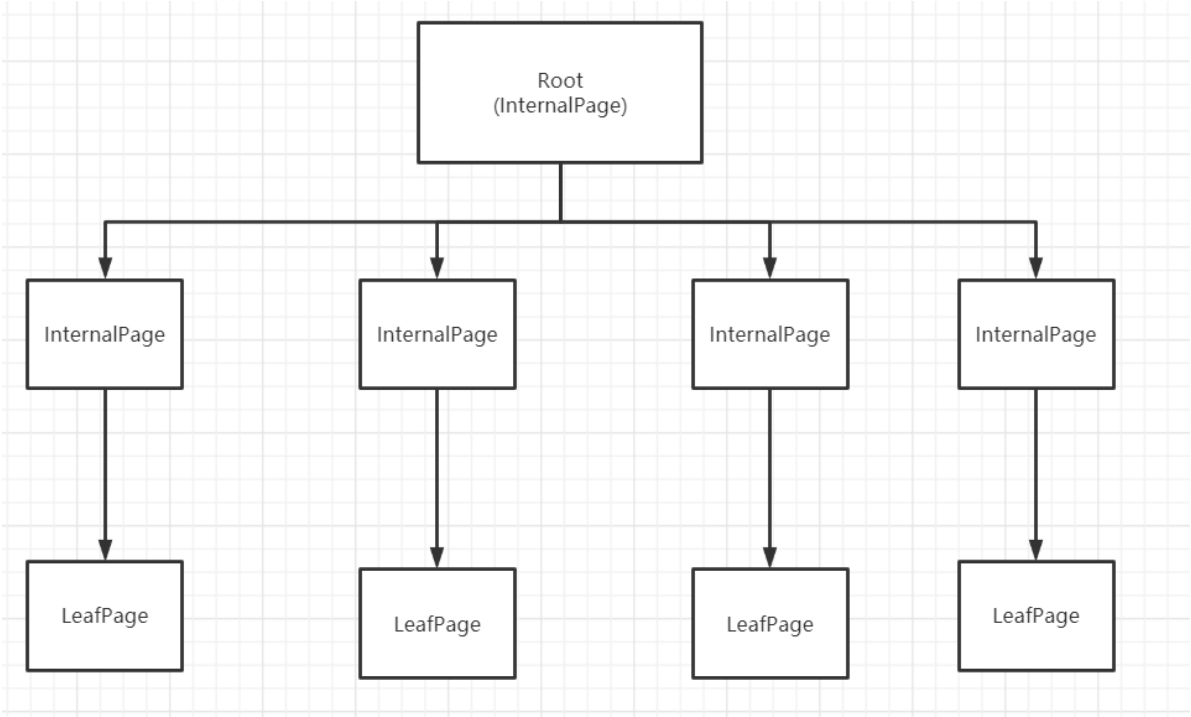
INDEX MANAGER

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。

由于通过堆表遍历来进行查找记录效率过于低下，因此本实验实现了一个基于磁盘的B+树动态索引结构。

2. 实验总体框架



可以看到需要实现的B+树的结构，由于内部节点和叶节点的结构不同，因此在实现B+树需要分别实现`BPlusTreePage`和`BPlusInternalPage`，最后根据B+树的操作分别调用数据页中提供的接口函数。

2. B+树数据页

2.1 BPlusTreePage

由于`LeafPage`和`InternalPage`均是继承自`BplusTreePage`，因此需要先实现BPlusTreePage

- 数据结构

```
• page_type_: 标记数据页是中间结点还是叶子结点；
• lsn_: 数据页的日志序列号，目前不会用到，如果之后感兴趣做Crash Recovery相关的内容需要用到；
• 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。
```

2.2 BPlusTreeLeafPage

- 数据结构

```
/*
 * Leaf page format (keys are stored in order):
 *  ----------------------------------------------------------------------
--
 * | HEADER | KEY(1) + RID(1) | KEY(2) + RID(2) | ... | KEY(n) + RID(n)
 *  ----------------------------------------------------------------------
--
 *
 *  Header format (size in byte, 24 bytes in total):
 *  ----------------------------------------------------------------------
-
 * | PageType (4) | CurrentSize (4) | MaxSize (4) | ParentPageId (4) |
 *  ----------------------------------------------------------------------
-
 *  -------------------------------
 * | PageId (4) | NextPageId (4)
 *  -------------------------------
 */
```

- 函数

  下面我将对关键函数进行详细说明

Insert相关函数

- `BPlusTreeLeafPage::int Insert(const KeyType &key, const ValueType &value, const KeyComparator &comparator);`

  设计思路：

  1. 在相应的LeafPage插入记录，更新LeafPage的`Size`
  2. 插入完成后，返回`Size`，方便上层判断是否需要进行分裂。

  代码：

```
INDEX_TEMPLATE_ARGUMENTS
int B_PLUS_TREE_LEAF_PAGE_TYPE::Insert(const KeyType &key, const
ValueType &value, const KeyComparator &comparator) {
  int index = -1;
  if (this->GetSize() + 1 <= this->GetMaxSize()) {
    for (int i = 0; i < this->GetSize(); i++) {
      // Find the Position Which the Key will insert into.
      if (comparator(key, this->array_[i].first) < 0) {
        index = i;
        break;
      }
    }
    // NewKey &Value should insert into end of the NewNode
    if (index == -1) {
      index = this->GetSize();
```

```
      if (this->GetSize() < this->GetMaxSize()) this->array_[index] = {key, value};

      this->IncreaseSize(1);
  /*    for (int i = 0; i < this->GetSize(); i++) {
        std::cout << this->array_[i].first  << endl;
      }*/
      return this->GetSize();
    }
    // NewKey& Value should insert into middle of the NewNode
    else {
      // Copy From the End to First
      for (int i = this->GetSize(); i > index; i--) {
        this->array_[i] = this->array_[i - 1];
      }
      // Insert the NewKey&Value into Right Position
      if (this->GetSize() < this->GetMaxSize()) this->array_[index] = {key, value};
      this->IncreaseSize(1);
      return this->GetSize();
    }
  } else {
    return this->GetSize()+1;
  }

}
```

- `void MoveHalfTo(BPlusTreeLeafPage *recipient, BufferPoolManager *bufferpoolManager);`

  设计思路：

  在上述`Insert`操作之后，节点个数大于临界值，需要进行`Split`

  - 将此页一半的Key移动到接收页中
  - 更新相邻节点的页号

  代码：

```
INDEX_TEMPLATE_ARGUMENTS
void B_PLUS_TREE_LEAF_PAGE_TYPE::MoveHalfTo(BPlusTreeLeafPage *recipient,BufferPoolManager* bufferpoolManager) {

    int DeleteSize = (this->GetMinSize());
    recipient->CopyNFrom(&this->array_[(this->GetSize()-this->GetMinSize())], DeleteSize);  // Recipient Increase Size Here
    if (this->GetNextPageId() != INVALID_PAGE_ID) {
      recipient->SetNextPageId(this->GetNextPageId());
    }
    this->SetNextPageId(recipient->GetPageId());

    //Decrease Size
    this->IncreaseSize(-recipient->GetSize());


}
```

- `void CopyNFrom(MappingType *items, int size);`

  设计思路：

  此函数是在MoveHalfTo函数的子函数，主要完成，复制的功能。

  - 将items[0]--items[size]的Element复制到此页中

  代码:

```
INDEX_TEMPLATE_ARGUMENTS
void B_PLUS_TREE_LEAF_PAGE_TYPE::CopyNFrom(MappingType *items, int size) {


  // Copy Entries into This Page
  for (int i = 0; i < size; i++) {

      this->array_[i].first = items[i].first;
      this->array_[i].second = items[i].second;

  }
  // Increment the Size
  this->IncreaseSize(size);

}
```

Remove相关函数

- `int RemoveAndDeleteRecord(const KeyType &key, const KeyComparator &comparator);`

  设计思路：

  - 首先在LeafPage进行查找相应的想要删除的`Key`
  - 如果存在，进行删除，不存在，返回
  - 返回值说明：返回`Size`以方便上层，进行判断，是否需要从兄弟节点`Merge`或者`Redistribute`。

  代码：

```
INDEX_TEMPLATE_ARGUMENTS
int B_PLUS_TREE_LEAF_PAGE_TYPE::RemoveAndDeleteRecord(const KeyType &key, const KeyComparator &comparator) {
    bool state = false;

    for (int i = 0; i < this->GetSize(); i++) {
      if (comparator(key, this->array_[i].first) == 0) {
        state = true;
        int index = i;
        for (int i = index + 1; i < this->GetSize(); i++) {
          this->array_[i - 1].first = this->array_[i].first;
          this->array_[i - 1].second = this->array_[i].second;
        }
```

```
        this->IncreaseSize(-1);
        break;
      }
    }

    if (state == false) {
      return this->GetSize();
    }
    return this->GetSize();
}
```

- `void MoveAllTo(BPlusTreeLeafPage *recipient);`

  设计思路:

  上述 `RemoveAndDelete` 函数可能出现Merge，此函数就是用来Merge功能的函数

  - 将所有的key，Value全部复制到接受页中

  代码:

```
INDEX_TEMPLATE_ARGUMENTS
void B_PLUS_TREE_LEAF_PAGE_TYPE::MoveAllTo(BPlusTreeLeafPage *recipient
) {
    //Copy the Instance into the recipient Page
    for (int i = 0; i < this->GetSize(); i++) {
      recipient->CopyLastFrom(this->array_[i]);  // Add Size Here
    }

    this->IncreaseSize(-this->GetSize());                    //
Decrease Size of this Page
    //Set the Next Page id.
    recipient->SetNextPageId(this->GetNextPageId());
}
```

- `void MoveFirstToEndOf(BPlusTreeLeafPage *recipient);`

  设计思路:

  上述 `RemoveAndDelete` 函数可能出现 `Redistribute`，此函数就是用来 `Redistribute` 功能的函数

  - 将此页的 第一个 `key & value` 复制到 `recipient`

  代码:

```
INDEX_TEMPLATE_ARGUMENTS
void B_PLUS_TREE_LEAF_PAGE_TYPE::MoveFirstToEndOf(BPlusTreeLeafPage
*recipient) {
    recipient->CopyLastFrom(this->array_[0]);//Add Size Here
    //Remove the First Key
    for (int i = 0; i < this->GetSize()-1; i++) {
      this->array_[i] = this->array_[i + 1];
    }
    this->IncreaseSize(-1);
}

```

- `void MoveLastToFrontOf(BPlusTreeLeafPage *recipient);`

  设计思路:

  上述 `RemoveAndDelete` 函数可能出现 `Redistribute`，此函数就是用来 `Redistribute` 功能的函数

  - 将此页的 末尾的 `key & value` 复制到 `recipient`

  代码:

```
INDEX_TEMPLATE_ARGUMENTS
void B_PLUS_TREE_LEAF_PAGE_TYPE::MoveLastToFrontOf(BPlusTreeLeafPage
*recipient) {

  recipient->CopyFirstFrom(this->array_[this->GetSize() - 1]);
  this->IncreaseSize(-1);
}

```

2.3 BPlusTreeInternalPage

- 数据结构

```
/*
 * Internal page format (keys are stored in increasing order):
 *  ---------------------------------------------------------------------
-----
 * | HEADER | KEY(1)+PAGE_ID(1) | KEY(2)+PAGE_ID(2) | ... |
KEY(n)+PAGE_ID(n) |
 *  ---------------------------------------------------------------------
-----
 */
```

- 函数说明

Insert相关函数

- `void PopulateNewRoot(const ValueType &old_value, const KeyType &new_key, const ValueType &new_value);`

  设计思路：

  此函数应用在如果Insert一直进行分裂，并且一直分裂到根节点，需要产生新的Root。

  - 将新产生的键值对插入RootNode

  代码：

```
INDEX_TEMPLATE_ARGUMENTS
void B_PLUS_TREE_INTERNAL_PAGE_TYPE::PopulateNewRoot(const ValueType &old_value, const KeyType &new_key,
                                                     const ValueType &new_value)
{

    //old_value means Old root-page id
  this->array_[0].second = old_value;
    SetKeyAt(1, new_key);
  this->array_[1].second = new_value;
  //Increase Size by two
  this->IncreaseSize(2);

}
```

- `int InsertNodeAfter(const ValueType &old_value, const KeyType &new_key, const ValueType &new_value);`

  设计思路：

  - 将键值对插入到`OldValue`之后。
  - 返回Size，来判断是否需要进行`Split`

  代码：

```
INDEX_TEMPLATE_ARGUMENTS
int B_PLUS_TREE_INTERNAL_PAGE_TYPE::InsertNodeAfter(const ValueType &old_value, const KeyType &new_key,
                                                    const ValueType &new_value) {
  //Get Index Of the old_value
  int index = ValueIndex(old_value);
  //If the index do not overflow
  if (index + 1 < this->GetMaxSize()) {
    for (int i = this->GetSize(); i > index+1; i--) {
      if (i - 1 > 0) {
        this->array_[i] = this->array_[i - 1];
      } else {
        this->array_[i].second = this->array_[i - 1].second;
      }
    }
    this->SetKeyAt(index + 1, new_key);
```

```
    array_[index + 1].second = new_value;
    this->IncreaseSize(1);
  } else {
    throw "B_PLUS_TREE_INTERNAL_PAGE_TYPE::InsertNodeAfter-
OverFlow" ;
    return this->GetSize() + 1;
  }

  return this->GetSize();
}
```

- `void MoveHalfTo(BPlusTreeInternalPage *recipient, BufferPoolManager *buffer_pool_manager);`

  设计思路：

  - 将一半的键值对复制到接受页

  代码：

```
INDEX_TEMPLATE_ARGUMENTS
void
B_PLUS_TREE_INTERNAL_PAGE_TYPE::MoveHalfTo(BPlusTreeInternalPage
*recipient,
                                           BufferPoolManager
*buffer_pool_manager) {

  // Copy into the recipient Page
  recipient->CopyNFrom(&this->array_[this->GetSize()-this-
>GetMinSize()],this->GetMinSize(),buffer_pool_manager);
  // Remove From the Current Page
  this->IncreaseSize(-recipient->GetSize());

  //buffer_pool_manager->UnpinPage(this->GetPageId(),true);


}
```

Remove 相关函数

- `void Remove(int index);`

  设计思路：

  - 将index对应的键值对进行删除
  - 更新Size

  代码：

```
INDEX_TEMPLATE_ARGUMENTS
void B_PLUS_TREE_INTERNAL_PAGE_TYPE::Remove(int index) {

    //Delete the first Child
    if (index == 0&&this->GetSize()>=2) {
        //Remove from the first Value
```

```
      for (int i = 0; i < this->GetSize(); i++) {
          if (i == 0) {
          this->array_[i].second = this->array_[i + 1].second;
          } else {
            this->array_[i] = this->array_[i + 1];
          }
      }
      this->IncreaseSize(-1);
    }

    if (this->GetSize()==0) {
    std::cerr << "Can not Remove" << endl;
    } else {
      //Move Forward the array_
       if (this->GetSize() == 2) {
          // it means it only has one child and one pair key& value
          // it delete the last element, so we need to delete the
last pair,
          // it does not need to remove

          this->IncreaseSize(-1);
          return;

      } else {
        for (int i = index + 1; i < this->GetSize(); i++) {
          this->array_[i - 1].first = this->array_[i].first;
          this->array_[i - 1].second = this->array_[i].second;
        }

      }

    }
    this->IncreaseSize(-1);
}
```

- `void MoveAllTo(BPlusTreeInternalPage *recipient, const KeyType &middle_key, BufferPoolManager *buffer_pool_manager);`

设计思路：

- 将此页的所有key&Value和Middle_key全部复制到接受页中

代码：

```
INDEX_TEMPLATE_ARGUMENTS
void B_PLUS_TREE_INTERNAL_PAGE_TYPE::MoveAllTo(BPlusTreeInternalPage
*recipient, const KeyType &middle_key,
                                               BufferPoolManager
*buffer_pool_manager) {
  MappingType NewPair = {middle_key, this->array_[0].second};
  for (int i = 0; i < this->GetSize(); i++) {
    if (i != 0)
      recipient->CopyLastFrom(this->array_[i], buffer_pool_manager);
    else
      recipient->CopyLastFrom(NewPair, buffer_pool_manager);
  }
  //Remove the Key from this Page
  this->IncreaseSize(-this->GetSize());
}
```

- `void MoveFirstToEndOf(BPlusTreeInternalPage *recipient, const KeyType &middle_key,BufferPoolManager *buffer_pool_manager);`

设计思路:

上述`RemoveAndDelete`函数可能出现`Redistribute`，此函数就是用来`Redistribute`功能的函数

- 将此页的 第一个 `key & value` 复制到 `recipient`

代码:

```
INDEX_TEMPLATE_ARGUMENTS
void B_PLUS_TREE_INTERNAL_PAGE_TYPE::MoveFirstToEndOf(BPlusTreeInternalPage
*recipient, const KeyType &middle_key,
                                                      BufferPoolManager
*buffer_pool_manager) {
  MappingType NewPair = {middle_key, this->array_[0].second};
  recipient->CopyLastFrom(NewPair,buffer_pool_manager);//Add Size Here
  //Remove the First Key
  // If we need to Update the Middle_key in the Parent node
  // We just Get from the Index ==0
  for (int i = 0; i < this->GetSize()-1; i++) {
    this->array_[i] = this->array_[i + 1];
  }
  this->IncreaseSize(-1);


}
```

- `void MoveLastToFrontOf(BPlusTreeInternalPage *recipient, const KeyType &middle_key,`
  `BufferPoolManager *buffer_pool_manager);`

设计思路：

上述`RemoveAndDelete`函数可能出现`Redistribute`，此函数就是用来`Redistribute`功能的函数

- 将此页的 末尾的 `key & value` 复制到 `recipient`

代码：

```
INDEX_TEMPLATE_ARGUMENTS
void B_PLUS_TREE_INTERNAL_PAGE_TYPE::MoveLastToFrontOf(BPlusTreeInternalPage *recipient, const KeyType &middle_key,
                                                       BufferPoolManager *buffer_pool_manager) {

  MappingType NewPair = {middle_key, this->array_[this->GetSize()-1].second};
  recipient->CopyFirstFrom(NewPair, buffer_pool_manager);//AddSize Here
  //If we need to update the Parent's middle_key just get from the index==GetSize
  this->IncreaseSize(-1);

}
```

- `void ResetParent(const page_id_t &old_node, const page_id_t &new_node, BufferPoolManager *buffer_pool_manager_);`

设计思路：

将此页的 末尾的 `key & value` 复制到 `recipient`

代码：

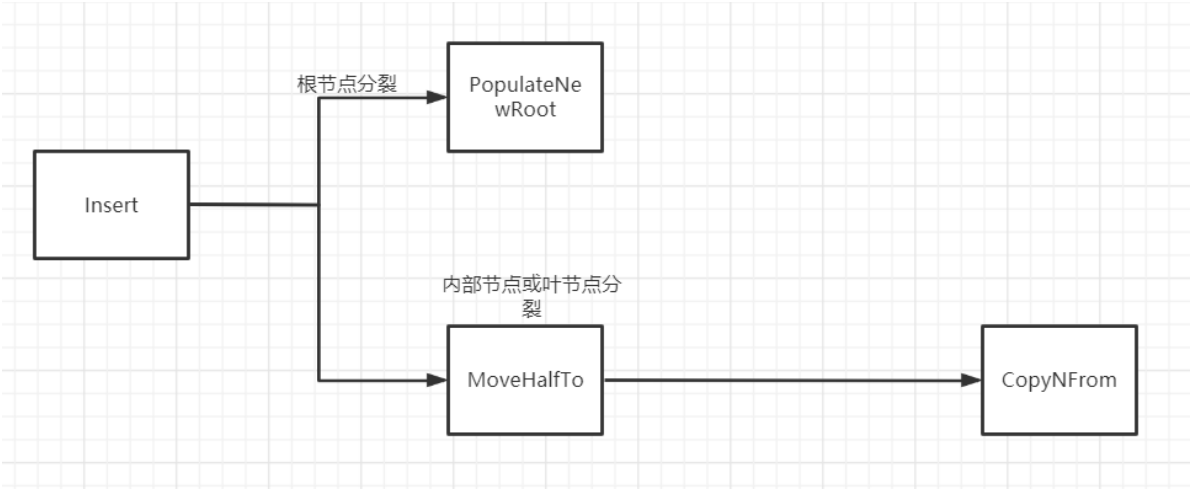
```
INDEX_TEMPLATE_ARGUMENTS
void B_PLUS_TREE_INTERNAL_PAGE_TYPE::ResetParent(const page_id_t &old_node, const page_id_t &new_node,BufferPoolManager* buffer_pool_manager_) {
  // Read the old_page and new_page
  BPlusTreePage *old_page = reinterpret_cast<BPlusTreePage *>(buffer_pool_manager_->FetchPage(old_node));
  BPlusTreePage *new_page = reinterpret_cast<BPlusTreePage *>(buffer_pool_manager_->FetchPage(new_node));
  old_page->SetParentPageId(this->GetPageId());
  new_page->SetParentPageId(this->GetPageId());
  buffer_pool_manager_->UnpinPage(old_node, true);
  buffer_pool_manager_->UnpinPage(new_node, true);
}
```

3. B+树索引

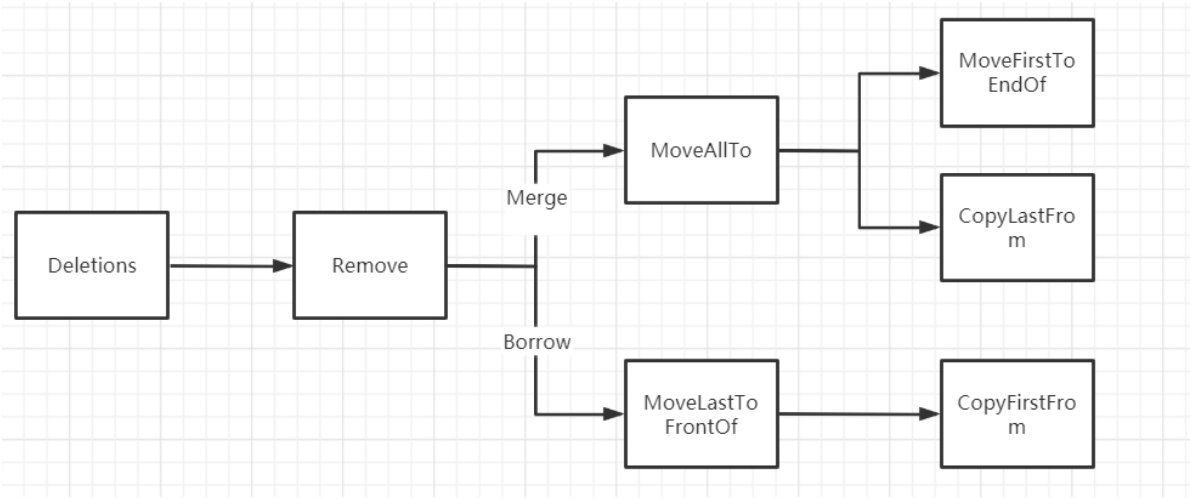
3.1 总体框架

由于B+树索引主要就是`Insert`和`Remove`两个操作，下面图片将说明这两个操作和上述两个数据页中的函数之间的关系。

1.Insert



2.Remove



3.2 函数说明

- Insert

  伪代码：

```
step1:若为空树，创建一个叶子结点，然后将记录插入其中，此时这个叶子结点也是根结点，插入操作结束。

step2:
    针对叶子类型结点：根据key值找到叶子结点，
    if(插入后节点的个数<=n-1)
        插入结束
    else
        //split
        divide into (1,2 ......n/2) and ((n/2)+1,.....n-1)
        将(n/2)+1节点推到父节点
        当前节点指向父节点
        //Insert Into Parent
        //索引类型节点
```

```
        while(当前的个数>=n-1)
        //split
        分成(m-1)/2,m-(m-1)/2两组
        将当前节点指向父节点
```

代码:

```
INDEX_TEMPLATE_ARGUMENTS
bool BPLUSTREE_TYPE::Insert(const KeyType &key, const ValueType &value,
Transaction *transaction) {
    bool state = false;
    //Empty Tree
    if (this->IsEmpty()) {
      //std::cout << "Insert::StartNewTree" << endl;
      this->StartNewTree(key, value);
      state = true;
    } else {
      //std::cout << "Insert::InsertIntoLeaf" << endl;
      state=InsertIntoLeaf(key, value, transaction);
    }
    return state;
}
```

Split相关函数

- `INDEX_TEMPLATE_ARGUMENTS`
  `template<typename N>`
  `N *BPLUSTREE_TYPE::Split(N *node)`

代码:

```
INDEX_TEMPLATE_ARGUMENTS
template<typename N>
N *BPLUSTREE_TYPE::Split(N *node) {
     page_id_t NewId = INVALID_PAGE_ID;
     auto *page = buffer_pool_manager_->NewPage(NewId);
     if (page != nullptr) {
       N *NewNode = reinterpret_cast<N *>(page->GetData());
       //Init
       NewNode->Init(NewId, INVALID_PAGE_ID,node->GetMaxSize());


       node->MoveHalfTo(NewNode,buffer_pool_manager_);

       //Set the Sibbling point to the Same Parent
       NewNode->SetParentPageId(node->GetParentPageId());
       buffer_pool_manager_->UnpinPage(NewId, true);

       return NewNode;
     } else {
       buffer_pool_manager_->UnpinPage(NewId, false);
       std::cerr << "BPLUSTREE_TYPE::InsertIntoParent---Can not Allocate
Memory AnyMore" << endl;
       return nullptr;
     }
```

```
24
25 }
```

InsertIntoParent相关函数

- `INDEX_TEMPLATE_ARGUMENTS void BPLUSTREE_TYPE::InsertIntoParent(BPlusTreePage *old_node, const KeyType &key, BPlusTreePage *new_node,Transaction *transaction)`

  代码:

```
1  INDEX_TEMPLATE_ARGUMENTS
2  void BPLUSTREE_TYPE::InsertIntoParent(BPlusTreePage *old_node,
   const KeyType &key, BPlusTreePage *new_node,
3      Transaction *transaction) {
4    KeyType NewKey = key;
5    while (1) {
6        //old_node is root
7      if (old_node->GetPageId() == this->root_page_id_) {
8        page_id_t NewId = INVALID_PAGE_ID;
9        auto *page = buffer_pool_manager_->NewPage(NewId);
10       if (page != nullptr) {
11         auto *node = reinterpret_cast<InternalPage *>(page-
   >GetData());
12         node->Init(NewId, INVALID_PAGE_ID,this-
   >internal_max_size_);
13         //Generate the New Root
14         node->PopulateNewRoot(old_node->GetPageId(), NewKey,
   new_node->GetPageId());
15         //Reset Parent
16         node->ResetParent(old_node->GetPageId(), new_node-
   >GetPageId(),buffer_pool_manager_);
17
18         //Update the New Root
19         this->root_page_id_ = NewId;
20         this->UpdateRootPageId(false);
21         buffer_pool_manager_->UnpinPage(NewId, true);
22         //std::cout << "BPLUSTREE_TYPE::InsertIntoParent------286"
   << endl;
23         break;
24       }
25       else {
26         buffer_pool_manager_->UnpinPage(NewId, false);
27         std::cerr << "BPLUSTREE_TYPE::InsertIntoParent---Can not
   Allocate Memory AnyMore" << endl;
28       }
29     }
30     else
31     {
32         //Percolate Up
33         //Get Parent Page From the BufferPoolManager
34         page_id_t Id = old_node->GetParentPageId();
35         auto *page = buffer_pool_manager_->FetchPage(Id);
36         if (page != nullptr)
37         {
```

```
            auto * ParentNode = reinterpret_cast<InternalPage *>(page->GetData());
            //Insert NewKey into the Parent Node
            /*for (int i = 0; i < ParentNode->GetSize(); i++) cout << ParentNode->array_[i].first << " ";
            cout << endl;*/
            int size = ParentNode->InsertNodeAfter(old_node->GetPageId(), NewKey, new_node->GetPageId());
            /*for (int i = 0; i < ParentNode->GetSize(); i++) cout << ParentNode->array_[i].first << " ";
            cout << endl;*/
            // if the Parent Node is not OverFlow
            if (size < this->internal_max_size_) {
            buffer_pool_manager_->UnpinPage(Id, true);
              //std::cout << "BPLUSTREE_TYPE::InsertIntoParent-----311" << endl;
            break;
            }
            else {
            //If the Parent Node is Full
            KeyType middle_key = ParentNode->KeyAt(ParentNode->GetSize()-ParentNode->GetMinSize());
            InternalPage *Sibbling = this->Split(ParentNode);
            old_node = ParentNode;
            NewKey = middle_key;
            new_node = Sibbling;
            buffer_pool_manager_->UnpinPage(Id, true);
            //std::cout << "BPLUSTREE_TYPE::InsertIntoParent-----321" << endl;
            continue;
            }
        }
        else
        {
          buffer_pool_manager_->UnpinPage(Id, false);
          //std::cerr << "BPLUSTREE_TYPE::InsertIntoParent---2Cannot Allocate Memory AnyMore" << endl;
        }
    }
  }

}
```

- Remove

  伪代码：

```
step1:删除叶子结点中对应的key。删除后若结点的key的个数大于等于Math.ceil(m-1)/2 - 1，删除操作结束,否则执行第2步。

step2:
    if(兄弟结点key有富余（大于Math.ceil(m-1)/2 - 1）),
        向兄弟结点借一个记录
```

```
        同时用借到的key替换父结（指当前结点和兄弟结点共同的父结点）点中的key，删除结束。
    else
        if(兄弟结点中没有富余的key)
        则当前结点和兄弟结点合并成一个新的叶子结点，并删除父结点中的key
        （父结点中的这个key两边的孩子指针就变成了一个指针，正好指向这个新的叶子结点）
        将当前结点指向父结点（必为索引结点）
        else
            while(索引结点的key的个数<Math.ceil(m-1)/2 - 1)
                if(兄弟结点有富余)
                    父结点key下移，兄弟结点key上移
                    break
                else
                    当前结点和兄弟结点及父结点下移key合并成一个新的结点。
                    将当前结点指向父结点
```

相关函数:

- `INDEX_TEMPLATE_ARGUMENTS`
  `void BPLUSTREE_TYPE::Remove(const KeyType &key, Transaction *transaction)`

  代码:

```
INDEX_TEMPLATE_ARGUMENTS
void BPLUSTREE_TYPE::Remove(const KeyType &key, Transaction *transaction) {
    if (this->IsEmpty()) return;
    auto page = FindLeafPage(key, false);
    auto node = reinterpret_cast<BPlusTreePage*> (page->GetData());
    page_id_t FirstLeafId = page->GetPageId();



    if (node->IsRootPage()) {
      LeafPage* Root = reinterpret_cast<LeafPage *> (node);
      int size=Root->RemoveAndDeleteRecord(key, comparator_);
      if (size == 0) {
        if (this->AdjustRoot(node)) {
          buffer_pool_manager_->UnpinPage(node->GetPageId(),true);

          bool state=buffer_pool_manager_->DeletePage(node->GetPageId());
          if (state == false) {
            throw "fuck";
          }
        }
      }
    }

    else {
      auto Leaf = reinterpret_cast<LeafPage *>(page->GetData());
      int size = Leaf->RemoveAndDeleteRecord(key, comparator_);
      //After the Deletetion the Leaf size >= MinSize
      if (size >= (Leaf->GetMinSize())) {
        buffer_pool_manager_->UnpinPage(FirstLeafId, true);
```

```
30
31          return;
32        }
33
34        else {
35          //After the Deletion the Leaf size < MinSize
36            //Merge or Borrow
37            ///
38
39          if (this->CoalesceOrRedistribute(&Leaf, transaction) ==
false) {
40            buffer_pool_manager_->UnpinPage(FirstLeafId, true);
41
42            return;
43          }
44          else {
45              // We need to Adjust Percolate Up
46
47              auto ParentPage = buffer_pool_manager_->FetchPage(Leaf-
>GetParentPageId());
48              page_id_t ParentId = ParentPage->GetPageId();
49              InternalPage *node = reinterpret_cast<InternalPage*>
(ParentPage->GetData());
50              buffer_pool_manager_->UnpinPage(Leaf->GetPageId(),
true);
51              bool state=buffer_pool_manager_->DeletePage(Leaf-
>GetPageId());
52              if (state == false) {
53                throw "fuck";
54              }
55              InternalPage *Parent = nullptr;
56
57
58
59
60              while (1) {
61                if (this->CoalesceOrRedistribute(&node, transaction)
== false) {
62                  buffer_pool_manager_->UnpinPage(FirstLeafId, true);
63                  buffer_pool_manager_->UnpinPage(ParentId,true);
64
65                  break;
66                }
67                else {
68
69                  Parent = reinterpret_cast<InternalPage *>
(buffer_pool_manager_->FetchPage(node->GetParentPageId())-
>GetData());
70                  buffer_pool_manager_->UnpinPage(ParentId, true);
71                  buffer_pool_manager_->UnpinPage(node->GetPageId(),
true);
72                  bool state = buffer_pool_manager_->DeletePage(node-
>GetPageId());
73                  ParentId = Parent->GetPageId();
74                  if (state == false) {
```

```
                    throw "fuck";
                  }
                  node = Parent;
                }
              }
            }

          }
        }
        buffer_pool_manager_->UnpinPage(node->GetPageId(), true);


}
```

- `INDEX_TEMPLATE_ARGUMENTS`
  `template<typename N>`
  `bool BPLUSTREE_TYPE::CoalesceOrRedistribute(N **node, Transaction *transaction)`
- 代码:

```
INDEX_TEMPLATE_ARGUMENTS
template<typename N>
bool BPLUSTREE_TYPE::CoalesceOrRedistribute(N **node, Transaction
*transaction)
{

  bool state = false;
  if (this->IsEmpty() == true) return false;
  InternalPage *ParentNode = nullptr;
  // Root Page
  if ((* node)->IsRootPage()) {
    if (( * node)->GetSize() == 1) {
      state=this->AdjustRoot(*node);
      if (state) {

        page_id_t PageId = (*node)->GetPageId();
        buffer_pool_manager_->UnpinPage(PageId, true);

        bool state=buffer_pool_manager_->DeletePage(PageId);
        if (state == false) {
          throw "fuck";
        }
      }
    }
    return false;
  }

      //Get the Parent Page
    page_id_t ParentPageId = (*node)->GetParentPageId();
    auto Page=buffer_pool_manager_->FetchPage(ParentPageId);
```

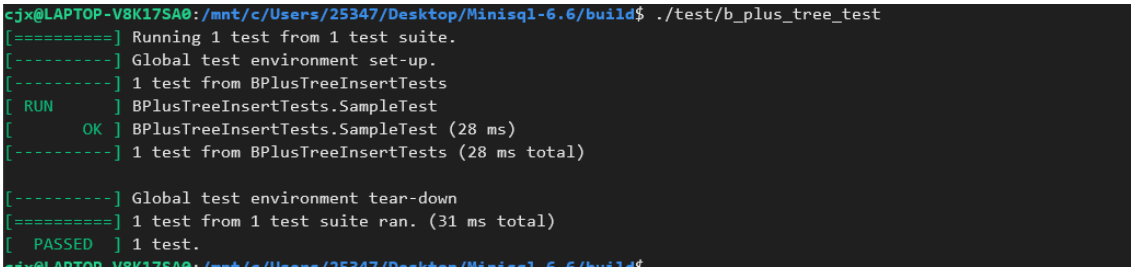
```
    ParentNode = reinterpret_cast<InternalPage *>(Page->GetData());
    //Get the child node index in the ParentNode
    int index = ParentNode->ValueIndex((*node)->GetPageId());

    //If the Node is LeafNode
    if (( * node)->IsLeafPage()) {
       // If the Node is the right Most Element
      if (index == ParentNode->GetSize() - 1) {
        // node is last child of the Parent Node
        int SibblingIndex = index - 1;
        page_id_t Sibbling_Page_Id = ParentNode->ValueAt(SibblingIndex);
        // Merge or Borrow from the Left Page
        auto SibblingPage = buffer_pool_manager_->FetchPage(Sibbling_Page_Id);
        LeafPage *SibblingNode = reinterpret_cast<LeafPage *>(SibblingPage->GetData());
        LeafPage *Node = reinterpret_cast<LeafPage *>(*node);
        //case 1----Borrow From the Sibbiling Node ----Test---Not Check
        if (SibblingNode->GetSize() + Node->GetSize() > Node->GetMaxSize()) {
          // Move the Sibbling Last Pair into the This Node
          this->Redistribute(SibblingNode, Node, 1);
          int index = ParentNode->ValueIndex(Node->GetPageId());
          ParentNode->SetKeyAt(index, Node->KeyAt(0));
          state = false;
        }
        //case 2---- Merge with the Sibbling Node  ----Test---OK
        else {
          state = this->Coalesce(&SibblingNode, &Node, &ParentNode, index, transaction,true);
        }
        buffer_pool_manager_->UnpinPage(Sibbling_Page_Id, true);
      }
      // if the node is not right most Node
      else {
        page_id_t Sibbling_Page_Id = ParentNode->ValueAt(index + 1);
        auto SibblingPage = buffer_pool_manager_->FetchPage(Sibbling_Page_Id);
        LeafPage *SibblingNode = reinterpret_cast<LeafPage *>(SibblingPage->GetData());
        LeafPage *Node = reinterpret_cast<LeafPage *>(*node);
        //case 1--- Borrow From the Sibbling Node------ Test---- Not Check
        if (SibblingNode->GetSize() + Node->GetSize() > Node->GetMaxSize()) {
          // Move the Redistribute First Pair into the End of this node
          this->Redistribute(SibblingNode, Node, 0);
          // Adjust the Key in the Parent Node
          int index = ParentNode->ValueIndex(SibblingNode->GetPageId());
```

```
71            ParentNode->SetKeyAt(index, SibblingNode->KeyAt(0));
72            state = false;
73          }
74          // case 2----Merge With the Sibbling Node -----Test---Ok
75          else {
76            state = this->Coalesce(&Node, &SibblingNode,
&ParentNode, index + 1, transaction,false);
77            *node = (reinterpret_cast<N*> (SibblingNode));
78          }
79          buffer_pool_manager_->UnpinPage(Sibbling_Page_Id, true);
80        }
81
82        buffer_pool_manager_->UnpinPage(ParentPageId, true);
83
84
85     } else {
86     // If the node is Internal Node
87          // node is >= MinSize
88       if (( * node)->GetSize() >= (*node)->GetMinSize()) {
89       buffer_pool_manager_->UnpinPage(ParentPageId,true);
90
91              return false;
92       }
93       // If the Node is the right Most Element
94       if (index == ParentNode->GetSize() - 1) {
95         // node is last child of the Parent Node
96         int SibblingIndex = index - 1;
97         page_id_t Sibbling_Page_Id = ParentNode-
>ValueAt(SibblingIndex);
98         // Merge or Borrow from the Left Page
99         auto SibblingPage = buffer_pool_manager_-
>FetchPage(Sibbling_Page_Id);
100
101        InternalPage *SibblingNode = reinterpret_cast<InternalPage
*>(SibblingPage->GetData());
102        InternalPage *Node = reinterpret_cast<InternalPage *>
(*node);
103
104
105        // case 1----Borrow From the Sibbiling Node ----  Test Not
Check
106        if (SibblingNode->GetSize() + Node->GetSize() > Node-
>GetMaxSize()) {
107          // Move the Sibbling Last Pair into the This Node
108          this->Redistribute(SibblingNode, Node, 1);
109          int index = ParentNode->ValueIndex(Node->GetPageId());
110          ParentNode->SetKeyAt(index, Node->KeyAt(SibblingNode-
>GetSize()));
111          state = false;
112        }
113        // case 2---- Merge with the Sibbling Node  ----Test Not
Check
114        else {
115          state = this->Coalesce(&SibblingNode, &Node, &ParentNode,
index, transaction,true);
```

```
116         }
117         buffer_pool_manager_->UnpinPage(Sibbling_Page_Id, true);
118       }
119       // if the node is not right most Node
120       else
121       {
122         page_id_t Sibbling_Page_Id = ParentNode->ValueAt(index +
    1);
123         auto SibblingPage = buffer_pool_manager_-
    >FetchPage(Sibbling_Page_Id);
124 
125         InternalPage *SibblingNode = reinterpret_cast<InternalPage
    *>(SibblingPage->GetData());
126         InternalPage *Node = reinterpret_cast<InternalPage *>
    (*node);
127 
128         // case 1--- Borrow From the Sibbling Node------ Test-Not
    Check
129         if (SibblingNode->GetSize() + Node->GetSize() > Node-
    >GetMaxSize()) {
130           // Move the Redistribute First Pair into the End of this
    node
131           this->Redistribute(SibblingNode, Node, 0);
132           // Adjust the Key in the Parent Node
133           int index = ParentNode->ValueIndex(SibblingNode-
    >GetPageId());
134           ParentNode->SetKeyAt(index, SibblingNode->KeyAt(0));
135           state = false;
136         }
137         // case 2----Merge With the Sibbling Node -----Test-Not
    Check
138         else {
139           state = this->Coalesce(&Node, &SibblingNode, &ParentNode,
    index + 1, transaction,false);
140           *node = (reinterpret_cast<N *>(SibblingNode));
141         }
142        //
143         buffer_pool_manager_->UnpinPage(Sibbling_Page_Id, true);
144       }
145 
146       buffer_pool_manager_->UnpinPage(ParentPageId, true);
147 
148 
149 
150 
151     }
152 
153 
154 
155   return state;
156 
157 }
```

3.3 测试结果

- 测试BplusTree的`Insert`和`Remove`



4.BPlusTreeIndexIterator

- 成员定义

  为了方便索引查询，提供迭代器，方便上层进行调用。

```
private:
  // add your own private member variables here
    BufferPoolManager *bufferPoolManager;
        page_id_t CurrPageId;
        int CurrLocation;
        // if we Read the Tuple from the Leaf Page , We need to Copy the Item
        MappingType *Item = nullptr;
```

- 基本操作
  - `const MappingType &operator*();`

    实现逻辑:

    取出叶节点的键值对

    代码:

```
INDEX_TEMPLATE_ARGUMENTS const MappingType &INDEXITERATOR_TYPE::operator*() {

  auto page = bufferPoolManager->FetchPage(this->CurrPageId);
  auto node = reinterpret_cast<LeafPage *>(page);
  // Free the preview Item
  if (this->Item != nullptr) delete this->Item;
  this->Item = new MappingType;
  MappingType *Pair = new (this->Item) MappingType(node->GetItem(this->CurrLocation));
  bufferPoolManager->UnpinPage(this->CurrPageId,false);
  return *Pair;

}
```

- `IndexIterator &operator++();`

  实现逻辑：

  完成`++`的Overload

```
INDEX_TEMPLATE_ARGUMENTS INDEXITERATOR_TYPE
&INDEXITERATOR_TYPE::operator++() {
   auto page = bufferPoolManager->FetchPage(this->CurrPageId);
   auto node = reinterpret_cast<LeafPage *>(page);
   int Capacity = node->GetSize();
  // just Point to Next Pair in this Page
   if (this->CurrLocation + 1 < Capacity) {
     bufferPoolManager->UnpinPage(this->CurrPageId, false);
     CurrLocation++;
   } else {
     page_id_t NextPage = node->GetNextPageId();

     bufferPoolManager->UnpinPage(this->CurrPageId,false);
     // It means NextPage is the Last Page
     if (NextPage == INVALID_PAGE_ID) {
       this->CurrPageId = INVALID_PAGE_ID;
       this->CurrLocation = 0;
     } else {
       // Update Next Page
       this->CurrPageId = NextPage;
       // Update Next Position
       this->CurrLocation = 0;
     }

   }
   return *this;
}
```

- `bool operator==(const IndexIterator &itr) const;`

  实现逻辑：

  判断当前的迭代器，是否和待比较的迭代器相等

  代码：

```
INDEX_TEMPLATE_ARGUMENTS
bool INDEXITERATOR_TYPE::operator==(const IndexIterator &itr) const {
  if (this->CurrLocation == itr.CurrLocation && this->CurrPageId ==
itr.CurrPageId) {
    return true;
  } else
    return false;
}
```

- `bool operator!=(const IndexIterator &itr) const;`

  实现逻辑：

  判断当前的迭代器，是否和待比较的迭代器不相等

代码:

```
INDEX_TEMPLATE_ARGUMENTS
bool INDEXITERATOR_TYPE::operator!=(const IndexIterator &itr) const {

    return !(*this == itr);

}
```

- 测试BplusTree的`Iterator`

```
cjx@LAPTOP-V8K17SA0:/mnt/c/Users/25347/Desktop/Minisql-6.6/build$ ./test/b_plus_tree_index_test
[==========] Running 2 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 2 tests from BPlusTreeTests
[ RUN      ] BPlusTreeTests.BPlusTreeIndexGenericKeyTest
[       OK ] BPlusTreeTests.BPlusTreeIndexGenericKeyTest (9 ms)
[ RUN      ] BPlusTreeTests.BPlusTreeIndexSimpleTest
[       OK ] BPlusTreeTests.BPlusTreeIndexSimpleTest (652 ms)
[----------] 2 tests from BPlusTreeTests (662 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 1 test suite ran. (664 ms total)
[  PASSED  ] 2 tests.
```

3.5 总体测试

- 测试BPlusTreeIndex的`BplusTree`和`Iterator`

```
cjx@LAPTOP-V8K17SA0:/mnt/c/Users/25347/Desktop/Minisql-6.6/build$ ./test/b_plus_tree_index_test
[==========] Running 2 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 2 tests from BPlusTreeTests
[ RUN      ] BPlusTreeTests.BPlusTreeIndexGenericKeyTest
[       OK ] BPlusTreeTests.BPlusTreeIndexGenericKeyTest (8 ms)
[ RUN      ] BPlusTreeTests.BPlusTreeIndexSimpleTest
[       OK ] BPlusTreeTests.BPlusTreeIndexSimpleTest (696 ms)
[----------] 2 tests from BPlusTreeTests (706 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 1 test suite ran. (707 ms total)
[  PASSED  ] 2 tests.
cjx@LAPTOP-V8K17SA0:/mnt/c/Users/25347/Desktop/Minisql-6.6/build$
```

5. 模块相关代码

- `src/include/storage/page/b_plus_tree_page.h`
- `src/page/b_plus_tree_page.cpp`
- `src/include/storage/page/b_plus_tree_internal_page.h`
- `src/storage/page/b_plus_tree_internal_page.cpp`
- `src/include/storage/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`