数据库系统

课程试验报告

试验题目： **MiniOB 数据库管理系统功能的补充与完善**

学 生 姓 名： 丁海桐

学 生 学 号： 202226010304

专 业 班 级： 软件 2203 班

开 课 时 间： 2023-2024-2 学期

试 验 日 期： 2024 年 3 月 23 日

1. 试验任务：

本课程的实验内容是：以教学用的数据库管理系统 MiniOB 作为载体，以问题作为驱动，掌握数据库核心技术。具体来说，先熟悉 MiniOB 的架构和基础代码，然后掌握编译、安装、测试、调试这些基本技能，再对 MiniOB 的功能进行完善。具体任务如下：

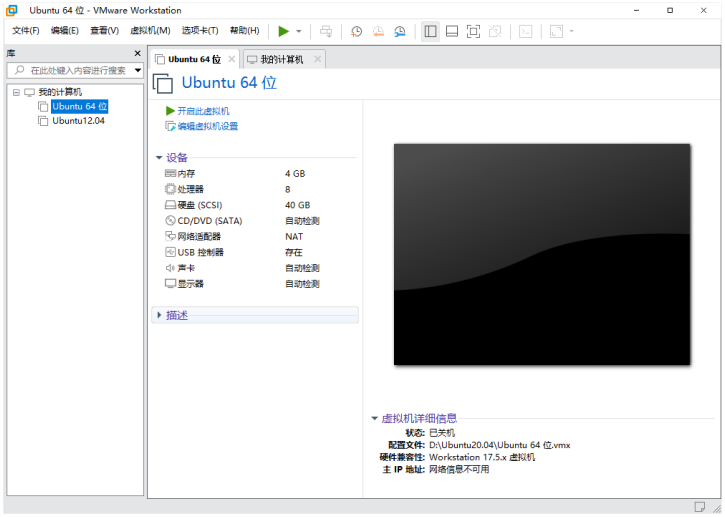
（1）验证 SELECT 语句的正确性。这里的正确性是指其中的表名和字段名。

（2）实现 drop table 这一功能。

2. 试验准备情况（对照任务，实验之前给出你的预案）：

（1） 配置 Linux 环境

使用的是 VMware 下的 Ubuntu20.04 版本。



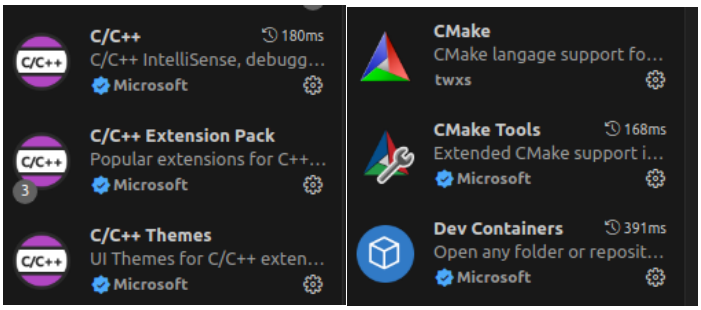
（

（2） 配置开发环境

1. 安装 Git、CMake、GCC 等程序。

```
dinghaitong@dinghaitong:~/my_miniob$ gcc --version
gcc (Ubuntu 9.4.0-1ubuntu1~20.04.2) 9.4.0
Copyright (C) 2019 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

dinghaitong@dinghaitong:~/my_miniob$ cmake --version
cmake version 3.29.0

CMake suite maintained and supported by Kitware (kitware.com/cmake).
dinghaitong@dinghaitong:~/my_miniob$ git --version
git version 2.25.1
```

2. 安装 Vscode，并配置相应插件。



（3） 下拉代码，完成初始化和首次编译及运行

https://github.com/oceanbase/miniob.git　#从 github 拉取代码

cd miniob

sudo bash build.sh init　　#初始化

sudo bash build.sh　　#编译

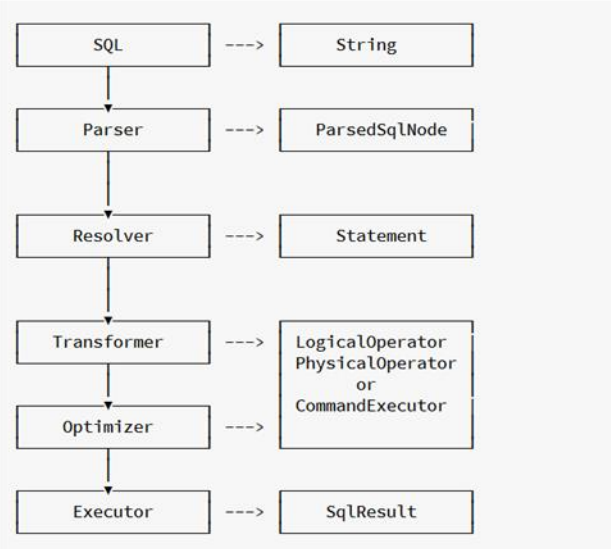
cd build

sudo ./bin/observer -f ../etc/observer.ini -P cli #启动

出现以下页面代表 miniob 成功运行，至此准备阶段结束。

```
dinghaitong@dinghaitong:~/my_miniob/build$ sudo ./bin/observer -f ../etc/observer.ini -P
error opening PID file /tmp/observer.pid,error:13:Permission denied
Successfully load ../etc/observer.ini
miniob >
```

（4） SQL 层理解



当用户在 miniob 中输入 SQL 语句时，该请求以字符串形式存储：

- Parser 阶段：将 SQL 字符串进行词法解析（lex_sql.l）和语法解析（yacc_sql.y），最终转换为 ParsedSqlNode（parse_defs.h）。

- Resolver 阶段：将 ParsedSqlNode 转换为 Stmt（Statement），进行语义解析。
- Transformer 和 Optimizer 阶段：将 Stmt 转换为 LogicalOperator，并在优化后输出 PhysicalOperator。这一阶段负责执行查询优化操作。
- 对于命令执行类型的 SQL 请求，系统会创建相应的 CommandExecutor。
- 执行阶段 Executor：将 PhysicalOperator（物理执行计划）转换为 SqlResult（执行结果），或者通过 SqlResult 输出 CommandExecutor 执行后的结果。

3. 试验过程记录（对照任务，对试验方案和结果进行记录和分析）：

（1） 验证 SELECT 语句的正确性。这里的正确性是指其中的表名和字段名。

```
miniob > show tables
Tables_in_SYS

miniob > create table demo1(id int, age int);
SUCCESS

miniob > show tables
Tables_in_SYS
demo1

miniob > insert into demo1 values (1, 20);
SUCCESS

miniob > insert into demo1 values (2, 28);
SUCCESS

miniob > select * from demo1;
id | age
1 | 20
2 | 28

miniob > select * frome demo1 where id=2;
SQL_SYNTAX > Failed to parse sql

miniob > select * from demo1 where id=2;
id | age
2 | 28
```

我们创建了表 demo1(id int, age int)，并向其插入了两个数据(1, 20), (2, 28)。
show tables 可见表被成功创建。select * from demo1，查找表中所有数据，成功；select * from demo1 where id=2，查找表中 id=2 的数据，成功。
题目 1 验证成功。

（2） 实现 drop table 这一功能。
我们从 SQL 的各个阶段按顺序查看是否有需要完善的地方。

1. Parser

src > observer > sql > parser > ≡ lex_sql.l

```
86  HELP                    RETURN_TOKEN(HELP);
87  DESC                    RETURN_TOKEN(DESC);
88  CREATE                  RETURN_TOKEN(CREATE);
89  DROP                    RETURN_TOKEN(DROP);
90  TABLE                   RETURN_TOKEN(TABLE);
91  TABLES                  RETURN_TOKEN(TABLES);
```

src > observer > sql > parser > yacc_sql.y

```
54  %parse-param { void * scanner }
55
56  //标识tokens
57  %token  SEMICOLON
58          CREATE
59          DROP
60          TABLE
61          TABLES
```

```
calc_stmt
| select_stmt
| insert_stmt
| update_stmt
| delete_stmt
| create_table_stmt
| drop_table_stmt
| show_tables_stmt
```

```
drop_table_stmt:    /*drop table 语句的语法解析树*/
    DROP TABLE ID {
      $$ = new ParsedSqlNode(SCF_DROP_TABLE);
      $$->drop_table.relation_name = $3;
      free($3);
    };
```

src > observer > sql > parser > parse_defs.h

```
288  class ParsedSqlNode
289  {
290  public:
291    enum SqlCommandFlag flag;
292    ErrorSqlNode        error;
293    CalcSqlNode         calc;
294    SelectSqlNode       selection;
295    InsertSqlNode       insertion;
296    DeleteSqlNode       deletion;
297    UpdateSqlNode       update;
298    CreateTableSqlNode  create_table;
299    DropTableSqlNode    drop_table;
300    CreateIndexSqlNode  create_index;
301    DropIndexSqlNode    drop_index;
```

由此可见 Parser 阶段已经完善。

2. Resolve

在 stmt.cpp 的 create_stmt()添加 case

src > observer > sql > stmt > stmt.cpp

```
34  {
37    switch (sql_node.flag) {
58    }
59    case SCF_DROP_TABLE: {
60      return DropTableStmt::create(db, sql_node.drop_table, stmt);
61    }
```

仿照该目录下其他文件格式，完成 drop_table_stmt.h 和 drop_table_stmt.cpp

src > observer > sql > stmt > C drop_table_stmt.h

```h
#pragma once

#include <string>
#include <vector>

#include "sql/stmt/stmt.h"

class Db;

/**
 * @brief 表示创建表的语句
 * @ingroup Statement
 * @details 虽然解析成了stmt，但是与原始的SQL解析后的数据也差不多
 */
class DropTableStmt : public Stmt
{
public:
  DropTableStmt(const std::string &table_name) : table_name_(table_name) {}
  virtual ~DropTableStmt() = default;

  StmtType type() const override { return StmtType::DROP_TABLE; }

  const std::string &table_name() const { return table_name_; }

  static RC create(Db *db, const DropTableSqlNode &desc_table, Stmt *&stmt);

private:
  std::string table_name_;
};

```

src > observer > sql > stmt > drop_table_stmt.cpp

```cpp
#include "sql/stmt/drop_table_stmt.h"

RC DropTableStmt::create(Db *db, const DropTableSqlNode &drop_table, Stmt *&stmt)
{
  stmt = new DropTableStmt(drop_table.relation_name);
  return RC::SUCCESS;
}
```

3. Transform & Optimizer

   完成

4. executor

   添加 drop_table_exector.h 和 drop_table_exector.cpp

src > observer > sql > executor > C drop_table_executor.h

```h
#pragma once

#include "common/rc.h"

class SQLStageEvent;

/**
 * @brief drop表的执行器
 * @ingroup Executor
 */
class DropTableExecutor
{
public:
  DropTableExecutor()          = default;
  virtual ~DropTableExecutor() = default;

  RC execute(SQLStageEvent *sql_event);
};
```

src > observer > sql > executor > drop_table_executor.cpp

```cpp
#include "sql/executor/drop_table_executor.h"

#include "common/log/log.h"
#include "event/session_event.h"
#include "event/sql_event.h"
#include "session/session.h"
#include "sql/stmt/drop_table_stmt.h"
#include "storage/db/db.h"


RC DropTableExecutor::execute(SQLStageEvent *sql_event)
{
  Stmt    *stmt    = sql_event->stmt();
  Session *session = sql_event->session_event()->session();
  ASSERT(stmt->type() == StmtType::DROP_TABLE,
      "drop table executor can not run this command: %d",
      static_cast<int>(stmt->type()));

  DropTableStmt *drop_table_stmt = static_cast<DropTableStmt *>(stmt);

  const char *table_name = drop_table_stmt->table_name().c_str();

  RC rc = session->get_current_db()->drop_table(table_name);
  return rc;
}
```

在 db.cpp 中，实现 drop_table 接口

src > observer > storage > db > db.cpp

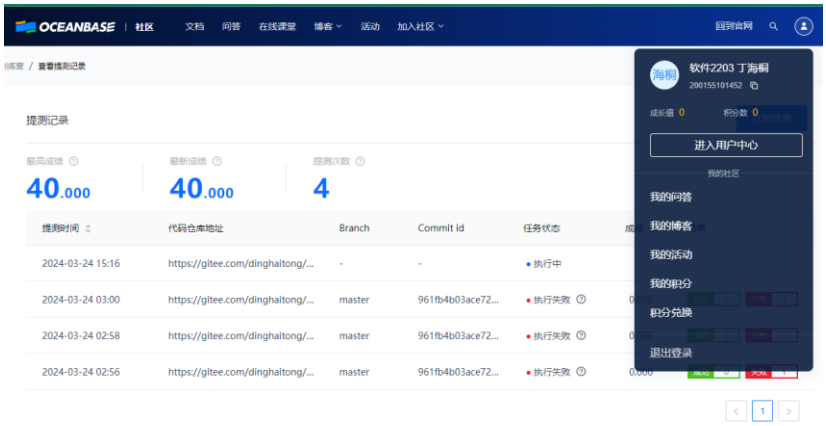
```cpp
RC Db::drop_table(const char* table_name)
{
    auto it = opened_tables_.find(table_name);
    if (it == opened_tables_.end())
    {
        return RC::SCHEMA_DB_NOT_EXIST; // 找不到表，返回错误
    }
    Table* table = it->second;
    RC rc = table->destroy(path_.c_str()); // 让表自己销毁资源
    if(rc != RC::SUCCESS) return rc;

    opened_tables_.erase(it); // 删除成功的话，从表list中将它删除
    delete table;
    return RC::SUCCESS;
}
```

table.cpp 中清理文件和相关数据

src > observer > storage > table > table.cpp

```cpp

RC Table::destroy(const char* dir){
  RC rc = sync();
  if(rc != RC::SUCCESS) return rc;
  std::string path = table_meta_file(dir,name());
  std::string path_data = table_data_file(dir,name());

  if(unlink(path.c_str()) != 0){
    LOG_ERROR("Remove meta file %s failed!, errno:%d",path, errno);
    return RC::INTERNAL;
  }

  if(unlink(path_data.c_str()) != 0){
    LOG_ERROR("Remove data file %s failed! errno:%d",path_data, errno);
    return RC::INTERNAL;
  }

  const int index_num = table_meta_.index_num();
  for(int i = 0;i < index_num;i++){
    ((BplusTreeIndex*)indexes_[i])->close();
    const IndexMeta* index_meta = table_meta_.index(i);
    std::string path_index = table_index_file(dir,name(),index_meta->name());
    if(unlink(path_index.c_str()) != 0){
      LOG_ERROR("Remove index file %s failed! errno:%d",path_index,errno);
      return RC::INTERNAL;
    }
  }
  return RC::SUCCESS;
}
```

5. 评测成功

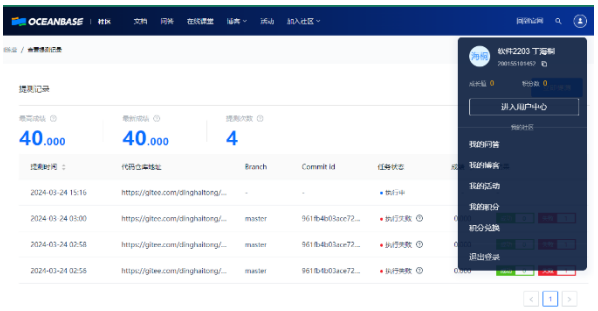


4. 试验完成情况：

题目 1

```
miniob > show tables
Tables_in_SYS

miniob > create table demo1(id int, age int);
SUCCESS

miniob > show tables
Tables_in_SYS
demo1

miniob > insert into demo1 values (1, 20);
SUCCESS

miniob > insert into demo1 values (2, 28);
SUCCESS

miniob > select * from demo1;
id | age
1 | 20
2 | 28

miniob > select * frome demo1 where id=2;
SQL_SYNTAX > Failed to parse sql

miniob > select * from demo1 where id=2;
id | age
2 | 28
```

题目 2：



5. 试验总结：（遇到的问题及解决措施，对试验的评价，感想和认识）

遇到的问题：

配置环境：docker 使用不熟练，在学习 docker 上投入了很多时间；运行时要进入 build 目录；看不懂代码，无从下手，只能硬着头皮读 miniob 的官方文档和介绍视频。

评价：

实验设计挺糟糕。如此庞大的一项工程应该留有充足时间准备，特别是在起始阶段，实验计划书应该在刚开学那段时间就发布，而不是在验收前三四天。

实验出发点是好的，结合的项目也很优秀，但是应该提供相应的讲解，纯自学的话耗费的时间过多。