数据库实验五: MINIOB 实验二

一，赛题的基本信息

名称：drop_table

1. 实现删除表(drop table)，清除表相关的资源。
2. 当前MiniOB支持建表与创建索引，但是没有删除表的功能。
3. 在实现此功能时，除了要删除所有与表关联的数据，不仅包括磁盘中的文件，还包括内存中的索引等数据。
4. 删除表的语句为 `drop table table-name`

二，实现思路的简要分析流程

实现drop_table功能删除表，并清理相关内容。

在`create table t`时，会新建一个t.table文件，同时为了存储数据也会新建一个t.data文件存储下来。同时创建索引的时候，也会创建记录索引数据的文件，在删除表时也要一起删除掉。那么删除表，就需要删除t.table文件、t.data文件和关联的索引文件。

同时由于buffer pool的存在，在新建表和插入数据的时候，会写入buffer pool缓存。所以drop table，不仅需要删除文件，也需要清空buffer pool，防止在数据没落盘的时候，再建立同名表，仍然可以查询到数据。

如果建立了索引，比如t_id on t(id)，那么也会新建一个t_id.index文件，也需要删除这个文件。

代码部分：

（1）首先是完成句柄stmt的编写：在src\observer\sql\stmt\中新建了两个文件drop_table_stmt.cpp与drop_table_stmt.h，仿照create_table_stmt.cpp与create_table_stmt.h编写。

drop_table_stmt.cpp：

```
#include "sql/stmt/drop_table_stmt.h"
#include "event/sql_debug.h"

RC DropTableStmt::create(Db *db, const DropTableSqlNode
&drop_table, Stmt *&stmt)
{
  stmt = new DropTableStmt(drop_table.relation_name);
  sql_debug("drop table statement: table name %s",
drop_table.relation_name.c_str());
  return RC::SUCCESS;
}
```

drop_table_stmt.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
 */
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
&create_table, Stmt *&stmt);
```

```
private:
  std::string table_name_;
};
```

（2）在src\observer\sql\stmt\stmt.cpp与stmt.h中添加drop table的相关声明，并且添加相应的头文件等。

```
case SCF_DROP_TABLE: {
    return DropTableStmt::create(db, sql_node.drop_table,
stmt);
}
```

（3）在executor中添加接口，添加两个新文件drop_table_executor.cpp与drop_table_executor.h。并且在command_executor.cpp中添加接口和头文件。

drop_table_executor.cpp：

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

#include "session/session.h"
#include "common/log/log.h"
#include "storage/table/table.h"
#include "sql/stmt/drop_table_stmt.h"
#include "event/sql_event.h"
#include "event/session_event.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;
}
```

drop_table_executor.h:

```
#pragma once

#include "common/rc.h"

class SQLStageEvent;

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

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

command_executor.cpp:

```
case StmtType::DROP_TABLE: {
      DropTableExecutor executor;
      return executor.execute(sql_event);
    } break;
```

(4) 在db.cpp中，实现drop table接口。并且同步修改db.h。

```
// 实现drop_table接口
RC Db::drop_table(const char* table_name)
```

```
{
    auto it = opened_tables_.find(table_name);
    if (it == opened_tables_.end())
    {
        return RC::SCHEMA_TABLE_NOT_EXIST; // 找不到表，要返回错误，测试程序中也会校验这种场景
    }
    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;
}
```

(5) table.cpp中清理文件和相关数据，并且同步修改table.h。

```
// 清理文件和相关数据
RC Table::destroy(const char* dir) {
    RC rc = sync();//刷新所有脏页

    if(rc != RC::SUCCESS) return rc;

    std::string path = table_meta_file(dir, name());
    if(unlink(path.c_str()) != 0) {
        LOG_ERROR("Failed to remove meta file=%s, errno=%d", path.c_str(), errno);
        return RC::GENERIC_ERROR;
    }

    std::string data_file = table_data_file(dir, name());//std::string(dir) + "/" + name() + TABLE_DATA_SUFFIX;
    if(unlink(data_file.c_str()) != 0) { // 删除描述表元数据的文件
        LOG_ERROR("Failed to remove data file=%s, errno=%d", data_file.c_str(), errno);
```

```
            return RC::GENERIC_ERROR;
        }

        const int index_num = table_meta_.index_num();
        for (int i = 0; i < index_num; i++) {  // 清理所有的索引
相关文件数据与索引元数据
            ((BplusTreeIndex*)indexes_[i])->close();
            const IndexMeta* index_meta =
table_meta_.index(i);
            std::string index_file = table_index_file(dir,
name(), index_meta->name());
            if(unlink(index_file.c_str()) != 0) {
                LOG_ERROR("Failed to remove index file=%s,
errno=%d", index_file.c_str(), errno);
                return RC::GENERIC_ERROR;
            }
        }
        return RC::SUCCESS;
}
```

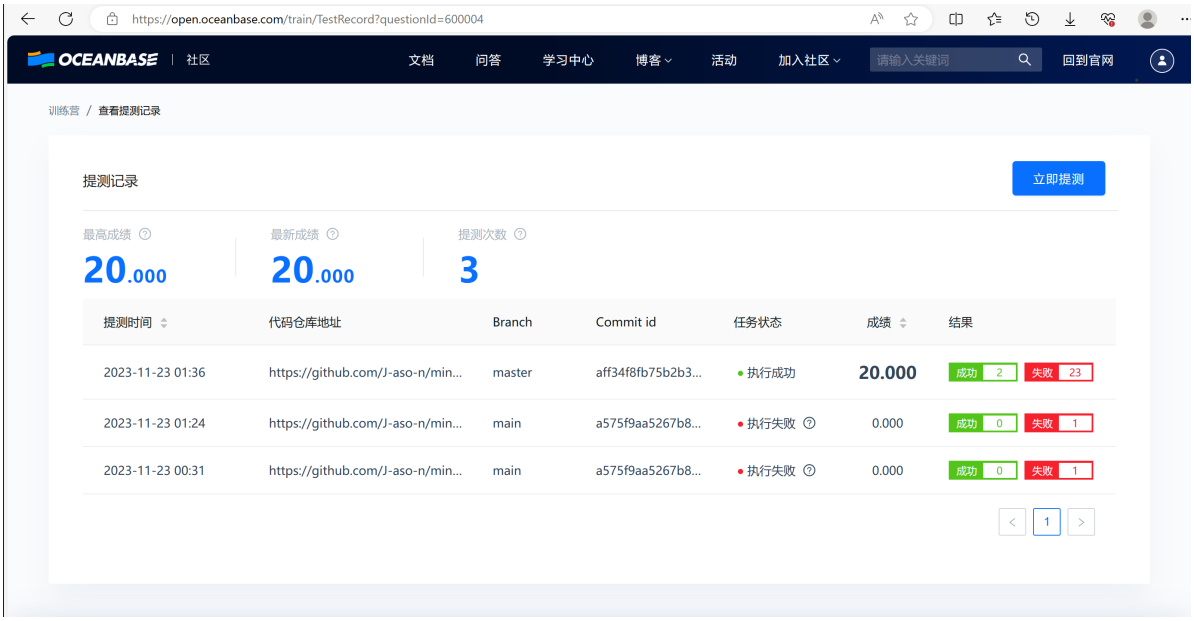
（6）在rc.h中添加一些有关错误的相关声明，做一些修改。完成代码部分。

提交部分：

将文件夹上传至docker容器，进入 miniob 目录，使用 bash build.sh --make -j4 进行编译。

编译成功后上传至github仓库，链接到oceanbase大赛提测通道进行提测。

三，和执行测试样例示例 的实验结果截图 (或是大赛提测通道的通过结果截图)



仓库链接：https://github.com/J-aso-n/miniob_exercise1.git