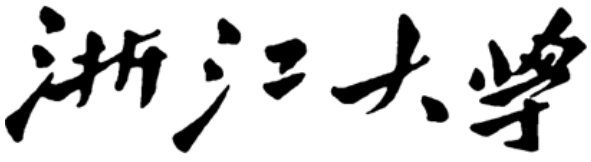


本科实验报告

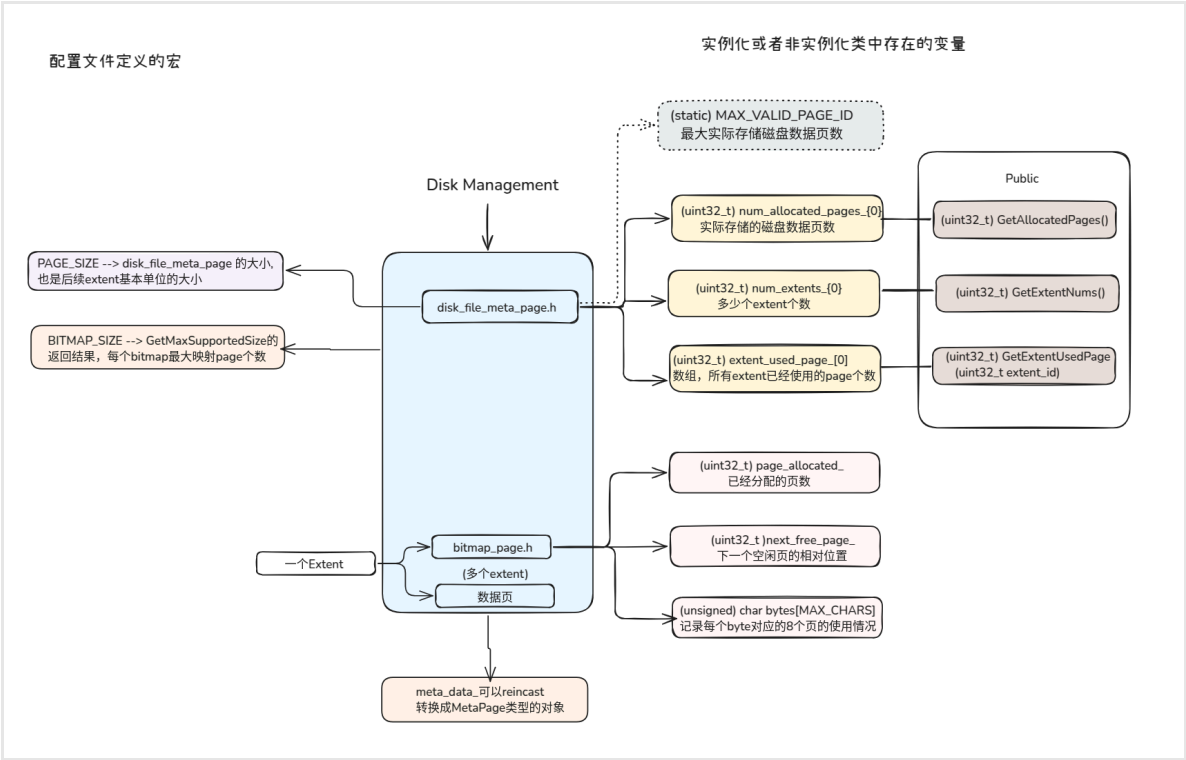
课程名称：	数据库系统
实验名称：	MiniSQL
姓　　名：	陈皓天
实验地点：	西教 506
指导教师：	苗晓烨
报告日期：	2025/04/19

第一章 模块 1

该模块实现了磁盘数据页管理、位图页、缓冲池管理和 LRU 替换策略，并加以必要的并发控制锁。其中，磁盘数据页管理和缓冲池管理的变量较多，逻辑较为复杂，我通过绘画思维导图的方式帮助自己完成代码。

1. 磁盘数据页管理

以 AllocatePage 为例，虽然这是个磁盘数据页管理类，但他一开始并不直接与磁盘相交互，而是先将 disk_file_meta_page 从内存中读取信息，然后根据需要创建特定索引的 Bitpage 类，然后从磁盘读入这个类，经过修改后再写回磁盘.我绘制的思维导图图如下：



可以看到，Disk_Management 直接与 disk_file_meta_page 和 bitmap_page 交互，来管理磁盘 page 的分配与释放; 在 Allocate 中，物理地址和逻辑地址的转换尤为重要，其中 locate_Bitmap 是我命名的来索引 extent 的变量，当确定某一个 locate_Bitmap 的值之后，就需要转化为物理地址，来传入 ReadPhysicalPage 函数，于是需要以下逻辑：

```
int locate_physics = 1 + (1 + BITMAP_SIZE) * locate_Bitmap;
```

这里考虑到了 实际存储数据页 和 meta_page 占用的页数，直接得到了 bitmap_page 的物理索引

2. 位图页

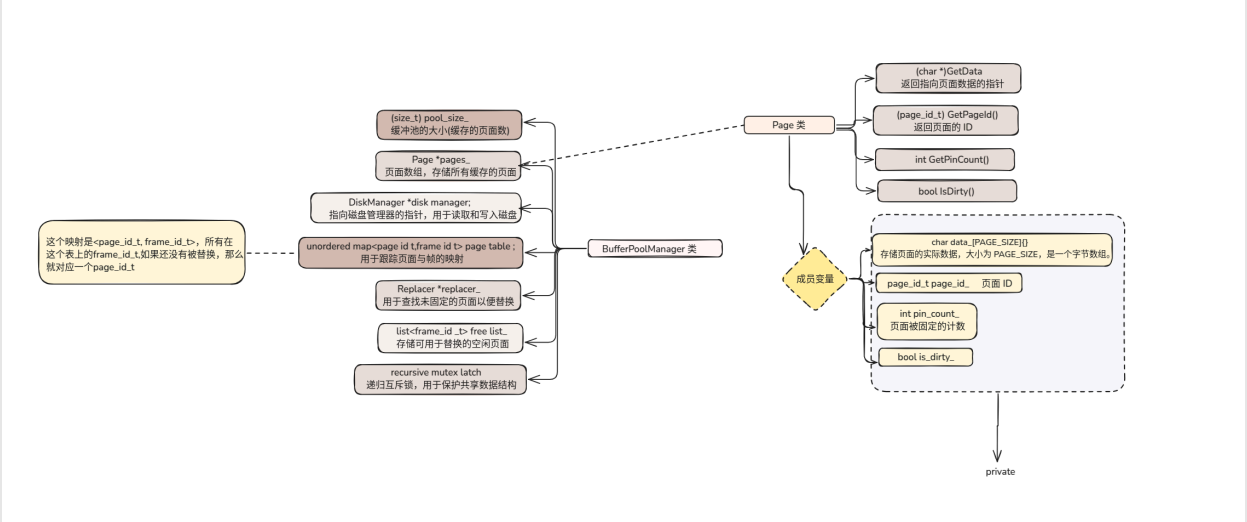
此处的代码较为简单，在上面的磁盘数据管理的逻辑中，一个 extent 包含着一个位图页和多个数据存储页，位图页的目的是为了管理和快速索引数据存储页。

值得注意的是，这个类中的 next_free_page_ 变量记录了下一个空闲的页数索引（从 0 开始）,有助于查找空闲页的速度.

bytes 数组的每一个位的长度是 8 个 bits,通过 byte_indexc 和 bit_index 的位操作可以得出该页是否空闲的信息

3. 缓冲池管理

此模块变量较多，我同样通过 mind map 的方式帮助自己记忆



这个缓冲池管理类和刚刚的磁盘数据页管理类的区别在于，缓冲池管理类是包含真的数据的。实际上其数据存储在一个 Page* 数组中，Page* 数组的索引是 frame_id;相当于这个 Page 数组包含着磁盘中的某几页数据，通过 page_table_ 将磁盘页和 frame 页相对应类起来；在必要的时候可以通过 LRU 策略替换其中的某些磁盘页。

4. LRU 替换策略

LRUReplacer 类提供了保存可替换 frame 的容器，用于在缓冲池 FetchPage 方法和 NewPage 方法时，当找不到空闲页的时候选择 last recently used 页进行替换，并且当其调用 Pin 方法的时候将对应 frame 移出容器；

这里一共有两个容器来保存可替换 frame,分别是 list<frame_id_t> lru_queue,用作记录顺序，以及 unordered_set<frame_id_t> lru_set;,用作快速查找；两者结合，lru 方法的效率可以得到大幅提升.

5. 正确性测试

分别对上述四个模块进行三个测试，均通过。

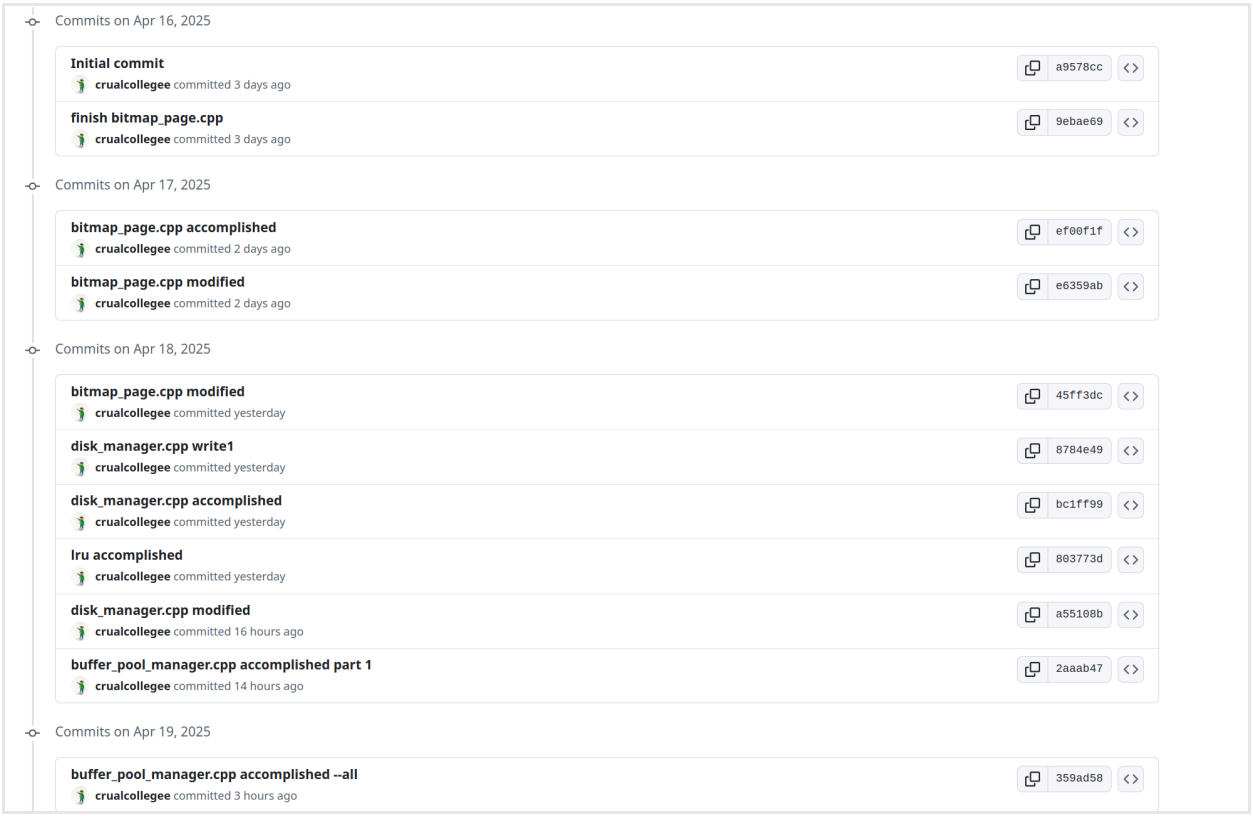
```
(base) linux@ubuntu:~/db/minisql/build/test$ ls
buffer_pool_manager_test  cmake_install.cmake  disk_manager_test  libminisql_test_main.so  Makefile
CMakeFiles                CTestTestfile.cmake  disk_test.db       lru_replacer_test
(base) linux@ubuntu:~/db/minisql/build/test$ ./disk_manager_test
[==========] Running 2 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 2 tests from DiskManagerTest
[ RUN      ] DiskManagerTest.BitMapPageTest
[       OK ] DiskManagerTest.BitMapPageTest (4 ms)
[ RUN      ] DiskManagerTest.FreePageAllocationTest
[       OK ] DiskManagerTest.FreePageAllocationTest (267 ms)
[----------] 2 tests from DiskManagerTest (272 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 1 test suite ran. (272 ms total)
[  PASSED  ] 2 tests.
(base) linux@ubuntu:~/db/minisql/build/test$ ./lru_replacer_test
[==========] Running 1 test from 1 test suite.
[----------] Global test environment set-up.
[----------] 1 test from LRUReplacerTest
[ RUN      ] LRUReplacerTest.SampleTest
[       OK ] LRUReplacerTest.SampleTest (0 ms)
[----------] 1 test from LRUReplacerTest (0 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test suite ran. (0 ms total)
[  PASSED  ] 1 test.
(base) linux@ubuntu:~/db/minisql/build/test$ ./buffer_pool_manager_test
[==========] Running 1 test from 1 test suite.
[----------] Global test environment set-up.
[----------] 1 test from BufferPoolManagerTest
[ RUN      ] BufferPoolManagerTest.BinaryDataTest
[       OK ] BufferPoolManagerTest.BinaryDataTest (1 ms)
[----------] 1 test from BufferPoolManagerTest (1 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test suite ran. (1 ms total)
[  PASSED  ] 1 test.
(base) linux@ubuntu:~/db/minisql/build/test$
```

以及我的 git 推送记录



第二章　模块 3

IndexManagement 模块的代码量较大，我花费了大约一周时间完成了这个代码，并且通过了测试；模块 2 是交给我的队友完成的，模块 2 与模块 3 的交集大概就是在 row.cpp 和 genericKey 上了；我首先读懂了这两个模块的代码，然后从底层代码开始看起。

1. 基本 page 类

首先是完成了 b_plus_tree_page.cpp 代码，这个模块较为简单，是叶子节点和内部节点的父类，就是完成了一些两者共有的方法，比较简单，不加以赘述.

接着我完成了 b_plus_tree_internal_page.cpp 代码；首先要明确的是，一个 internal 类在一般情况下分为多个 genericKey 类和 page_id_t 类，这两个类合在一起为一个 pair；但其中的地一个 pair 一般只包含 page_id_t 类，这是 b+树特有的性质。

这个模块给我带来印象最深的就是多个 internalpage 类的交互，通过外界参数 recipient 的传入，我们可以将 this 的部分 pair 通过类的 public 接口转移到 recipient 去，实现了传递。

```
// 加分隔层和调用remove
void InternalPage::MoveFirstToEndOf(InternalPage *recipient, GenericKey *middle_key,
BufferPoolManager *buffer_pool_manager) {
  recipient->CopyLastFrom(middle_key, ValueAt(0), buffer_pool_manager);
  Remove(0);
}
```

但做到这里为止，我发现 internalpage 类有些接口没有被自己使用到，而且接口繁多，我猜测是有其他类将来会引用这些接口

然后便是 leafpage 类的完成，这个类和 internalpage 类就较为重合了，我较为轻松的完成了；我特别注意到了删除某个 pair 这个方法的时候，我没有使用 PairCopy 进行批量“位移”，因为担心内存重叠的危险，我按照类似冒泡排序中的逐个位移的方法，一次只 PairCopy 1 个 pair.

```
int LeafPage::RemoveAndDeleteRecord(const GenericKey *key, const KeyManager &KM) {
  RowId value_t;
  bool isSuccess = Lookup(key,value_t,KM);
  if (!isSuccess){
    return GetSize();
  }
  else{
    //Paircopy多个可能有内存重叠的危险
    int index_t = KeyIndex(key,KM);
    char* char_index = reinterpret_cast<char*>(KeyAt(index_t));
    for (int i=index_t;i<GetSize()-1;i++){
      PairCopy(char_index,char_index+pair_size,1);
      char_index+=pair_size;
    }
    SetSize(GetSize()-1);
    return GetSize();
  }
}
```

特别需要注意的是，这个 pair 包含的是一个 genericKey 和 rowID 的指针，也就是说包含类一个指向实际数据（rowID）的指针。而刚刚的 internalpage 的 pair 的第二个元素就是 一个 pageID（不是指针）,可以调用 FetchPage() 来获取相应 pageID 的 page.

（在以前（ADS 课程）的实现中内部节点往往是通过指针指向子节点的，而这里有 buffer_pool_manager.cpp 充当了管理页面的作用）

2. b_plus_tree 树类

然后就是工程量最大的 b_plus_tree.cpp 的编写，也就是这个地方才会和 buffer_pool_manager.cpp 打交道，尤其是其 UnpinPage() 函数，要被经常调用；一开始我写的时候经常忘了这茬,但后来对读取页面和保存页面的操作熟悉之后就得心应手了。

这个类其实就是一棵树，但类不包含树的所有信息，而是在需要的时候从内存池中 fetch 下来。其实我有个疑惑，就是这个类对 buffer_pool_manager_->FetchPage(INDEX_ROOTS_PAGE_ID)) 这个记录 树的 index 和 root_page_id 的页面经常调用和修改，但从来没看见被使用过。我只能合理猜测在后面的模块中会被用到吧。(树的 index 在这个模块里也没被使用过)

其他的模块对我印象较深的就是删除的向上递归过程，以及 CoalesceOrRedistribute()、Coalesce() 和 Redistribute()的相互调用，显得比较有逻辑感。这几个函数也调用了 c (internal/leaf) page 类大量的 movehalfto()、moveallto()等之前看起来比较奇怪的函数。

```
bool BPlusTree::Coalesce(InternalPage *&neighbor_node, InternalPage *&node, InternalPage *&parent,
                         Txn *transaction) {
  page_id_t node_index = node->GetPageId();
  node->MoveAllTo(neighbor_node,parent->KeyAt(index),buffer_pool_manager_);

  parent->Remove(parent->ValueIndex(node_index));
  if (parent->GetSize()<parent->GetMinSize()){
    return CoalesceOrRedistribute<BPlusTree::InternalPage>(parent,transaction);
  }
  return true;
}
```

3. index_iterator 类

这个类比较简单，实现了一个简单的迭代器 iterator,具体到某一页的某一个 pair.另外，这个 iterator 也在 b_plus_tree 中被使用到了,在此不加以赘述

```
std::pair<GenericKey *, RowId> IndexIterator::operator*() {
  return page->GetItem(item_index);
}
```

4. 正确性测试

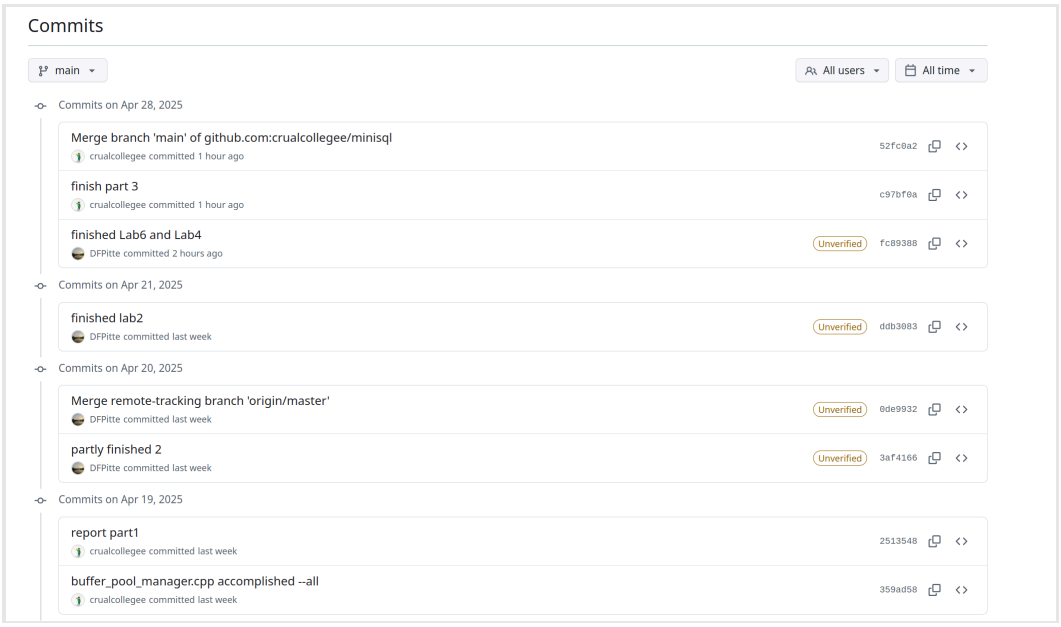
```
(base) linux@ubuntu:~/db/minisql/build/test$ ./b_plus_tree_test
[==========] Running 1 test from 1 test suite.
[----------] Global test environment set-up.
[----------] 1 test from BPlusTreeTests
[ RUN      ] BPlusTreeTests.SampleTest
[       OK ] BPlusTreeTests.SampleTest (403 ms)
[----------] 1 test from BPlusTreeTests (403 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test suite ran. (403 ms total)
[  PASSED  ] 1 test.
(base) linux@ubuntu:~/db/minisql/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 (0 ms)
[ RUN      ] BPlusTreeTests.BPlusTreeIndexSimpleTest
[       OK ] BPlusTreeTests.BPlusTreeIndexSimpleTest (80 ms)
[----------] 2 tests from BPlusTreeTests (80 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 1 test suite ran. (80 ms total)
[  PASSED  ] 2 tests.
(base) linux@ubuntu:~/db/minisql/build/test$ ./index_iterator_test
[==========] Running 1 test from 1 test suite.
[----------] Global test environment set-up.
[----------] 1 test from BPlusTreeTests
[ RUN      ] BPlusTreeTests.IndexIteratorTest
[       OK ] BPlusTreeTests.IndexIteratorTest (86 ms)
[----------] 1 test from BPlusTreeTests (86 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test suite ran. (86 ms total)
[  PASSED  ] 1 test.
```

5. git 推送记录

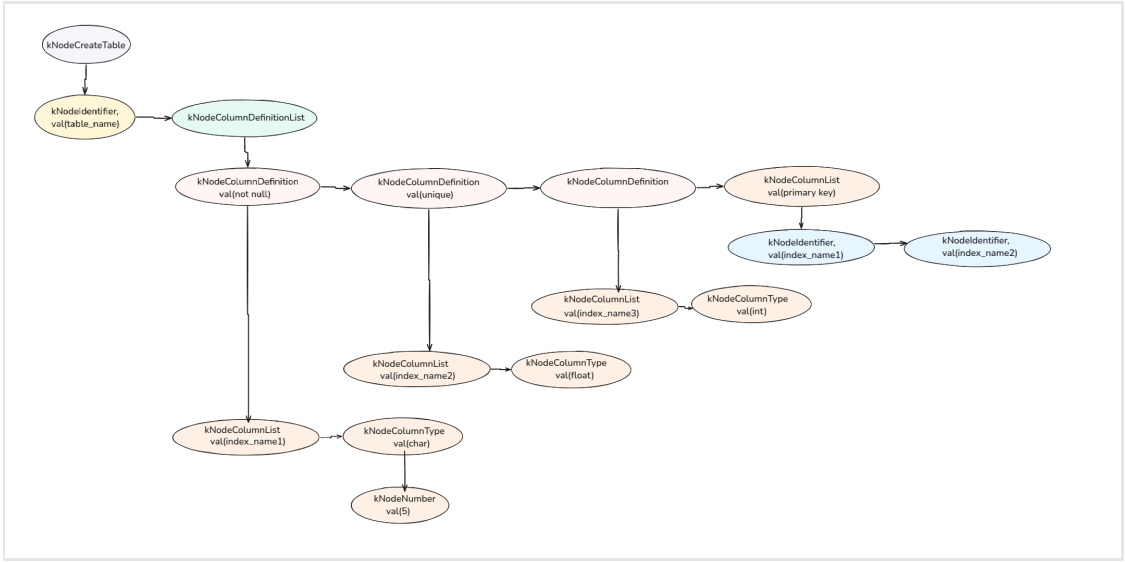


第三章　模块 5

本模块分为两个部分，planner 和 executor；planner 负责将用户输入的 sql 语句转化成规范化的，类封装的 ast 树；而 executor 则是根据 ast 树执行相应的操作。这个模块运用到了大量的 catalog.cpp 函数，以及 syntax_tree.h 和 derr.h 内容，所以我花费了一定时间去了解了外部的一些定义。

实际要完成的工作就是 execute_engine.cpp 的部分内容，其开头的 executor 调用和 select()方法实现已经完成了,我们实际上只需要完成部分 executor 的实现即可。

对于 CreateTable 方法，其具体的实现其实不难，但要理解它 ast 树的逻辑有点困难；为此，我运行了 bin 目录下的 main，在同目录下观察 syntax.txt 画出了以下逻辑图



根据逻辑图，我能较为轻松的完成这个函数的设计，但要注意以下逻辑：1. 当 primary key 只有一个元素的时候，那么这个元素一定是 unique key 2. primary key 和 unique key 都要调用 CreateIndex()函数来形成 b+树索引.

剩余几个函数都较为简单，ExecuteDropTable()方法用到了 catalog.cpp 中 GetTableIndexes() DropIndex() DropTable()多个函数，需要花费一定时间理解

2.正确性测试

executorTest 并不能测试到我要实现的代码，即对表的一些无关乎数据的操作；如果要测试这部分，需要我运行/bin 目录下的 main 进行手动命令行交互：

```
(base) linux@ubuntu:~/db/minisql/build/bin$ ./main
minisql > create database mydb1;
[INFO] Sql syntax parse ok!
minisql > show databases;
[INFO] Sql syntax parse ok!
+----------+
| Database |
+----------+
| mydb1    |
+----------+
minisql > drop database mydb1;
[INFO] Sql syntax parse ok!
minisql > show databases;
[INFO] Sql syntax parse ok!
Empty set (0.00 sec)
minisql > create database mydb2;
[INFO] Sql syntax parse ok!
minisql > use mydb2;
[INFO] Sql syntax parse ok!
Database changed
minisql > show tables;
[INFO] Sql syntax parse ok!
+-----------------+
| Tables_in_mydb2 |
+-----------------+
+-----------------+
minisql > create table orders(
  order_id   int,
  customer   char(30) unique,
  amount     float,
  primary key(order_id, amount)
);
[INFO] Sql syntax parse ok!
minisql > create table employees(
  emp_no     char(10),
  emp_age    int,
  emp_salary float unique,
  primary key(emp_no, emp_salary)
);
[INFO] Sql syntax parse ok!
minisql > show tables;
[INFO] Sql syntax parse ok!
+-----------------+
| Tables_in_mydb2 |
+-----------------+
| employees       |
| orders          |
+-----------------+
minisql > drop table employees;
[INFO] Sql syntax parse ok!
```

```
minisql > show tables;
[INFO] Sql syntax parse ok!
+-----------------+
| Tables_in_mydb2 |
+-----------------+
| orders          |
+-----------------+
minisql > create index idx_orders_customer on orders(customer, amount);
[INFO] Sql syntax parse ok!
minisql > show indexes;
[INFO] Sql syntax parse ok!
+-------------------------------+
| Index                         |
+-------------------------------+
| idx_orders_customer           |
| index_order_idamount_on_orders |
| index_customer_on_orders      |
+-------------------------------+
minisql > drop index idx_orders_customer;
[INFO] Sql syntax parse ok!
minisql > show indexes;
[INFO] Sql syntax parse ok!
+-------------------------------+
| Index                         |
+-------------------------------+
| index_order_idamount_on_orders |
| index_customer_on_orders      |
+-------------------------------+
minisql > insert into orders values(101, "alice", 123.45);
insert into orders values(202, "bob",   678.90);
insert into orders values(303, "carol", 50.25);[INFO] Sql syntax parse ok!
Query OK, 1 row affected(0.0010 sec).
minisql > [INFO] Sql syntax parse ok!
Query OK, 1 row affected(0.0000 sec).
minisql > select * from orders;
[INFO] Sql syntax parse ok!
Query OK, 1 row affected(0.0000 sec).
minisql > Minisql parse error at line 1, col 5, message: syntax error
syntax error
minisql > select * from orders;
[INFO] Sql syntax parse ok!
+----------+----------+------------+
| order_id | customer | amount     |
+----------+----------+------------+
| 101      | alice    | 123.449997 |
| 202      | bob      | 678.900024 |
| 303      | carol    | 50.250000  |
+----------+----------+------------+
3 row in set(0.0000 sec).
minisql > update orders set customer = "guest";
[INFO] Sql syntax parse ok!
Query OK, 3 row affected(0.0010 sec).
```

```
minisql > select * from orders;
[INFO] Sql syntax parse ok!
+----------+----------+------------+
| order_id | customer | amount     |
+----------+----------+------------+
| 101      | guest    | 123.449997 |
| 202      | guest    | 678.900024 |
| 303      | guest    | 50.250000  |
+----------+----------+------------+
3 row in set(0.0000 sec).
minisql > update orders
  set order_id = 999, customer = "vip"
  where amount = 123.45;
[INFO] Sql syntax parse ok!
Query OK, 1 row affected(0.0000 sec).
minisql > select * from orders;
[INFO] Sql syntax parse ok!
+----------+----------+------------+
| order_id | customer | amount     |
+----------+----------+------------+
| 999      | vip      | 123.449997 |
| 202      | guest    | 678.900024 |
| 303      | guest    | 50.250000  |
+----------+----------+------------+
3 row in set(0.0000 sec).
minisql >
select order_id, amount from orders;
[INFO] Sql syntax parse ok!
+----------+------------+
| order_id | amount     |
+----------+------------+
| 999      | 123.449997 |
| 202      | 678.900024 |
| 303      | 50.250000  |
+----------+------------+
3 row in set(0.0000 sec).
minisql > select * from orders where order_id = 202;
[INFO] Sql syntax parse ok!
+----------+----------+------------+
| order_id | customer | amount     |
+----------+----------+------------+
| 202      | guest    | 678.900024 |
+----------+----------+------------+
1 row in set(0.0000 sec).
minisql > select * from orders where order_id > 100 and order_id < 300;
[INFO] Sql syntax parse ok!
+----------+----------+------------+
| order_id | customer | amount     |
+----------+----------+------------+
| 202      | guest    | 678.900024 |
+----------+----------+------------+
1 row in set(0.0000 sec).
minisql > delete from orders where order_id = 999 and amount = 123.45;
[INFO] Sql syntax parse ok!
```

```
minisql > delete from orders where order_id = 999 and amount = 123.45;
[INFO] Sql syntax parse ok!
Error Encountered in Executor Execution: std::exception
Query OK, 0 row affected(0.0000 sec).
minisql > select * from orders;
[INFO] Sql syntax parse ok!
+----------+----------+------------+
| order_id | customer | amount     |
+----------+----------+------------+
| 202      | guest    | 678.900024 |
| 303      | guest    | 50.250000  |
+----------+----------+------------+
2 row in set(0.0000 sec).
minisql > quit;
[INFO] Sql syntax parse ok!
minisql > show tables;
[INFO] Sql syntax parse ok!
No database selected
minisql >
```

当然，executorTest 可以验证其他一些模块例如 catalog.cpp 不会出错：

```
(base) linux@ubuntu:~/db/minisql/build/test$ ./executor_test
[==========] Running 4 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 4 tests from ExecutorTest
[ RUN      ] ExecutorTest.SimpleSeqScanTest
[       OK ] ExecutorTest.SimpleSeqScanTest (112 ms)
[ RUN      ] ExecutorTest.SimpleDeleteTest
[       OK ] ExecutorTest.SimpleDeleteTest (111 ms)
[ RUN      ] ExecutorTest.SimpleRawInsertTest
[       OK ] ExecutorTest.SimpleRawInsertTest (93 ms)
[ RUN      ] ExecutorTest.SimpleUpdateTest
[       OK ] ExecutorTest.SimpleUpdateTest (109 ms)
[----------] 4 tests from ExecutorTest (426 ms total)

[----------] Global test environment tear-down
[==========] 4 tests from 1 test suite ran. (426 ms total)
[  PASSED  ] 4 tests.
(base) linux@ubuntu:~/db/minisql/build/test$
```

3.git 推送记录

```
commit 07054c872740a43ad298dc01b21f2eca99149874 (HEAD -> main, origin/main, origin/HEAD)
Author: cht <1051292989@qq.com>
Date:   Thu May 8 15:40:34 2025 +0800

    executor_engine modify

commit b622abd57e7f12bd4dc024d434f2a7059cadc9f9
Author: cht <1051292989@qq.com>
Date:   Thu May 8 10:51:44 2025 +0800

    finish part 5

commit 22c66c771f4c8c51a7df4408d4dedcbcbbff197a
Author: cht <1051292989@qq.com>
Date:   Wed May 7 10:55:49 2025 +0800

    diskmanager modify

commit fa3851ea1c84e5afc191bf0a1e99641027cba4f2
Author: cht <1051292989@qq.com>
Date:   Wed May 7 10:34:59 2025 +0800

    lru modify

commit c0b7b10a9c75e64f338ec9cf5d08c5a6b7e30bcd
Author: DFPitte <2093732622@qq.com>
Date:   Tue May 6 21:50:13 2025 +0800

    bug fix

commit 0317885db2eeb52edce817446fb06c2136dbc34e
Author: DFPitte <2093732622@qq.com>
Date:   Fri May 2 20:11:20 2025 +0800

    finished lab7
.
```