实验六 MiniSQL的实现（模块五）

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，熟悉执行器的实现。

3、学习通过语法树调用数据库相关组件，提升对于数据库系统的理解。

实验平台：

1、操作系统： WSL：Ubuntu

实验内容和要求：

1、实现SQL执行器。

执行器的实现和检验：

1、体系结构

执行器有以下函数需要我们实现。其基本操作为通过一个语法树根节点对语法树进行解析，得到相应的操作命令，同时向下传递指令，通过数据库底层的类和数据结构对命令进行执行操作。

```
dberr_t ExecuteCreateDatabase(pSyntaxNode ast, ExecuteContext *context);

dberr_t ExecuteDropDatabase(pSyntaxNode ast, ExecuteContext *context);

dberr_t ExecuteShowDatabases(pSyntaxNode ast, ExecuteContext *context);

dberr_t ExecuteUseDatabase(pSyntaxNode ast, ExecuteContext *context);

dberr_t ExecuteShowTables(pSyntaxNode ast, ExecuteContext *context);

dberr_t ExecuteCreateTable(pSyntaxNode ast, ExecuteContext *context);

dberr_t ExecuteDropTable(pSyntaxNode ast, ExecuteContext *context);

dberr_t ExecuteShowIndexes(pSyntaxNode ast, ExecuteContext *context);

dberr_t ExecuteCreateIndex(pSyntaxNode ast, ExecuteContext *context);
```

```
  dberr_t ExecuteDropIndex(pSyntaxNode ast, ExecuteContext *context);

  dberr_t ExecuteSelect(pSyntaxNode ast, ExecuteContext *context);

  dberr_t ExecuteInsert(pSyntaxNode ast, ExecuteContext *context);

  dberr_t ExecuteDelete(pSyntaxNode ast, ExecuteContext *context);
```

2、具体实现

在数据库层面，通过`dbs_`这个`unordered_map`传入数据库名以获取当前的数据库存储引擎的指针，通过该指针可以调用数据库中的各项次级结构。

```
  [[maybe_unused]] std::unordered_map<std::string, DBStorageEngine *> dbs_;  /**
all opened databases */
  [[maybe_unused]] std::string current_db_;  /** current database */
```

在表的层面，以如下代码为例，通过`DBStorageEngine`下的次级结构`CatalogManager`可以获取有关单个表的信息，保存在`TableInfo`中间。

```
  DBStorageEngine* current_db_engine = dbs_[current_db_];
  if(current_db_engine == nullptr)
  {
    std::cout << "no database selected." << endl;
    return DB_FAILED;
  }

  std::vector<TableInfo*> vec_table_info;
  current_db_engine->catalog_mgr_->GetTables(vec_table_info);
  for(int i=0; i<int(vec_table_info.size()); i++)
  {
    std::cout << vec_table_info[i]->GetTableName() << std::endl;
  }
  std::cout << "table(s) of number: " << vec_table_info.size() << endl;
  return DB_SUCCESS;
```

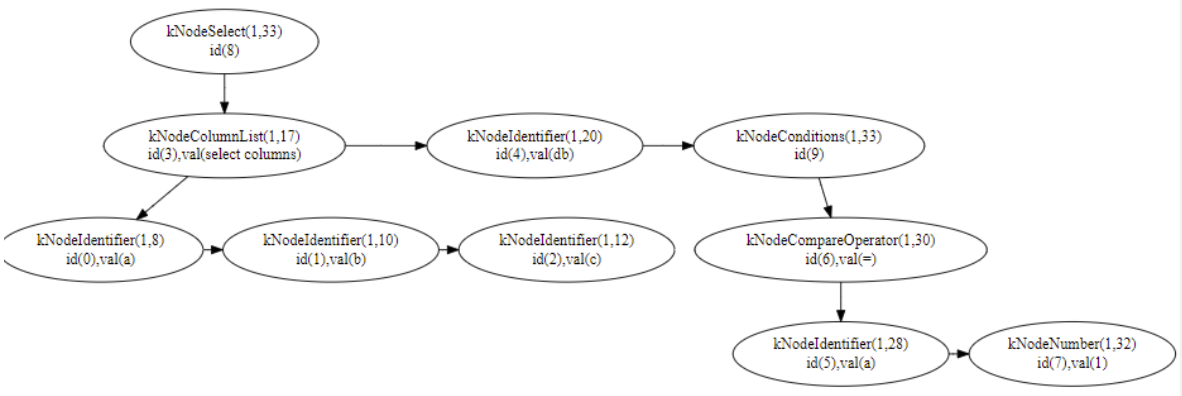
在索引的层面，同样是从`Catalogmanager`中获取相应的信息。

```
  vector<TableInfo*> vec_tableinfo;
  // vector<vector<IndexInfo*>> vvec_indexinfo;
  dberr_t if_gettable_success = dbs_[current_db_]->catalog_mgr_-
>GetTables(vec_tableinfo);
  if(if_gettable_success != DB_SUCCESS)
    return if_gettable_success;
  CatalogManager* current_CMgr = dbs_[current_db_]->catalog_mgr_;
  for(TableInfo* tmp_tableinfo: vec_tableinfo)
  {
    vector<IndexInfo*> vec_tmp_indexinfo;
    string table_name = tmp_tableinfo->GetTableName();
    dberr_t if_getindex_success = current_CMgr->GetTableIndexes(table_name,
vec_tmp_indexinfo);
    if(if_getindex_success != DB_SUCCESS)
      return if_getindex_success;
    for(IndexInfo* tmp_indexinfo: vec_tmp_indexinfo)
    {
      string index_name = tmp_indexinfo->GetIndexName();
      std::cout << index_name << endl;
    }
  }
```

在获取单个元组的数据的层面，通过`TableHeap`进行`GetTuple`操作，从而获取单个元组的数据。

```
    uint32_t index_val;
    RowId r_rowid(rowid);
    Row row(r_rowid);
    tmp_table_info->GetTableHeap()->GetTuple(&row, nullptr);
```

语法树架构方面，语法树中保存着命令执行需要的各种信息，根据信息和指令对于各项数据进行调用。

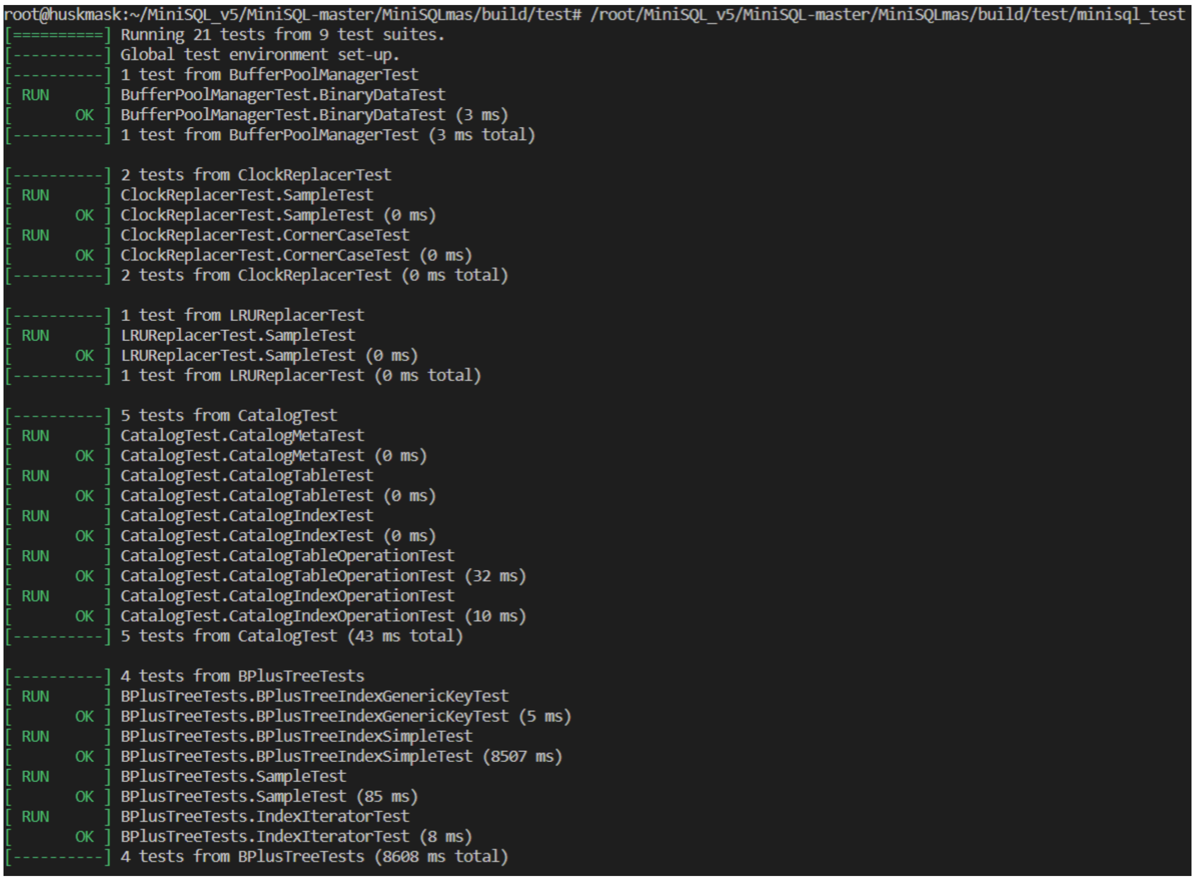


```
/**
 * Syntax node definition used in abstract syntax tree.
 */
struct SyntaxNode {
  int id_;    /** node id for allocated syntax node, used for debug */
  SyntaxNodeType type_; /** syntax node type */
  int line_no_; /** line number of this syntax node appears in sql */
  int col_no_;  /** column number of this syntax node appears in sql */
  struct SyntaxNode *child_;  /** children of this syntax node */
  struct SyntaxNode *next_;   /** siblings of this syntax node, linked by a
single linked list */
  char *val_; /** attribute value of this syntax node, use deep copy */
};
typedef struct SyntaxNode *pSyntaxNode;
```

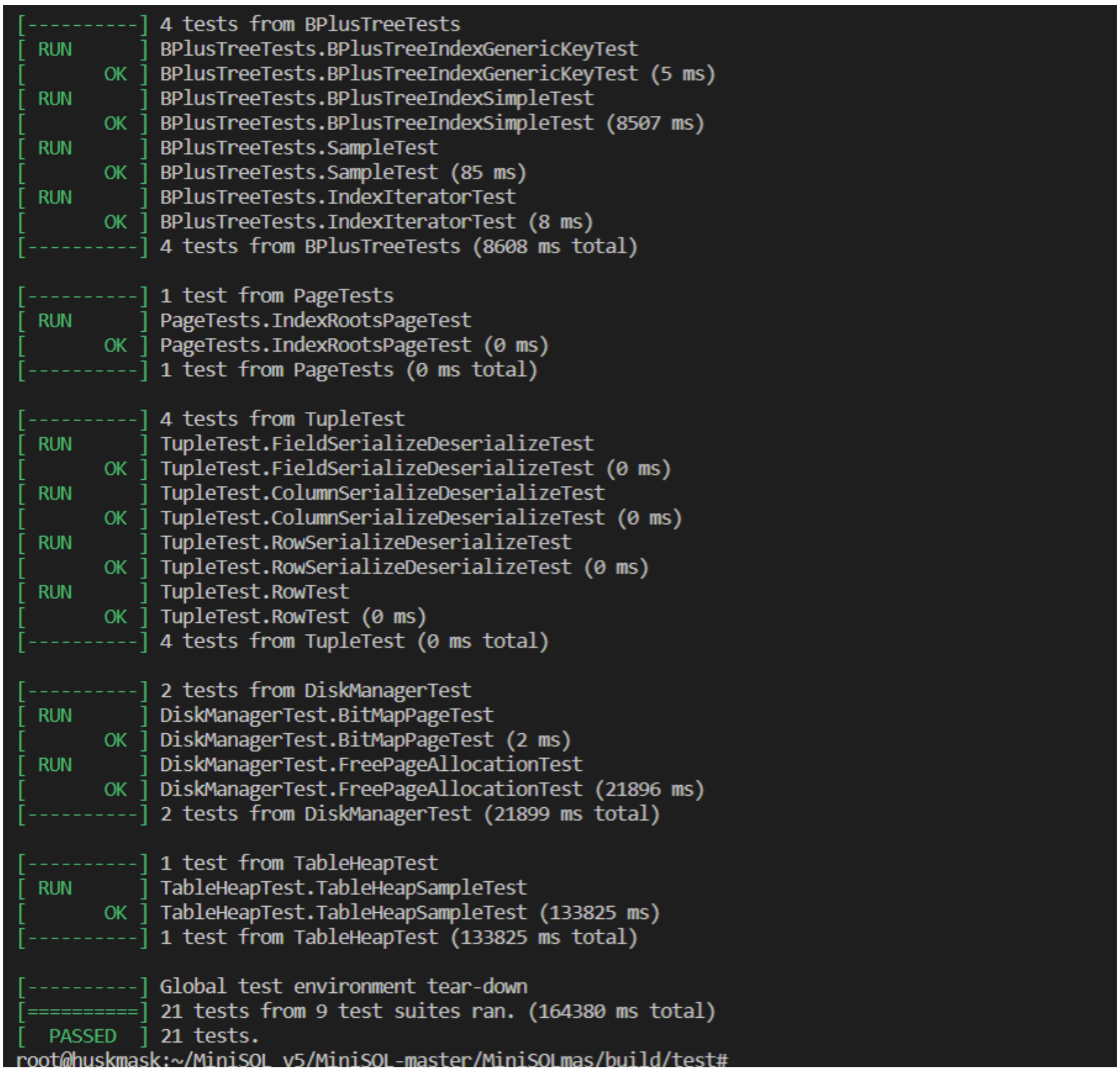
3、正确性检验

前置模块检验

首先在检验前四个模块能够通过测试的基础上：



```
root@huskmask:~/MiniSQL_v5/MiniSQL-master/MiniSQLmas/build/test# /root/MiniSQL_v5/MiniSQL-master/MiniSQLmas/build/test/minisql_test
[==========] Running 21 tests from 9 test suites.
[----------] Global test environment set-up.
[----------] 1 test from BufferPoolManagerTest
[ RUN      ] BufferPoolManagerTest.BinaryDataTest
[       OK ] BufferPoolManagerTest.BinaryDataTest (3 ms)
[----------] 1 test from BufferPoolManagerTest (3 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)

[----------] 1 test from LRUReplacerTest
[ RUN      ] LRUReplacerTest.SampleTest
[       OK ] LRUReplacerTest.SampleTest (0 ms)
[----------] 1 test from LRUReplacerTest (0 ms total)

[----------] 5 tests from CatalogTest
[ RUN      ] CatalogTest.CatalogMetaTest
[       OK ] CatalogTest.CatalogMetaTest (0 ms)
[ RUN      ] CatalogTest.CatalogTableTest
[       OK ] CatalogTest.CatalogTableTest (0 ms)
[ RUN      ] CatalogTest.CatalogIndexTest
[       OK ] CatalogTest.CatalogIndexTest (0 ms)
[ RUN      ] CatalogTest.CatalogTableOperationTest
[       OK ] CatalogTest.CatalogTableOperationTest (32 ms)
[ RUN      ] CatalogTest.CatalogIndexOperationTest
[       OK ] CatalogTest.CatalogIndexOperationTest (10 ms)
[----------] 5 tests from CatalogTest (43 ms total)

[----------] 4 tests from BPlusTreeTests
[ RUN      ] BPlusTreeTests.BPlusTreeIndexGenericKeyTest
[       OK ] BPlusTreeTests.BPlusTreeIndexGenericKeyTest (5 ms)
[ RUN      ] BPlusTreeTests.BPlusTreeIndexSimpleTest
[       OK ] BPlusTreeTests.BPlusTreeIndexSimpleTest (8507 ms)
[ RUN      ] BPlusTreeTests.SampleTest
[       OK ] BPlusTreeTests.SampleTest (85 ms)
[ RUN      ] BPlusTreeTests.IndexIteratorTest
[       OK ] BPlusTreeTests.IndexIteratorTest (8 ms)
[----------] 4 tests from BPlusTreeTests (8608 ms total)
```



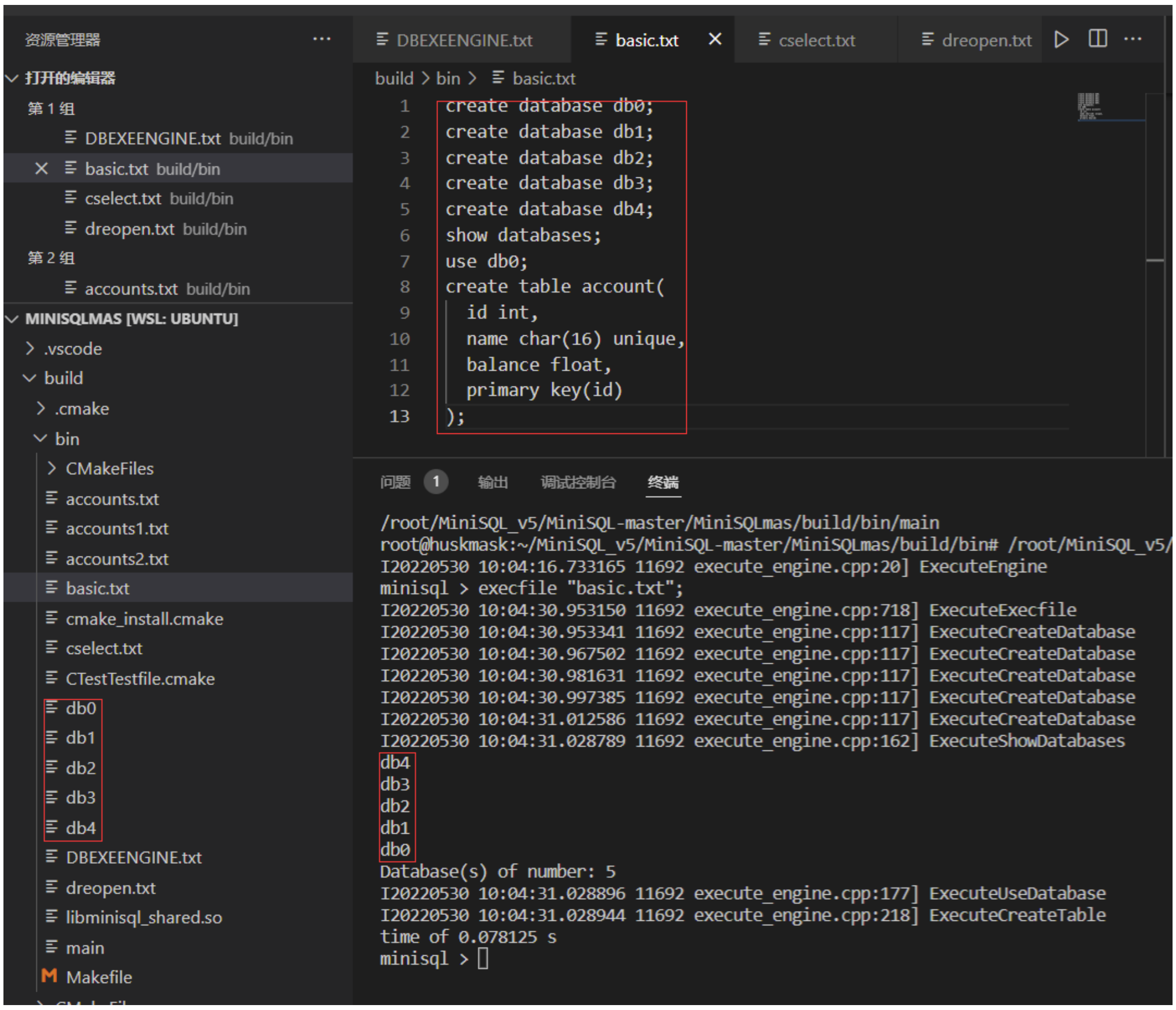
```
[----------] 4 tests from BPlusTreeTests
[ RUN      ] BPlusTreeTests.BPlusTreeIndexGenericKeyTest
[       OK ] BPlusTreeTests.BPlusTreeIndexGenericKeyTest (5 ms)
[ RUN      ] BPlusTreeTests.BPlusTreeIndexSimpleTest
[       OK ] BPlusTreeTests.BPlusTreeIndexSimpleTest (8507 ms)
[ RUN      ] BPlusTreeTests.SampleTest
[       OK ] BPlusTreeTests.SampleTest (85 ms)
[ RUN      ] BPlusTreeTests.IndexIteratorTest
[       OK ] BPlusTreeTests.IndexIteratorTest (8 ms)
[----------] 4 tests from BPlusTreeTests (8608 ms total)
```

```
[----------] 1 test from PageTests
[ RUN      ] PageTests.IndexRootsPageTest
[       OK ] PageTests.IndexRootsPageTest (0 ms)
[----------] 1 test from PageTests (0 ms total)

[----------] 4 tests from TupleTest
[ RUN      ] TupleTest.FieldSerializeDeserializeTest
[       OK ] TupleTest.FieldSerializeDeserializeTest (0 ms)
[ RUN      ] TupleTest.ColumnSerializeDeserializeTest
[       OK ] TupleTest.ColumnSerializeDeserializeTest (0 ms)
[ RUN      ] TupleTest.RowSerializeDeserializeTest
[       OK ] TupleTest.RowSerializeDeserializeTest (0 ms)
[ RUN      ] TupleTest.RowTest
[       OK ] TupleTest.RowTest (0 ms)
[----------] 4 tests from TupleTest (0 ms total)

[----------] 2 tests from DiskManagerTest
[ RUN      ] DiskManagerTest.BitMapPageTest
[       OK ] DiskManagerTest.BitMapPageTest (2 ms)
[ RUN      ] DiskManagerTest.FreePageAllocationTest
[       OK ] DiskManagerTest.FreePageAllocationTest (21896 ms)
[----------] 2 tests from DiskManagerTest (21899 ms total)

[----------] 1 test from TableHeapTest
[ RUN      ] TableHeapTest.TableHeapSampleTest
[       OK ] TableHeapTest.TableHeapSampleTest (133825 ms)
[----------] 1 test from TableHeapTest (133825 ms total)

[----------] Global test environment tear-down
[==========] 21 tests from 9 test suites ran. (164380 ms total)
[  PASSED  ] 21 tests.
root@huskmask:~/MiniSQL_v5/MiniSQL-master/MiniSQLmas/build/test#
```

建表建库操作检验

在此基础上完成模块五的编写后通过命令行指令进行检验。

首先执行文件内指令，进行建库和建表操作：



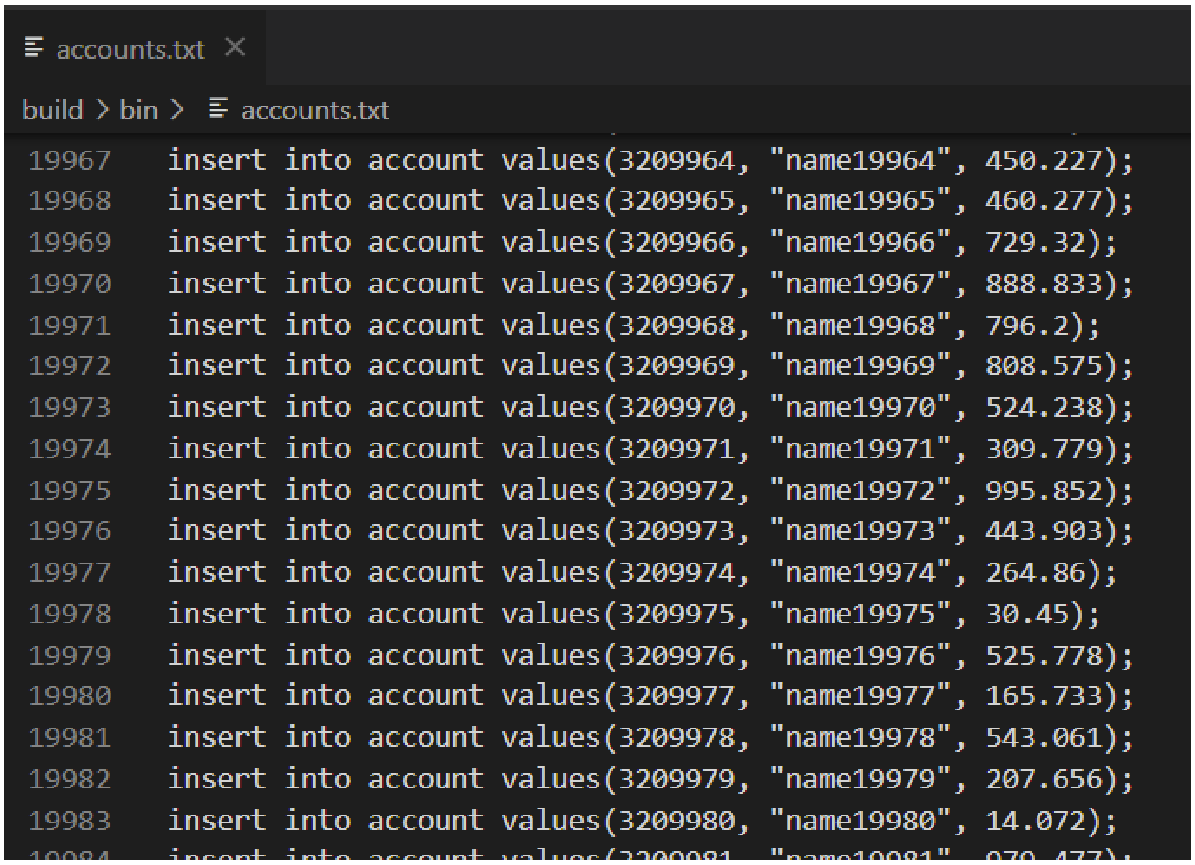
```
create database db0;
create database db1;
create database db2;
create database db3;
create database db4;
show databases;
use db0;
create table account(
  id int,
  name char(16) unique,
  balance float,
  primary key(id)
);
```

```
/root/MiniSQL_v5/MiniSQL-master/MiniSQLmas/build/bin/main
root@huskmask:~/MiniSQL_v5/MiniSQL-master/MiniSQLmas/build/bin# /root/MiniSQL_v5/
I20220530 10:04:16.733165 11692 execute_engine.cpp:20] ExecuteEngine
minisql > execfile "basic.txt";
I20220530 10:04:30.953150 11692 execute_engine.cpp:718] ExecuteExecfile
I20220530 10:04:30.953341 11692 execute_engine.cpp:117] ExecuteCreateDatabase
I20220530 10:04:30.967502 11692 execute_engine.cpp:117] ExecuteCreateDatabase
I20220530 10:04:30.981631 11692 execute_engine.cpp:117] ExecuteCreateDatabase
I20220530 10:04:30.997385 11692 execute_engine.cpp:117] ExecuteCreateDatabase
I20220530 10:04:31.012586 11692 execute_engine.cpp:117] ExecuteCreateDatabase
I20220530 10:04:31.028789 11692 execute_engine.cpp:162] ExecuteShowDatabases
db4
db3
db2
db1
db0
Database(s) of number: 5
I20220530 10:04:31.028896 11692 execute_engine.cpp:177] ExecuteUseDatabase
I20220530 10:04:31.028944 11692 execute_engine.cpp:218] ExecuteCreateTable
time of 0.078125 s
minisql > 
```

建表操作成功进行，数据库文件也成功生成。

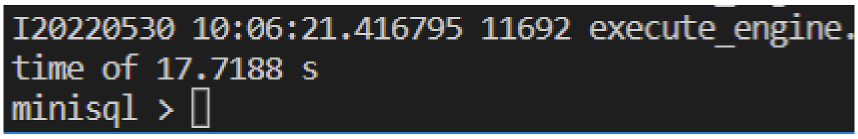
插入操作

执行`execfile`指令，执行`accounts.txt`中的插入指令，插入`20000`条数据。



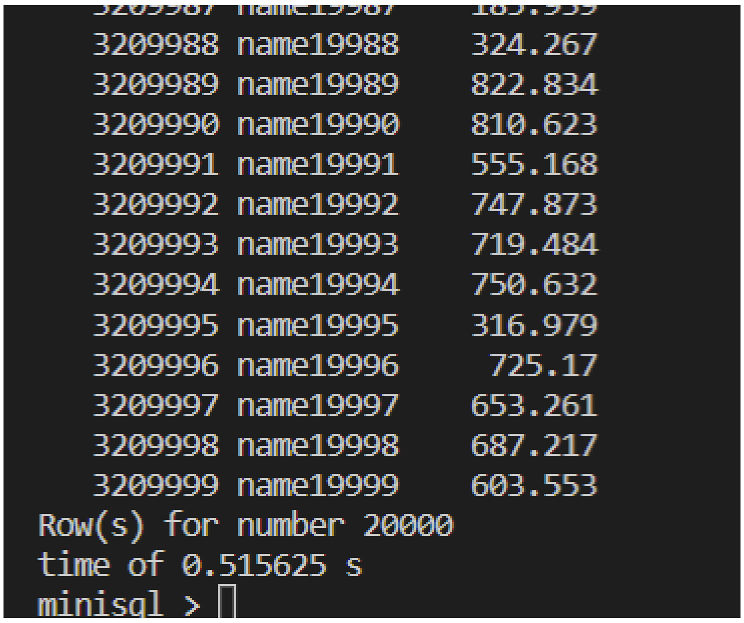
```
insert into account values(3209964, "name19964", 450.227);
insert into account values(3209965, "name19965", 460.277);
insert into account values(3209966, "name19966", 729.32);
insert into account values(3209967, "name19967", 888.833);
insert into account values(3209968, "name19968", 796.2);
insert into account values(3209969, "name19969", 808.575);
insert into account values(3209970, "name19970", 524.238);
insert into account values(3209971, "name19971", 309.779);
insert into account values(3209972, "name19972", 995.852);
insert into account values(3209973, "name19973", 443.903);
insert into account values(3209974, "name19974", 264.86);
insert into account values(3209975, "name19975", 30.45);
insert into account values(3209976, "name19976", 525.778);
insert into account values(3209977, "name19977", 165.733);
insert into account values(3209978, "name19978", 543.061);
insert into account values(3209979, "name19979", 207.656);
insert into account values(3209980, "name19980", 14.072);
```

插入成功进行，耗时17.72s。



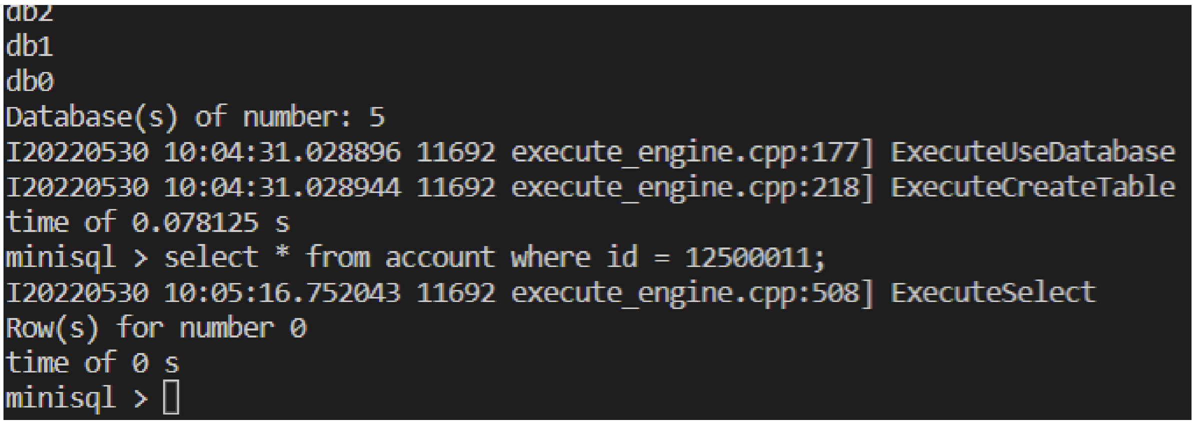
```
I20220530 10:06:21.416795 11692 execute_engine.
time of 17.7188 s
minisql > 
```

执行全选，检验插入成功：



```
    3209988 name19988    324.267
    3209989 name19989    822.834
    3209990 name19990    810.623
    3209991 name19991    555.168
    3209992 name19992    747.873
    3209993 name19993    719.484
    3209994 name19994    750.632
    3209995 name19995    316.979
    3209996 name19996     725.17
    3209997 name19997    653.261
    3209998 name19998    687.217
    3209999 name19999    603.553
Row(s) for number 20000
time of 0.515625 s
minisql > 
```

选取操作



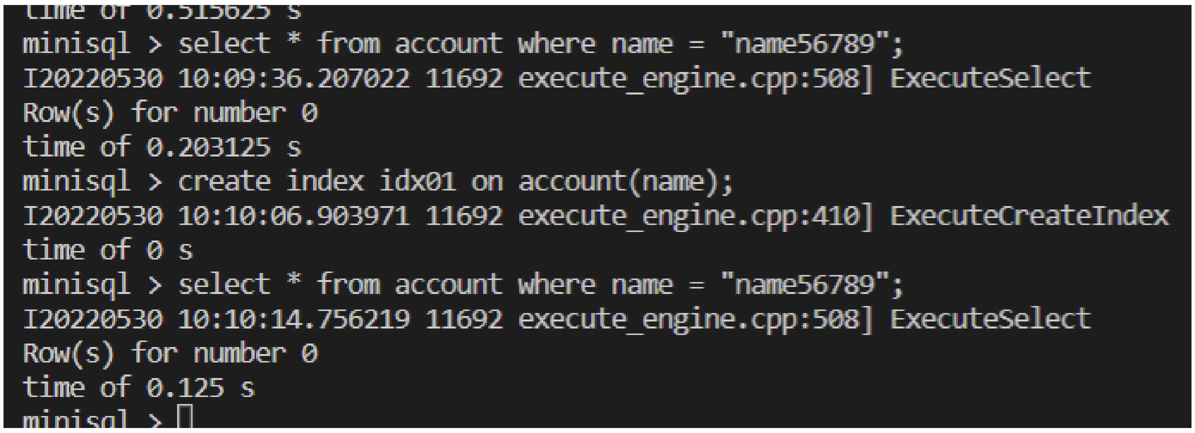
```
db2
db1
db0
Database(s) of number: 5
I20220530 10:04:31.028896 11692 execute_engine.cpp:177] ExecuteUseDatabase
I20220530 10:04:31.028944 11692 execute_engine.cpp:218] ExecuteCreateTable
time of 0.078125 s
minisql > select * from account where id = 12500011;
I20220530 10:05:16.752043 11692 execute_engine.cpp:508] ExecuteSelect
Row(s) for number 0
time of 0 s
minisql >
```

执行`select`语句，可以看到在插入完成后能够成功获取指定id的信息。

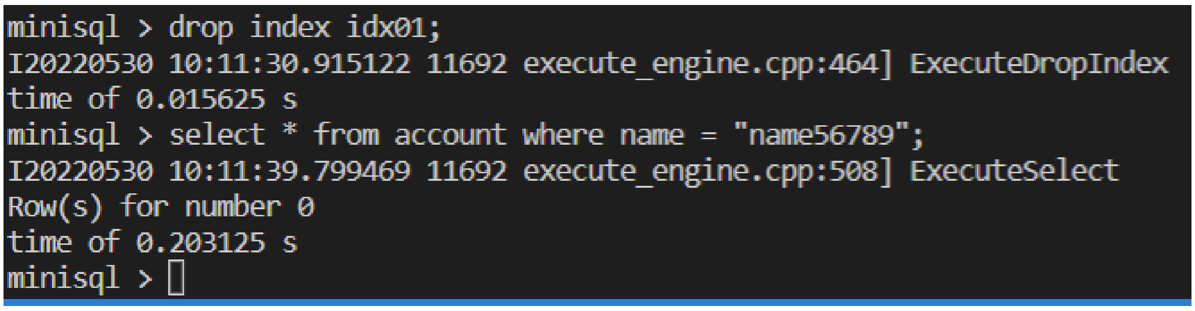
删除、建立索引操作

建索引前后搜索时间对比，可以明显看到搜索速度有所提升，删除后下降，说明索引成功建成，并且能够

成功地提升搜索的效率。



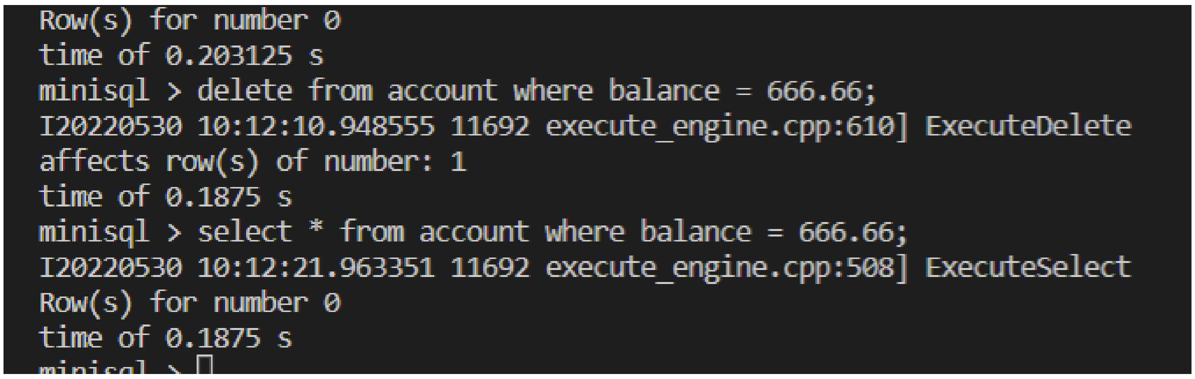
```
time of 0.515625 s
minisql > select * from account where name = "name56789";
I20220530 10:09:36.207022 11692 execute_engine.cpp:508] ExecuteSelect
Row(s) for number 0
time of 0.203125 s
minisql > create index idx01 on account(name);
I20220530 10:10:06.903971 11692 execute_engine.cpp:410] ExecuteCreateIndex
time of 0 s
minisql > select * from account where name = "name56789";
I20220530 10:10:14.756219 11692 execute_engine.cpp:508] ExecuteSelect
Row(s) for number 0
time of 0.125 s
minisql >
```



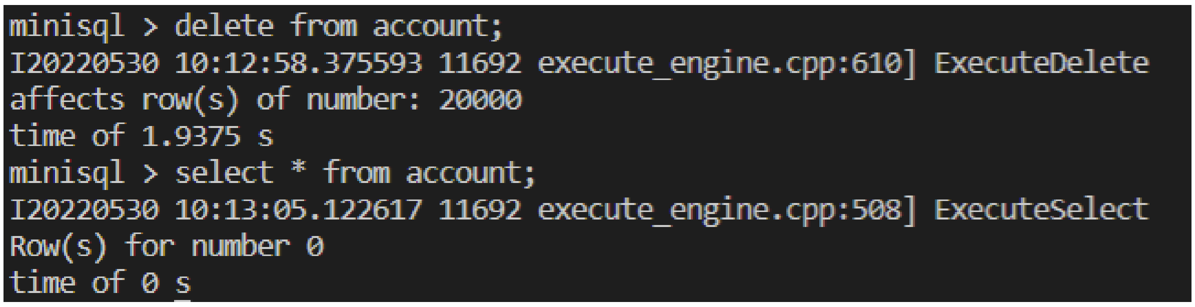
```
minisql > drop index idx01;
I20220530 10:11:30.915122 11692 execute_engine.cpp:464] ExecuteDropIndex
time of 0.015625 s
minisql > select * from account where name = "name56789";
I20220530 10:11:39.799469 11692 execute_engine.cpp:508] ExecuteSelect
Row(s) for number 0
time of 0.203125 s
minisql >
```

删除操作

通过执行`delete`对于元组进行删除：



```
Row(s) for number 0
time of 0.203125 s
minisql > delete from account where balance = 666.66;
I20220530 10:12:10.948555 11692 execute_engine.cpp:610] ExecuteDelete
affects row(s) of number: 1
time of 0.1875 s
minisql > select * from account where balance = 666.66;
I20220530 10:12:21.963351 11692 execute_engine.cpp:508] ExecuteSelect
Row(s) for number 0
time of 0.1875 s
minisql >
```

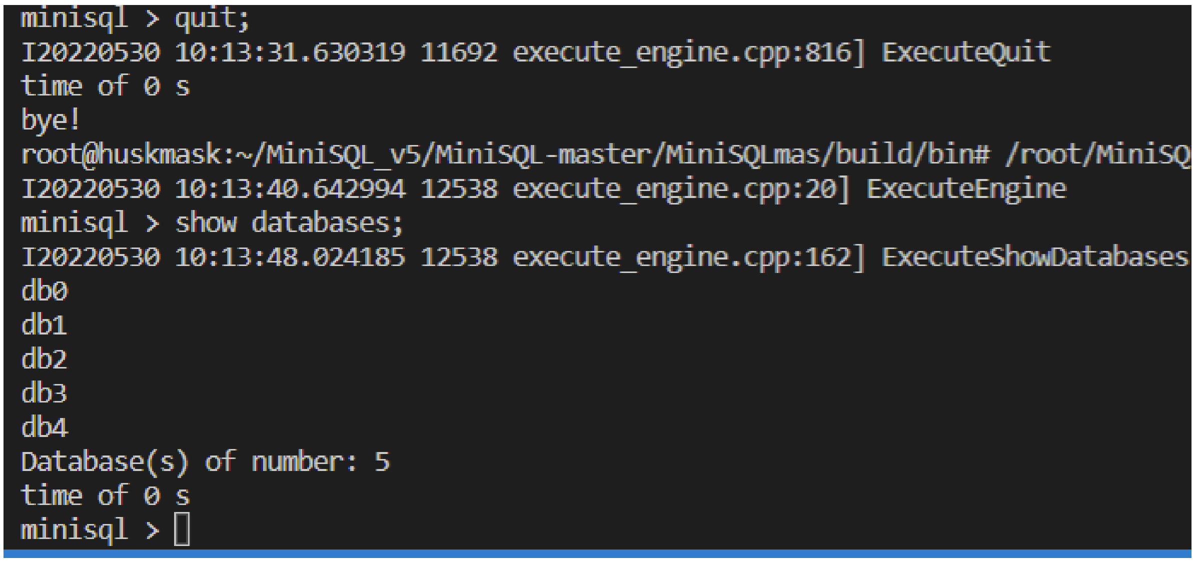


```
minisql > delete from account;
I20220530 10:12:58.375593 11692 execute_engine.cpp:610] ExecuteDelete
affects row(s) of number: 20000
time of 1.9375 s
minisql > select * from account;
I20220530 10:13:05.122617 11692 execute_engine.cpp:508] ExecuteSelect
Row(s) for number 0
time of 0 s
```

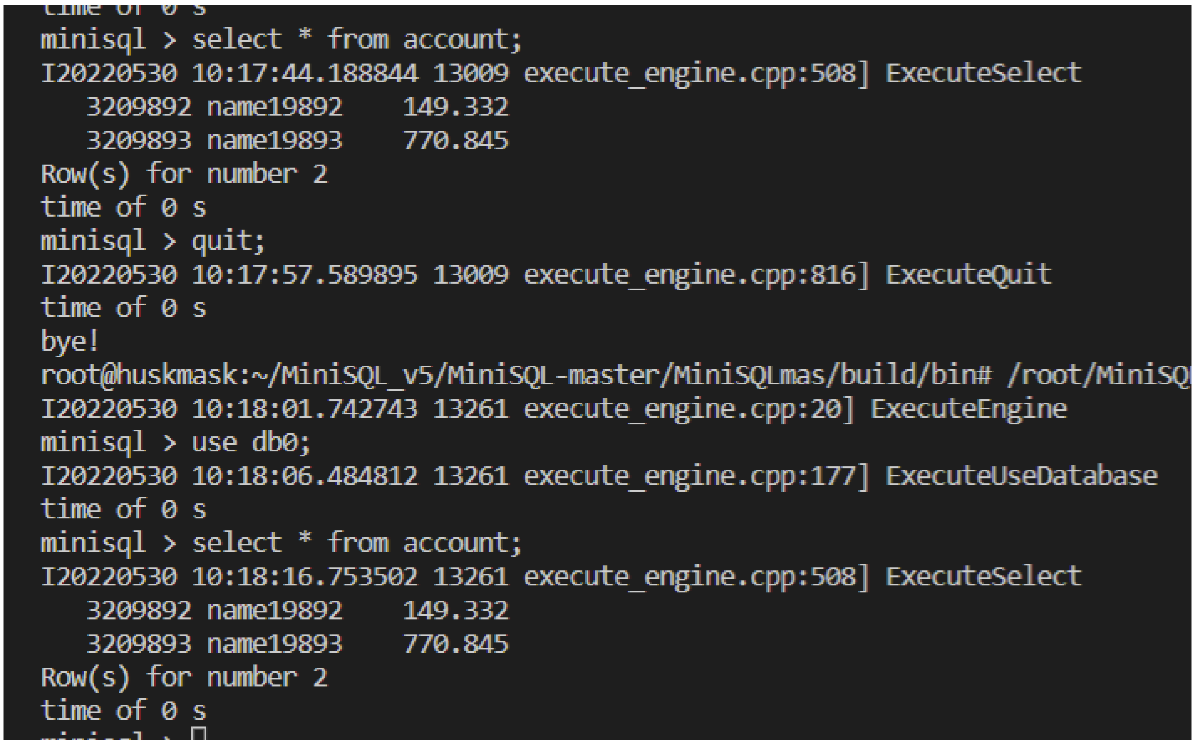
可以看到删除操作得到正确的结果。

数据持久性检验

执行`quit`指令后重新运行程序，可以发现之前一次的数据都保留在数据库中：



```
minisql > quit;
I20220530 10:13:31.630319 11692 execute_engine.cpp:816] ExecuteQuit
time of 0 s
bye!
root@huskmask:~/MiniSQL_v5/MiniSQL-master/MiniSQLmas/build/bin# /root/MiniSQ
I20220530 10:13:40.642994 12538 execute_engine.cpp:20] ExecuteEngine
minisql > show databases;
I20220530 10:13:48.024185 12538 execute_engine.cpp:162] ExecuteShowDatabases
db0
db1
db2
db3
db4
Database(s) of number: 5
time of 0 s
minisql > 
```



```
time of 0 s
minisql > select * from account;
I20220530 10:17:44.188844 13009 execute_engine.cpp:508] ExecuteSelect
   3209892 name19892    149.332
   3209893 name19893    770.845
Row(s) for number 2
time of 0 s
minisql > quit;
I20220530 10:17:57.589895 13009 execute_engine.cpp:816] ExecuteQuit
time of 0 s
bye!
root@huskmask:~/MiniSQL_v5/MiniSQL-master/MiniSQLmas/build/bin# /root/MiniSQ
I20220530 10:18:01.742743 13261 execute_engine.cpp:20] ExecuteEngine
minisql > use db0;
I20220530 10:18:06.484812 13261 execute_engine.cpp:177] ExecuteUseDatabase
time of 0 s
minisql > select * from account;
I20220530 10:18:16.753502 13261 execute_engine.cpp:508] ExecuteSelect
   3209892 name19892    149.332
   3209893 name19893    770.845
Row(s) for number 2
time of 0 s
minisql > 
```

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。

更改`Row`等部分的操作，在底层获取数据的功能，目前很多操作比如使用`RowId`获取元组的操作需要在顶层操作，导致不同同学之间的分工不明确。增加`TableInfo`持久化的测试样例，目前这部分没有测试样例。