai2022-05-21 21:42

可能是current page size?

fakeyyy2022-05-19 00:08

文档中提到BPlusTreePage类和Page类的data*域在内存分布上是相同的（通俗来说，data*域中PAGE_SIZE个字节存放的就是BPlusTreePage对象，但是为什么在b_plus_tree.h中的PrintTree中出现了如下代码
Page *root_page = buffer_pool_manager->*FetchPage(root_page_id*);
BPlusTreePage *node = reinterpret_cast<BPlusTreePage *>(root_page);
这里并没有对root_page->GetData()，请问是为什么？

YingChengJun2022-05-19 11:01

这里是应该用root_page->GetData()的。没出问题是由于data_域在这个结构的内存排布中刚好位于最前面，这个之后会改一下。

余浩鸣H. Yu2022-05-19 10:17

引用原文：叶结点BPlusTreeLeafPage存储实际的数据，它按照顺序存储[公式]个键和[公式]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。叶结点和中间结点一样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。

b_plus_tree_leaf_page.h中应该是28个字节的头文件长度，LSN继承自b_plus_tree_page类，但是注释中没有体现

余浩鸣H. Yu2022-05-19 10:18

注释中是24个字节

- +1 2

应该是32，还有next_page_id . 宏的定义是对的

HAL 90002022-05-21 14:21

在b_plus_tree.cpp的Insert函数中第一次遇到要求抛出exception，项目没有自定义exception类，抛出std::exception()就行吗

YingChengJun2022-05-21 15:52

可以的。也可以不抛出Exception，直接ASSERT(false, "错误信息")让它强制挂掉。

Brucecai2022-05-23 04:28

刚写完索引部分，给大家一个建议，最好自顶向下看一下，并且最好能画出每一个page中的函数之间关系图，理清楚了，写起来轻松多了。

YingChengJun2022-05-23 11:46

很棒。通常在做工程的时候，如果能够形成一种自顶向下的认识，在对整体的系统或者模块有一定的理解的前提下，再回过头来进行自底向上的实现会轻松很多。
PS：以后学习计算机网络这门课程也是，自顶向下进行学习会更简单易懂。

gkelp2022-05-25 10:38

这里src/index/b_plus_tree.cpp中的BPLUSTREE_TYPE::UpdateRootPageId的代码注释写错了，header page的page_id应当是INDEX_ROOTS_PAGE_ID而并非0，不然会在下一个部分和CATALOG_META_PAGE_ID冲突（

ianafp2022-05-29 14:10

建议测试大规模数据的时候一定要注释调tree.print()这个函数！
因为这个接口调用的ToGraph()方法是有bug的，在B+树规模较大时会进入异常。
原因如下：
ToGraph()接口以递归方式遍历B+树，fetch子节点页后没有unpin, 导致b+树规模较大时，会让内存池全pin，这个函数会进入异常，如果在打印函数处陷入异常，建议注释掉打印函数再调试。

if you2022-05-30 00:41

请问一下有些函数比如MoveHalfTo()在leaf_page和internal_page中的参数定义并不一致，那么在3.3实现索引的时候该怎么使用模板呢？是需要自己改动参数使得在leaf_page和internal_page中一致吗？

YingChengJun2022-05-30 10:34

使用if进行判断是leaf还是internal

Naive2023-05-29 10:51IP 属地浙江

我发现框架有个地方好像是有错误，在CompareKeys函数中，如果两个key值不相等，在返回前会将Field中的字符串指针进行delete，但是这个指针实际指向的是row中的field，在row的析构函数中会再次进行delete从而导致double free的问题。

Winegee丶2024-05-13 12:38IP 属地浙江

在IsEmpty的判断中，buffer_pool_manager会fetch root_page但是没有Unpin，在Isempty函数中需要手动给root_page Unpin一下

LaaMa2024-05-23 23:42IP 属地浙江

void BPlusTree::UpdateRootPageId(int insert_record)这个方法需要调用bool HeaderPage::InsertRecord(const std::string &name, const page_id_t root_id)和bool HeaderPage::UpdateRecord(const std::string &name, const page_id_t root_id)，我一个BPlusTree怎么获得索引的name参数呢?

LaaMa2024-05-25 11:08IP 属地浙江

已解决：注释里的所有header page都应改为index roots page