《数据库系统原理》实验报告（）					
题目：MINIOB 实验一					
学号	2152809	姓名	曾崇然	日期	2023-11-9

实验环境：从 docer 仓库中拉取的 oceanbase/miniob 镜像，在运行的容器中从 github 上克隆 miniob 的源代码，进行编译和运行

实验步骤及结果截图：

1.创建 miniob 环境

a)克隆代码

```
# git clone https://github.com/oceanbase/miniob.git
Cloning into 'miniob'...
remote: Enumerating objects: 4434, done.
remote: Counting objects: 100% (2351/2351), done.
remote: Compressing objects: 100% (518/518), done.
remote: Total 4434 (delta 1886), reused 1850 (delta 1833), pack-reused 2083
Receiving objects: 100% (4434/4434), 26.46 MiB | 3.41 MiB/s, done.
Resolving deltas: 100% (2890/2890), done.
```

b)编译运行

```
# cd miniob
# ls
benchmark  cmake          CODE_OF_CONDUCT.md  deps    docs      etc      NOTICE     src   tools
build.sh   CMakeLists.txt  CONTRIBUTING.md     docker  Doxyfile  License  README.md  test  unittest
# bash build.sh --make -j4
build.sh --make -j4
```

```
# ./bin/observer -s miniob.sock -f ../etc/observer.ini &
# Successfully load ../etc/observer.ini
./bin/obclient -s miniob.sock
```

2.创建表

```
miniob > create table Scores(id int,name char(10),score float);
SUCCESS
miniob > show tables;
Tables_in_SYS
Scores
```

3.插入数据

```
miniob > insert into Scores values(2251435,'李明浩',81.2);
SUCCESS
miniob > insert into Scores values(2210465,'赵毅斌',91.3);
SUCCESS
miniob > insert into Scores values(2332133,'刘孔阳',56.3);
SUCCESS
miniob > insert into Scores values(2231435,'王亚伟',73.2);
SUCCESS
miniob > insert into Scores values(1950723,'孙鹏翼',89.2);
SUCCESS
```

4.展示学号姓名

```
miniob > select * from Scores;
id | name | score
2251435 | 李明浩 | 81.2
2210465 | 赵毅斌 | 91.3
2332133 | 刘孔阳 | 56.3
2231435 | 王亚伟 | 73.2
1950723 | 孙鹏翼 | 89.2
```

5.修改成绩

```
miniob > update Scores set score=91.3 where id=2251435;
SUCCESS
miniob > update Scores set score=87.2 where id=2231435;
SUCCESS
```

结果：没有修改成功

```
miniob > select * from Scores;
id | name | score
2251435 | 李明浩 | 81.2
2210465 | 赵毅斌 | 91.3
2332133 | 刘孔阳 | 56.3
2231435 | 王亚伟 | 73.2
1950723 | 孙鹏翼 | 89.2
```

原因：miniob 是一个用于学习的的数据库，其虽然实现了 update 功能的词法和语法解析，但是没有实现其对应的执行，也就是说 update 并没有实际的执行

6.删除记录

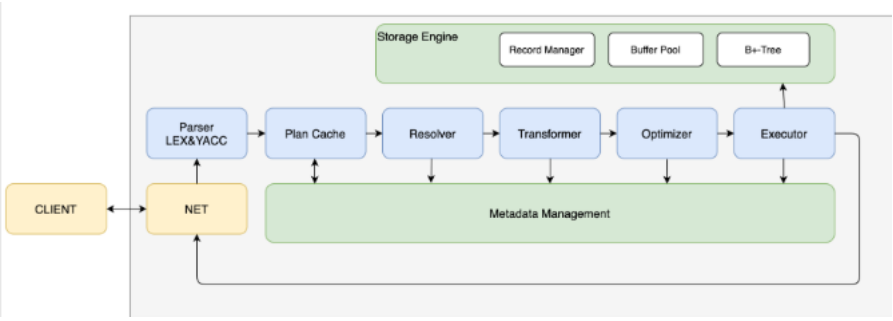
```
miniob > delete from Scores where name='赵毅斌';
SUCCESS
miniob > delete from Scores where name='孙鹏翼';
SUCCESS
```

```
miniob > select * from Scores;
id | name | score
2251435 | 李明浩 | 81.2
2332133 | 刘孔阳 | 56.3
2231435 | 王亚伟 | 73.2
```

7.代码分析和理解

选取功能：create table

Miniob 的框架如下：



具体到 create table：

a)client 接收输入的命令，create table Scores(id int,name chars(10),score floats)通过网络模块传递到服务端(observer)

b)服务端接收到该 sql 语句后进行处理，在 session_stage.cpp 的 handle_sql 中定义了处理的流程

```
RC rc = query_cache_stage_.handle_request(sql_event);
```

```
rc = parse_stage_.handle_request(sql_event);
```

```
 rc = resolve_stage_.handle_request(sql_event);
```

```
rc = optimize_stage_.handle_request(sql_event);
```

```
rc = execute_stage_.handle_request(sql_event);
```

c)cache_stage 目前是直接跳过了

```
RC QueryCacheStage::handle_request(SQLStageEvent *sql_event)
{
  return RC::SUCCESS;
}
```

d)parser_stage 对输入的 create 语句进行词法和语法解析，将分析的结果如语句类型，属性等放入对应的数据结构中，传递给下一个阶段

c)resolve_stage 根据语法解析出来的结果来创建对应的执行语句，将其放入 sql_event 中交给下一阶段，具体为：

先获取当前数据库：

```
Db *db = session_event->session()->get_current_db();
```

创建执行语句：是根据语句的类型(create)将其分发给具体的创建函数进行创建

```
rc = Stmt::create_stmt(db, *sql_node, stmt);
```

修改事件：

```
 sql_event->set_stmt(stmt);
```

d)optimize_stage create 语句在这个阶段没有优化，直接返回了

```
37        RC rc = create_logical_plan(sql_event, logical_operator);
(gdb) n
38        if (rc != RC::SUCCESS) {
(gdb) n
39          if (rc != RC::UNIMPLENMENT) {
(gdb) n
42          return rc;
(gdb) print rc
$1 = RC::UNIMPLENMENT
```

e)execute_stage 首先判断有没有物理执行计划，create table 没有：

```
(gdb) n
38        if (physical_operator != nullptr) {
(gdb) n
42        SessionEvent *session_event = sql_event->session_event();
```

然后根据执行语句去进行执行

```
rc = command_executor.execute(sql_event);
```

具体的执行如下：

根据语句类型分发给具体的执行器：

```
case StmtType::CREATE_TABLE: {
  CreateTableExecutor executor;
  return executor.execute(sql_event);
} break;
```

在对应的 execute 函数中进入到 create_table 的执行：

```
RC rc = session->get_current_db()->create_table(table_name, attribute_count, create_table_stmt-
```

create_table 函数：

检查表名：

```
if (opened_tables_.count(table_name) != 0)
```

执行创建：

```
rc = table->create(table_id, table_file_path.c_str(), table_name, path_.c_str(), attribute
```

Table_create 函数：这是和存储和文件打交道的创建的物理操作包括以下部分：创建文件，记录元数据等，至此，数据表成功创建。

f)语句执行的结果通过 net 模块传递到客户端进行显示

出现的问题：

代码的阅读，尽管 miniob 是一个微型的数据库，但是其代码量依然很大，同时项目结构较为复杂，类很多，让人阅读和分析起来找不到方向。

解决方案：

1.查阅资料：我去观看了官方网站的视频和文档，对 miniob 的代码结构有了基础的了解

2.使用 chatGPT：对于阅读出现困难的部分我使用了 chatGPT 来进行辅助分析

3.使用 gdb 调试：通过 gdb 调试逐步观察一条 sql 语句的执行步骤，并深度到对应的代码部分进行理解，这让我对整个项目有了更多的理解