实验六 MiniSQL的实现（模块三）

实验目的：

1、学习使用C++语言进行MiniSQL的实现。

2、了解SQL的实现，熟悉B+树索引的实现。

3、学习定义、插入数据的操作，提升解决问题的能力。

实验平台：

1、操作系统： WSL：Ubuntu

实验内容和要求：

1、实现B+树（索引）以及迭代器。

2、检验B+树（索引）以及迭代器功能的准确性。

B+树的实现和检验：

1、体系结构

B+树的构成以`page`e为基本单元，其数据都保存在`page`中，根据节点数据形式的不同，分为`Internal Page`和`Leaf Page`，可以通过类型转换将`page`转化为`Internal Page`或`Leaf Page`。

在操作过程中，B+树通过不断地`Fetchpage`从`buffer pool manager`处获取`Page`，同时`UnpinPage`并标记为脏以更新磁盘处的B+树数据。

2、具体实现

通过以下方式新建一个B+树，同时考虑到原本磁盘处无数据以及已经有数据从磁盘中读取出来的情况。

```
INDEX_TEMPLATE_ARGUMENTS
BPLUSTREE_TYPE::BPlusTree(index_id_t index_id,
BufferPoolManager *buffer_pool_manager, const KeyComparator &comparator,
int leaf_max_size, int internal_max_size):
  index_id_(index_id),
  buffer_pool_manager_(buffer_pool_manager),
  comparator_(comparator),
  leaf_max_size_(leaf_max_size),
```

```
  internal_max_size_(internal_max_size) {
  auto root_page = reinterpret_cast<IndexRootsPage *>(buffer_pool_manager-
>FetchPage(INDEX_ROOTS_PAGE_ID));

  if (!root_page->GetRootId(index_id, &this->root_page_id_)) {
    this->root_page_id_ = INVALID_PAGE_ID;
  }
  buffer_pool_manager->UnpinPage(INDEX_ROOTS_PAGE_ID, true);
  buffer_pool_manager_->UnpinPage(root_page_id_, true);
}
```

同时，以下函数能够向上对接索引的接口函数，保证整个索引体系的可扩展性

```
  // Returns true if this B+ tree has no keys and values.
  bool IsEmpty() const;

  // Insert a key-value pair into this B+ tree.
  bool Insert(const KeyType &key, const ValueType &value, Transaction
*transaction = nullptr);

  // Remove a key and its value from this B+ tree.
  void Remove(const KeyType &key, Transaction *transaction = nullptr);

  // return the value associated with a given key
  bool GetValue(const KeyType &key, std::vector<ValueType> &result, Transaction
*transaction = nullptr);
```

还有如以下函数等，负责B+树结构体系的维护和调整，单个函数的具体功能较为简单，但是函数数量很多，在此不多赘述。

```
// Split and Merge utility methods
  void MoveHalfTo(BPlusTreeLeafPage *recipient);

  void MoveAllTo(BPlusTreeLeafPage *recipient);

  void MoveFirstToEndOf(BPlusTreeLeafPage *recipient);

  void MoveLastToFrontOf(BPlusTreeLeafPage *recipient);
```

3、B+树正确性检验、

检验代码通过将一个`int`向量随机排序作为`value`，原顺序作为`key`，将其插入B+树中，再将部分节点移除，通过原数组进行比较和检验。同时，测试代码还会通过`Check`检验是否每一个`Page`都在B+树操作结束后被正常的`Unpin`了。

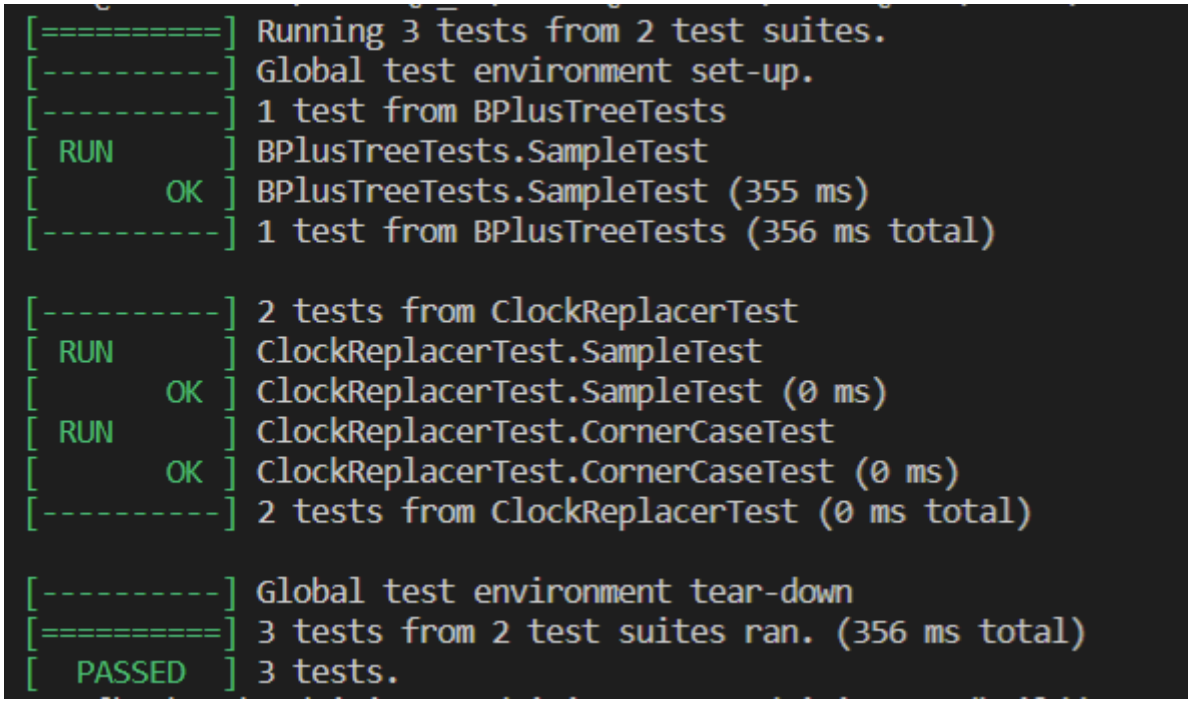
通过`b_plus_tree_test.cpp`中的代码对程序的准确性进行检验：

```
[==========] Running 3 tests from 2 test suites.
[----------] Global test environment set-up.
[----------] 1 test from BPlusTreeTests
[ RUN      ] BPlusTreeTests.SampleTest
[       OK ] BPlusTreeTests.SampleTest (100 ms)
[----------] 1 test from BPlusTreeTests (100 ms total)

[----------] 2 tests from ClockReplacerTest
[ RUN      ] ClockReplacerTest.SampleTest
[       OK ] ClockReplacerTest.SampleTest (0 ms)
[ RUN      ] ClockReplacerTest.CornerCaseTest
[       OK ] ClockReplacerTest.CornerCaseTest (0 ms)
[----------] 2 tests from ClockReplacerTest (0 ms total)

[----------] Global test environment tear-down
[==========] 3 tests from 2 test suites ran. (100 ms total)
[  PASSED  ] 3 tests.
```

执行测试，发现测试结果正确。

增大数据量进行测试，检验B+树的稳定性：

```
TEST(BPlusTreeTests, SampleTest) {
  // Init engine
  DBStorageEngine engine(db_name);
  BasicComparator<int> comparator;
  BPlusTree<int, int, BasicComparator<int>> tree(0, engine.bpm
  TreeFileManagers mgr("tree_");
  // Prepare data
  const int n = 100000;
  vector<int> keys;
  vector<int> values;
  vector<int> delete_seq;
  map<int, int> kv_map;
  ASSERT_TRUE(tree.Check());
```

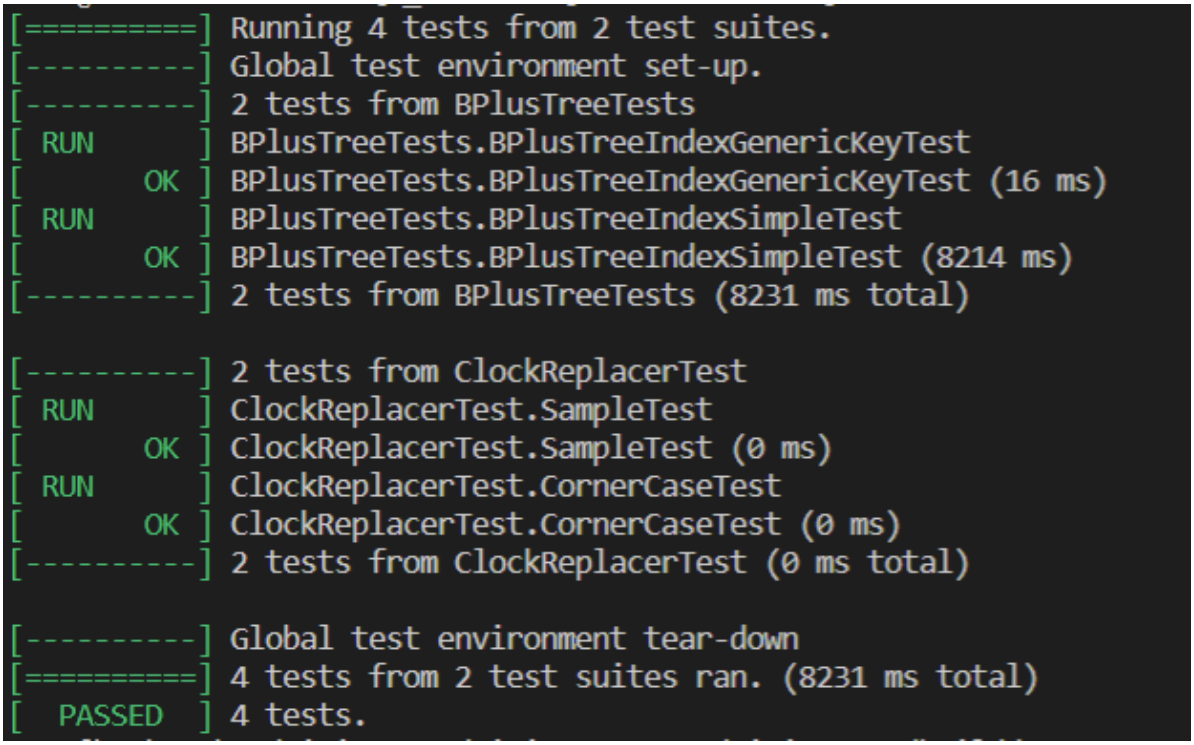


发现结果也是正确的。

4、B+树索引的正确性检验

B+树索引的准确性检验和B+树自身的检验相近，不同的是将插入的元素改为了`RowId`和`Row`。通过`Comparator`对`Key`也就是`Row`进行比较，检验增加索引接口后的程序准确性。

运行测试程序，可以看到B+树索引也是准确的。



迭代器的实现和检验：

1、迭代器的结构

迭代器的存在是为了实现对于所有保存的数据进行一次线性的遍历，在B+树的体系结构中，实现的迭代器能够获取B+树索引的头节点以及对于单个迭代器节点自增的操作，从而为上层程序获取单个元组的信息提供可能。

2、迭代器的实现

这部分的代码较为简单，只需要实现自增和等价比较的几个算符即可，其自身的序号通过`index_`这个私有变量进行描述，自增时需要考虑跨`Page`以及到达末尾的情况。

```
INDEX_TEMPLATE_ARGUMENTS const MappingType &INDEXITERATOR_TYPE::operator*()
{
  // ASSERT(false, "Not implemented yet.");
  return leaf_page_->GetItem(index_);
}

INDEX_TEMPLATE_ARGUMENTS INDEXITERATOR_TYPE &INDEXITERATOR_TYPE::operator++()
{
  if (index_ == leaf_page_->GetSize() - 1
      && leaf_page_->GetNextPageId() != INVALID_PAGE_ID)
  {
    Page* next_page = buffer_pool_manager_->FetchPage(leaf_page_-
>GetNextPageId());
    buffer_pool_manager_->UnpinPage(page_->GetPageId(), false);
    page_ = next_page;
    leaf_page_ = reinterpret_cast<LeafPage *>(page_->GetData());
    index_ = 0;
  }
  else
  {
    index_++;
  }

  return *this;
}

INDEX_TEMPLATE_ARGUMENTS
bool INDEXITERATOR_TYPE::operator==(const IndexIterator &itr) const
{
  if(itr.index_ == this->index_
    && itr.leaf_page_->GetPageId() == this->leaf_page_->GetPageId())
  {
    return true;
  }
  return false;
}

INDEX_TEMPLATE_ARGUMENTS
bool INDEXITERATOR_TYPE::operator!=(const IndexIterator &itr) const
{
  if(itr.index_ == this->index_
    && itr.leaf_page_->GetPageId() == this->leaf_page_->GetPageId())
  {
    return false;
  }
  return true;
```

```
  // return false;
}
```

2、迭代器的检验

运行测试代码，可以看到运行样例全都通过。

```
[==========] Running 3 tests from 2 test suites.
[----------] Global test environment set-up.
[----------] 1 test from BPlusTreeTests
[ RUN      ] BPlusTreeTests.IndexIteratorTest
[       OK ] BPlusTreeTests.IndexIteratorTest (21 ms)
[----------] 1 test from BPlusTreeTests (21 ms total)

[----------] 2 tests from ClockReplacerTest
[ RUN      ] ClockReplacerTest.SampleTest
[       OK ] ClockReplacerTest.SampleTest (0 ms)
[ RUN      ] ClockReplacerTest.CornerCaseTest
[       OK ] ClockReplacerTest.CornerCaseTest (0 ms)
[----------] 2 tests from ClockReplacerTest (0 ms total)

[----------] Global test environment tear-down
[==========] 3 tests from 2 test suites ran. (21 ms total)
[  PASSED  ] 3 tests.
```

模块建设性意见：

建议能够只给出索引的接口，同时提供以另一种数据结构（如哈希等）实现的现成的索引作为样例，让同学们体会如何具体地实现索引。首先B+树索引的难度比较大，但由于后续模块的需要，如果不实现就无法进行后续模块的编写，这就导致一个小组的进度被单个同学拖累，如果有一个现成的索引，一个同学写B+树的时候其他同学可以进行后续模块的开发，这可能能够更好地平衡小组内部的工作量；同时目前的模块架构较为复杂，注释也含糊不清，让同学们自己在体会索引的基本架构后自行建立B+树的架构可能会更好一点，或者将原本的框架作为一个参考，同学们可以自行修改B+树类的结构。