《数据库系统原理》实验报告（五）					
题目：MINIOB 实验二					
学号	2151140	姓名	王谦	日期	2023.11.12

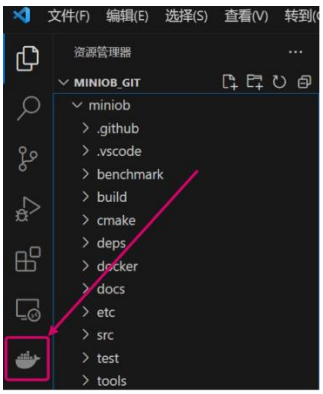
实验环境：Docker + miniob + VSCode

题目及要求：

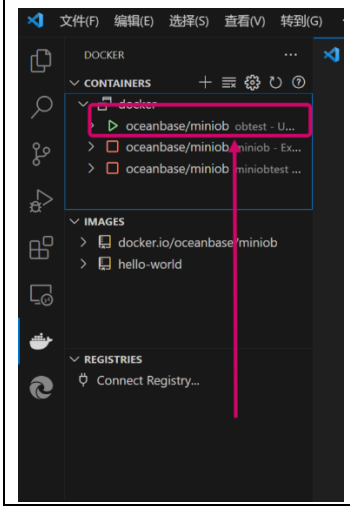
> drop-table
>
> 状态：未通过　通过率：84.7%　难度：简单　题目总分：10
>
> 1. 实现删除表(drop table)，清除表相关的资源。
> 2. 当前MiniOB支持建表与创建索引，但是没有删除表的功能。
> 3. 在实现此功能时，除了要删除所有与表关联的数据，不仅包括磁盘中的文件，还包括内存中的索引等数据。
> 4. 删除表的语句为 `drop table table-name`

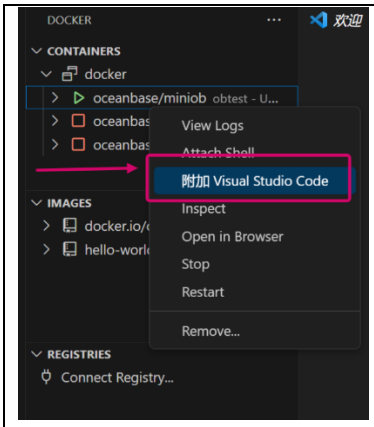
环境配置：

Vscode 环境配置：

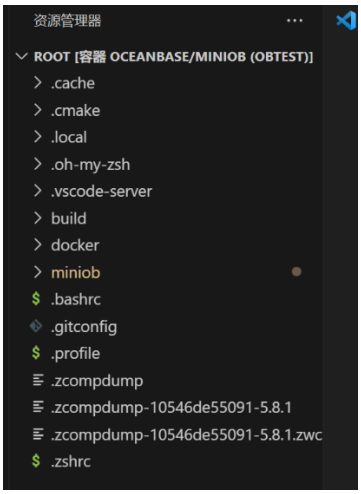


右键点击：

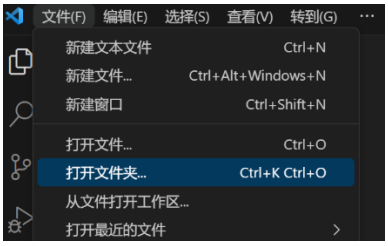




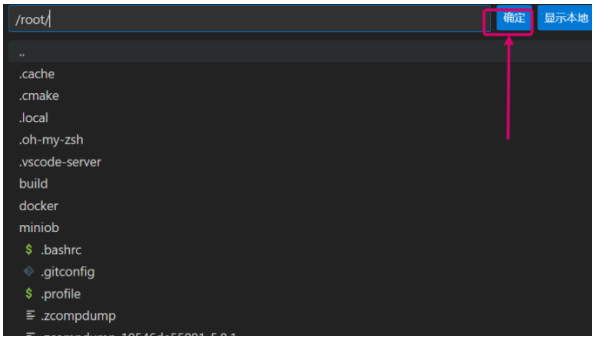
如果右边栏直接这样的话，说明 ok 了：



如果是空的话，我们还需要点击“文件”——“打开文件夹”：

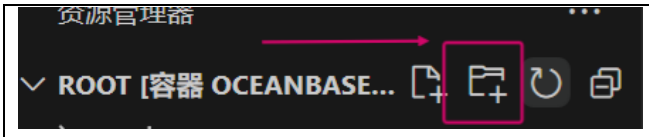


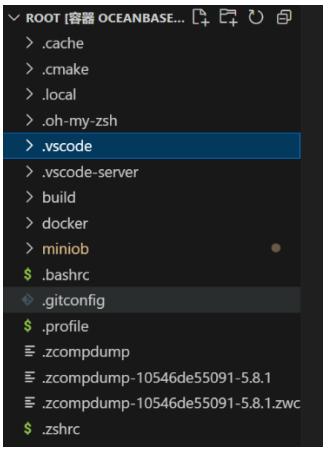
弹出的窗口应该会默认是/root/，点击确定即可。



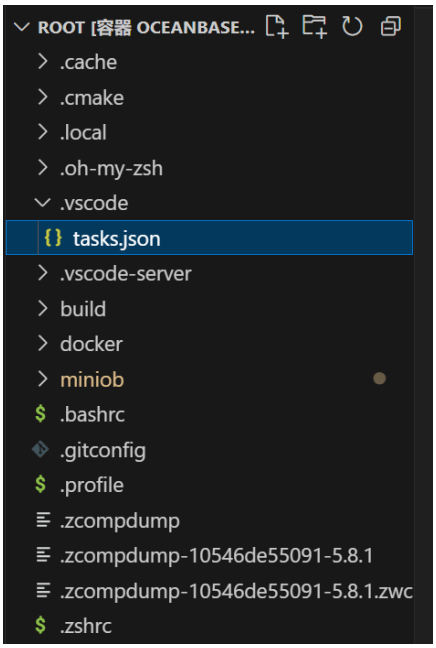
编译与链接：

新建一个.vscode 文件夹：



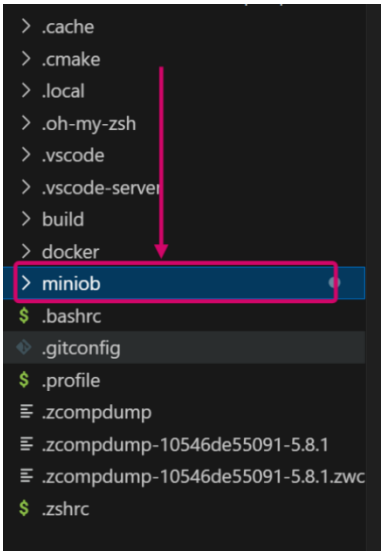


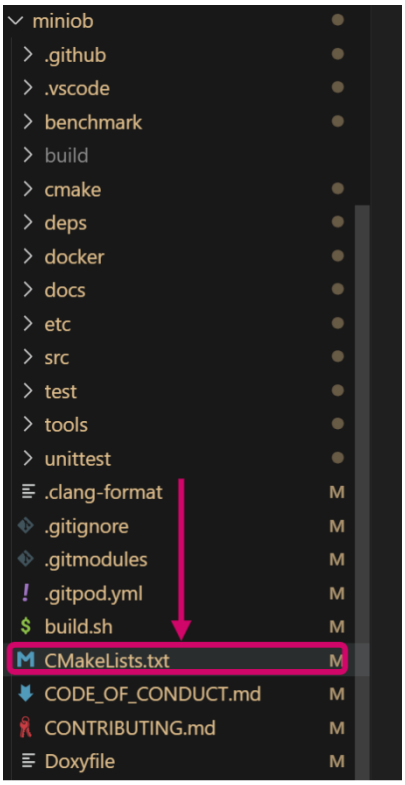
在.vscode 文件夹中新建 tasks.json 文件



在 tasks.json 中输入：

接下来，我们找到 miniob 下的 CMakelist.txt



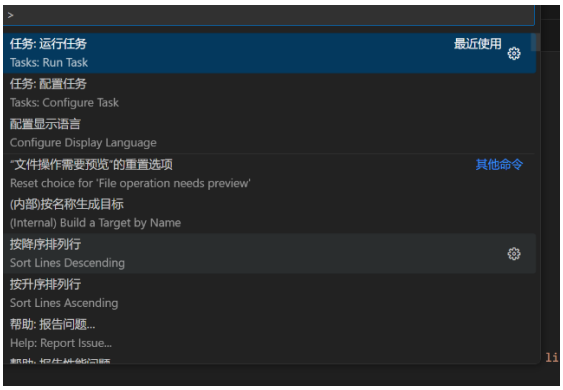


在其开头添加一句：

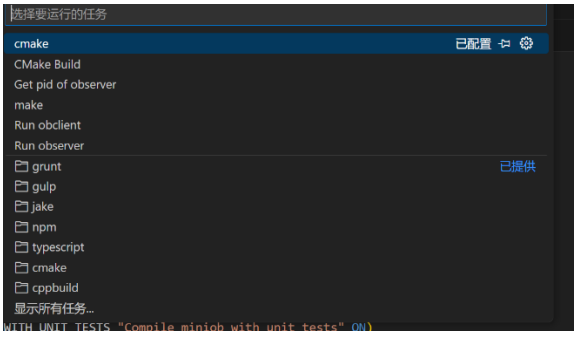
add_definitions("-Wall -g")

这句话可以防止 gdb 调试断点失效的问题。仍然 ctrl+s 保存。

然后使用 ctrl+shift+p 调出窗口，搜索“任务”，找到运行任务：



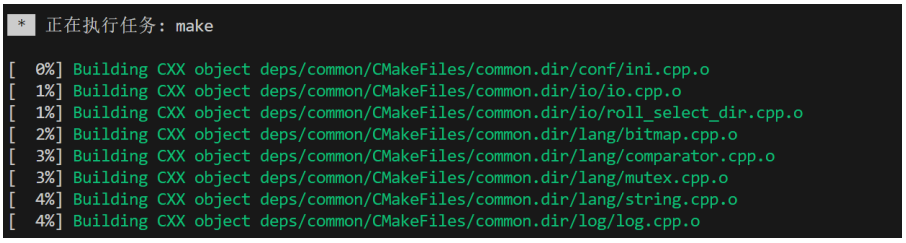
点击 cmake 进行编译；



选择“继续而不扫描任务输出”：

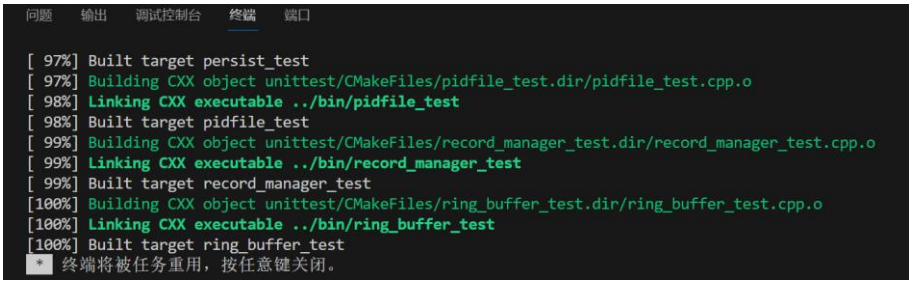
如果终端中显示这样说明成功。

接着跟上面差不多，“任务：运行任务”——“make”——“继续而不扫描任务输出“，终端输出如下说明成功：

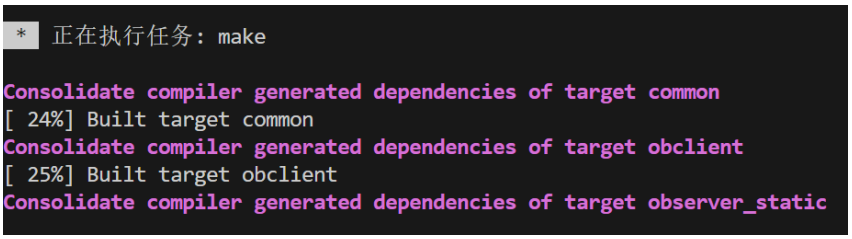


这一步需要等待较长的时间。

终端输出如下说明成功：



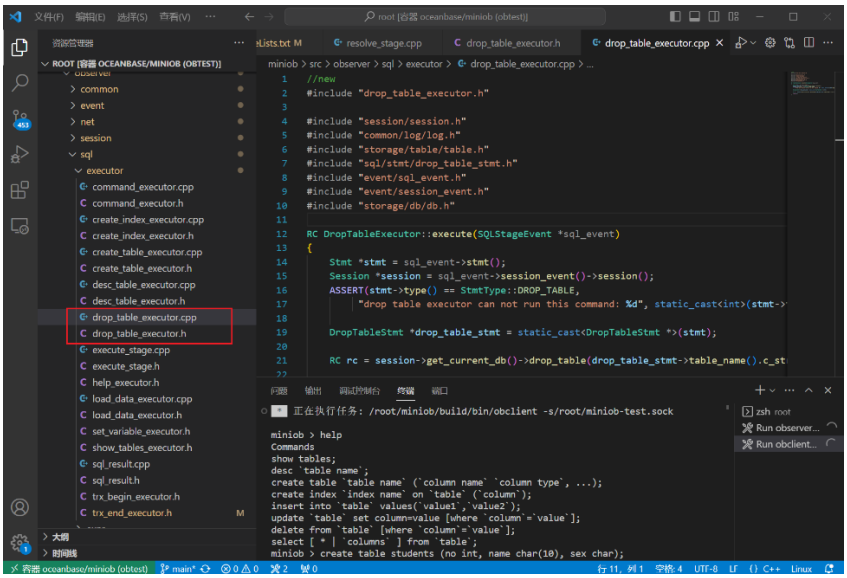
最后，“任务：运行任务”——“cmake build”——“继续而不扫描任务输出“：

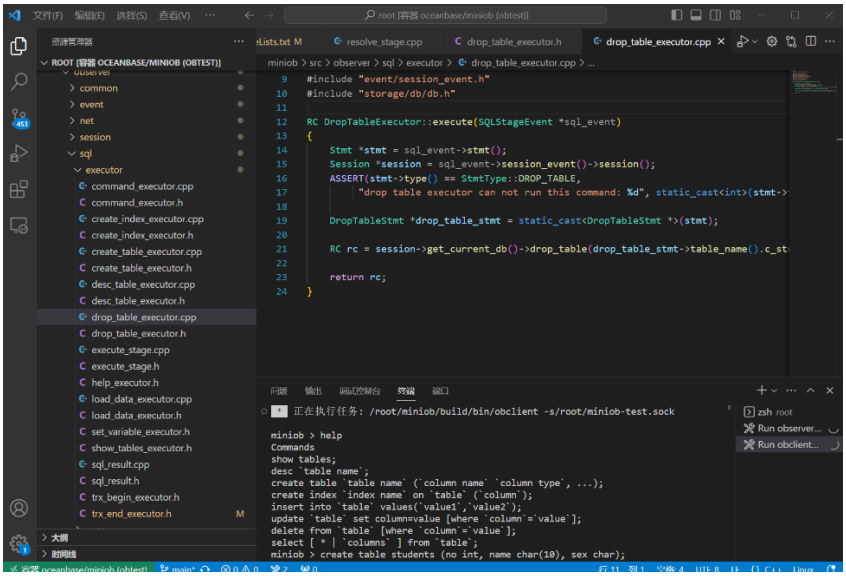


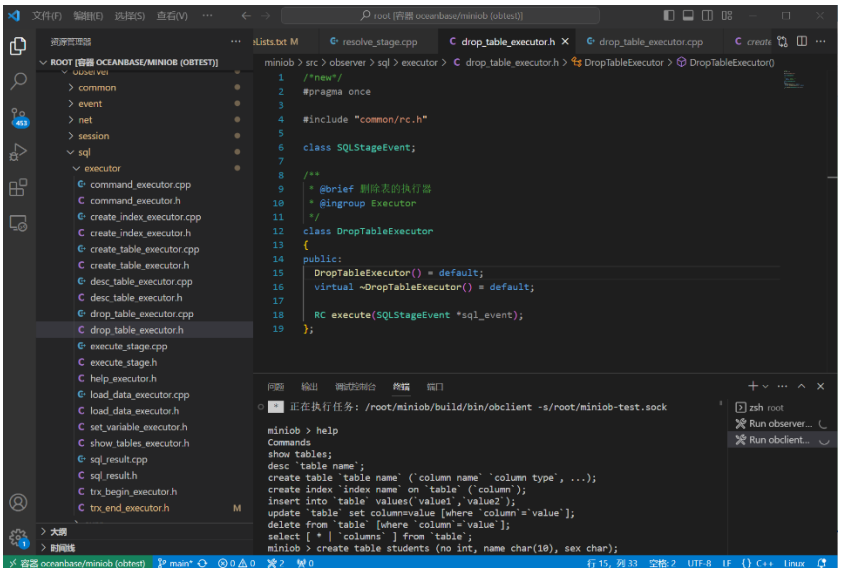
终端输出如下说明成功：

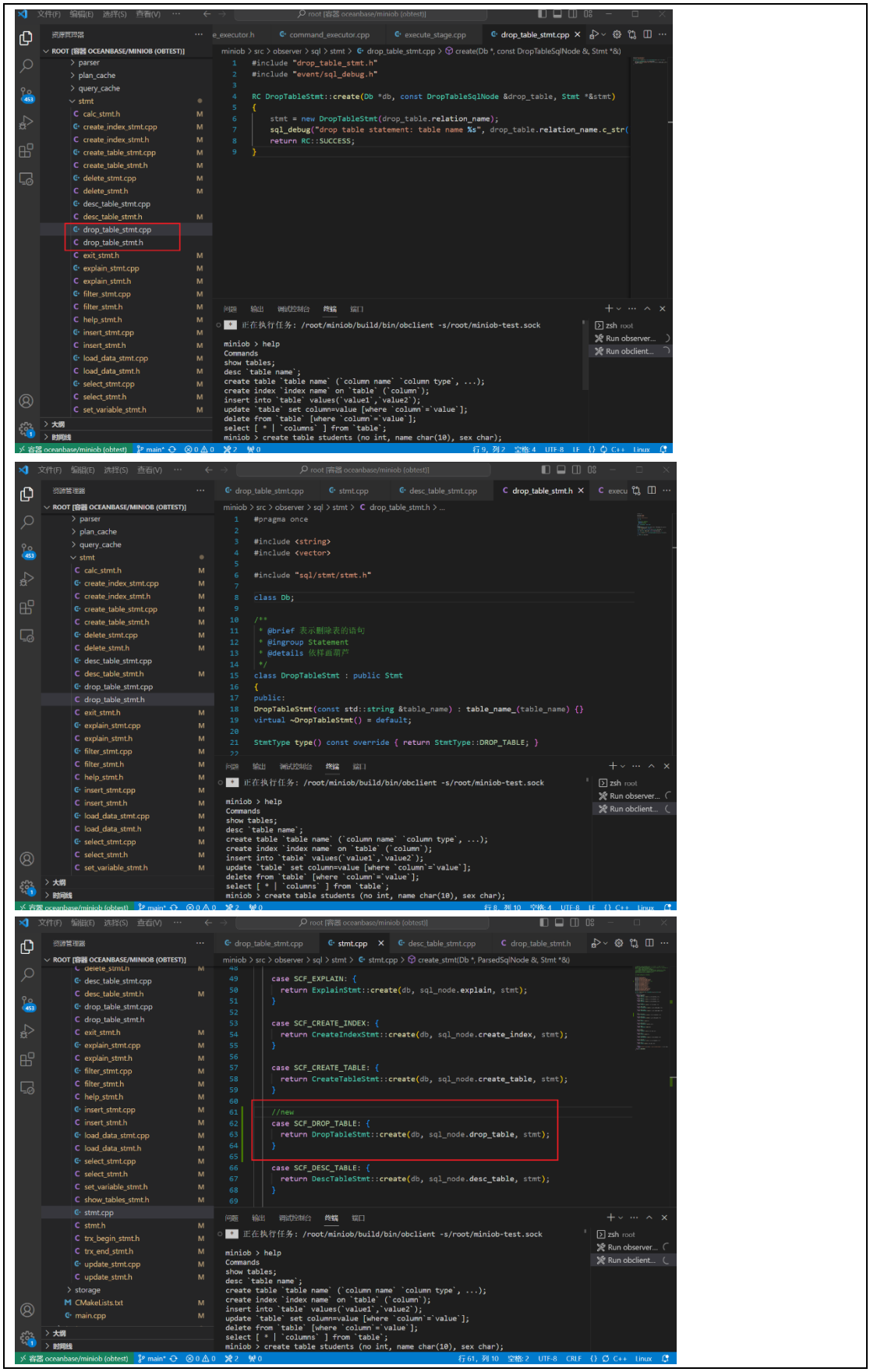
这样，环境配置完成。

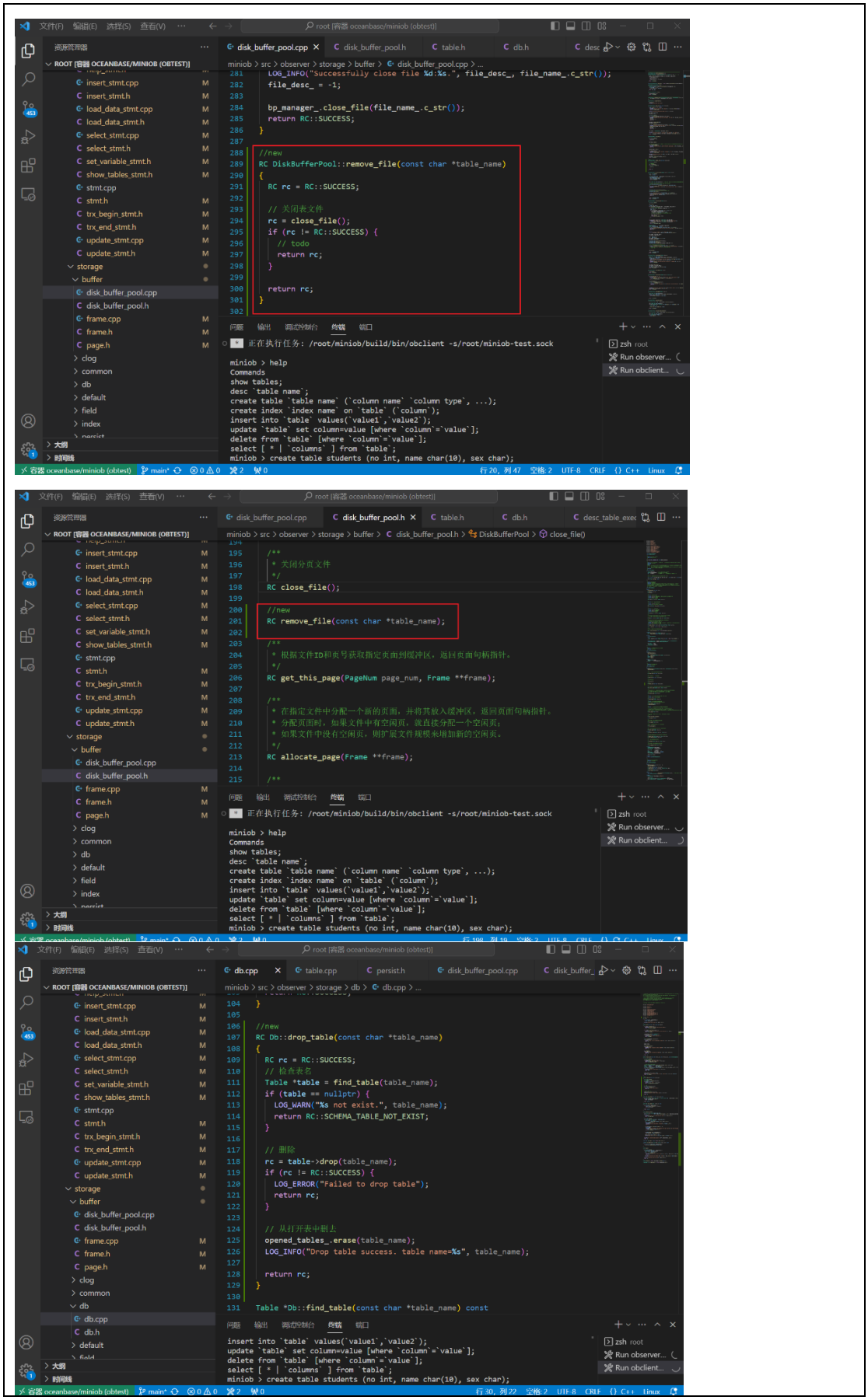
Drop table 功能的实现：

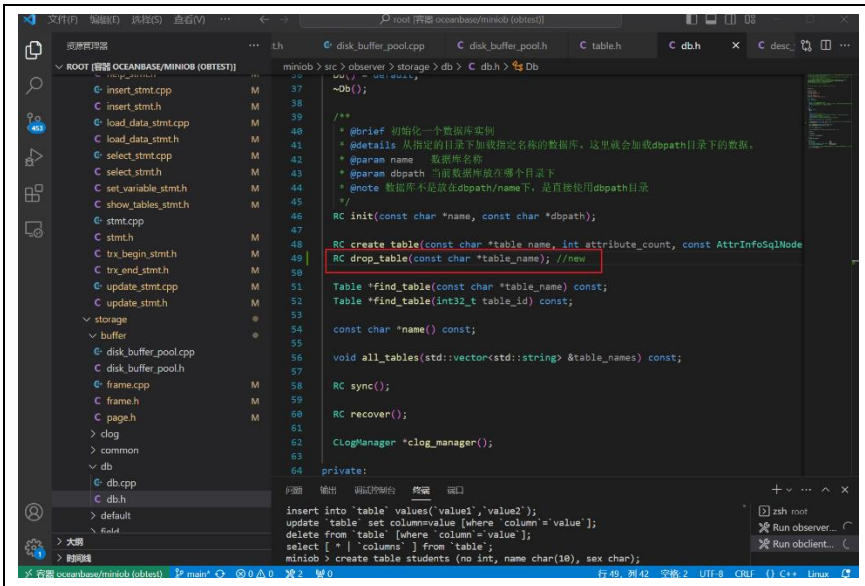







#include "drop_table_stmt.h"
#include "event/sql_debug.h"
RC DropTableStmt::create(Db *db, const DropTableSqlNode &drop_table, Stmt *&stmt)
stmt = new DropTableStmt(drop_table.relation_name);
sql_debug("drop table statement: table name %s", drop_table.relation_name.c_str(
return RC::SUCCESS;
正在执行任务: /root/miniob/build/bin/obclient -s/root/miniob-test.sock
miniob > help
Commands
show tables;
desc `table name`;
create table `table name` (`column name` `column type`, ...);
create index `index name` on `table` (`column`);
insert into `table` values(`value1`,`value2`);
update `table` set column=value [where `column`=`value`];
delete from `table` [where `column`=`value`];
select [* | `columns`] from `table`;
miniob > create table students (no int, name char(10), sex char);
#pragma once
#include <string>
#include <vector>
#include "sql/stmt/stmt.h"
class Db;
/**
* @brief 表示删除表的语句
* @ingroup Statement
* @details 依样画葫芦
*/
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; }
正在执行任务: /root/miniob/build/bin/obclient -s/root/miniob-test.sock
miniob > help
Commands
show tables;
desc `table name`;
create table `table name` (`column name` `column type`, ...);
create index `index name` on `table` (`column`);
insert into `table` values(`value1`,`value2`);
update `table` set column=value [where `column`=`value`];
delete from `table` [where `column`=`value`];
select [* | `columns`] from `table`;
miniob > create table students (no int, name char(10), sex char);
case SCF_EXPLAIN: {
return ExplainStmt::create(db, sql_node.explain, stmt);
}
case SCF_CREATE_INDEX: {
return CreateIndexStmt::create(db, sql_node.create_index, stmt);
}
case SCF_CREATE_TABLE: {
return CreateTableStmt::create(db, sql_node.create_table, stmt);
}
//new
case SCF_DROP_TABLE: {
return DropTableStmt::create(db, sql_node.drop_table, stmt);
}
case SCF_DESC_TABLE: {
return DescTableStmt::create(db, sql_node.desc_table, stmt);
}
正在执行任务: /root/miniob/build/bin/obclient -s/root/miniob-test.sock
miniob > help
Commands
show tables;
desc `table name`;
create table `table name` (`column name` `column type`, ...);
create index `index name` on `table` (`column`);
insert into `table` values(`value1`,`value2`);
update `table` set column=value [where `column`=`value`];
delete from `table` [where `column`=`value`];
select [* | `columns`] from `table`;
miniob > create table students (no int, name char(10), sex char);


disk_buffer_pool.cpp
disk_buffer_pool.h
table.h
db.h
miniob > src > observer > storage > buffer > disk_buffer_pool.cpp > ...
LOG_INFO("Successfully close file %d:%s.", file_desc_, file_name_.c_str());
file_desc_ = -1;
bp_manager_.close_file(file_name_.c_str());
return RC::SUCCESS;
}
//new
RC DiskBufferPool::remove_file(const char *table_name)
{
RC rc = RC::SUCCESS;
// 关闭表文件
rc = close_file();
if (rc != RC::SUCCESS) {
// todo
return rc;
}
return rc;
}
正在执行任务: /root/miniob/build/bin/obclient -s/root/miniob-test.sock
miniob > help
Commands
show tables;
desc `table name`;
create table `table name` (`column name` `column type`, ...);
create index `index name` on `table` (`column`);
insert into `table` values(`value1`,`value2`);
update `table` set column=value [where `column`=`value`];
delete from `table` [where `column`=`value`];
select [* | `columns`] from `table`;
miniob > create table students (no int, name char(10), sex char);
disk_buffer_pool.h
miniob > src > observer > storage > buffer > disk_buffer_pool.h > DiskBufferPool > close_file()
/**
* 关闭分页文件
*/
RC close_file();
//new
RC remove_file(const char *table_name);
/**
* 根据文件ID和页号获取指定页面到缓冲区，返回页面句柄指针。
*/
RC get_this_page(PageNum page_num, Frame **frame);
/**
* 在指定文件中分配一个新的页面，并将其放入缓冲区，返回页面句柄指针。
* 分配页面时，如果文件中有空闲页，就直接分配一个空闲页；
* 如果文件中没有空闲页，则扩展文件规模来增加新的空闲页。
*/
RC allocate_page(Frame **frame);
/**
db.cpp
table.cpp
persist.h
disk_buffer_pool.cpp
miniob > src > observer > storage > db > db.cpp > ...
}
//new
RC Db::drop_table(const char *table_name)
{
RC rc = RC::SUCCESS;
// 检查表名
Table *table = find_table(table_name);
if (table == nullptr) {
LOG_WARN("%s not exist.", table_name);
return RC::SCHEMA_TABLE_NOT_EXIST;
}
// 删除
rc = table->drop(table_name);
if (rc != RC::SUCCESS) {
LOG_ERROR("Failed to drop table");
return rc;
}
// 从打开表中删去
opened_tables_.erase(table_name);
LOG_INFO("Drop table success. table name=%s", table_name);
return rc;
}
Table *Db::find_table(const char *table_name) const


db.h
/**
* @brief 初始化一个数据库实例
* @details 从指定的目录下加载指定名称的数据库。这里就会加载dbpath目录下的数据。
* @param name 数据库名称
* @param dbpath 当前数据库放在哪个目录下
* @note 数据库不是放在dbpath/name下，是直接使用dbpath目录
*/
RC init(const char *name, const char *dbpath);
RC create_table(const char *table_name, int attribute_count, const AttrInfoSqlNode
RC drop_table(const char *table_name); //new
Table *find_table(const char *table_name) const;
Table *find_table(int32_t table_id) const;
const char *name() const;
void all_tables(std::vector<std::string> &table_names) const;
RC sync();
RC recover();
CLogManager *clog_manager();
private:
insert into `table` values(`value1`,`value2`);
update `table` set column=value [where `column`=`value`];
delete from `table` [where `column`=`value`];
select [* | `columns`] from `table`;
miniob > create table students (no int, name char(10), sex char);

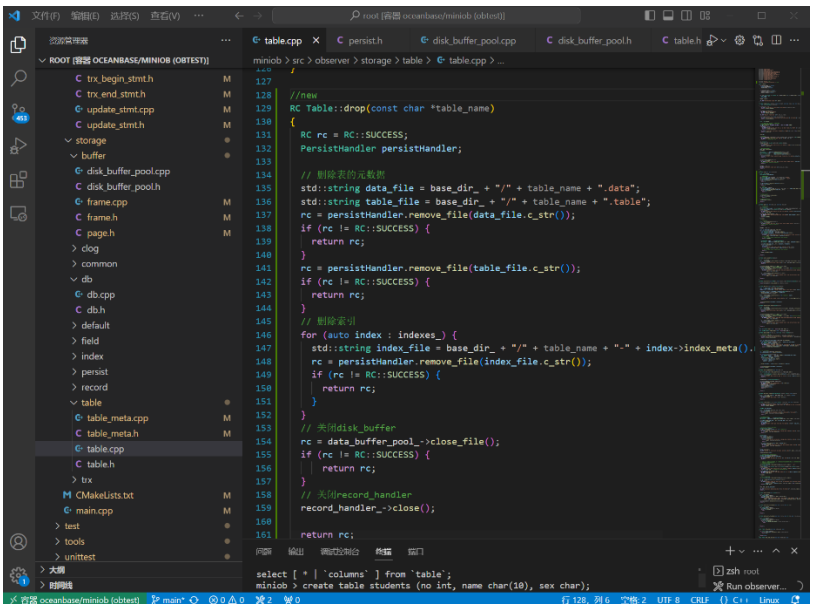


table.cpp
//new
RC Table::drop(const char *table_name)
{
RC rc = RC::SUCCESS;
PersistHandler persistHandler;
// 删除表的元数据
std::string data_file = base_dir_ + "/" + table_name + ".data";
std::string table_file = base_dir_ + "/" + table_name + ".table";
rc = persistHandler.remove_file(data_file.c_str());
if (rc != RC::SUCCESS) {
return rc;
}
rc = persistHandler.remove_file(table_file.c_str());
if (rc != RC::SUCCESS) {
return rc;
}
// 删除索引
for (auto index : indexes_) {
std::string index_file = base_dir_ + "/" + table_name + "-" + index->index_meta().
rc = persistHandler.remove_file(index_file.c_str());
if (rc != RC::SUCCESS) {
return rc;
}
}
// 关闭disk_buffer
rc = data_buffer_pool_->close_file();
if (rc != RC::SUCCESS) {
return rc;
}
// 关闭record_handler
record_handler_->close();
return rc;
select [* | `columns`] from `table`;
miniob > create table students (no int, name char(10), sex char);

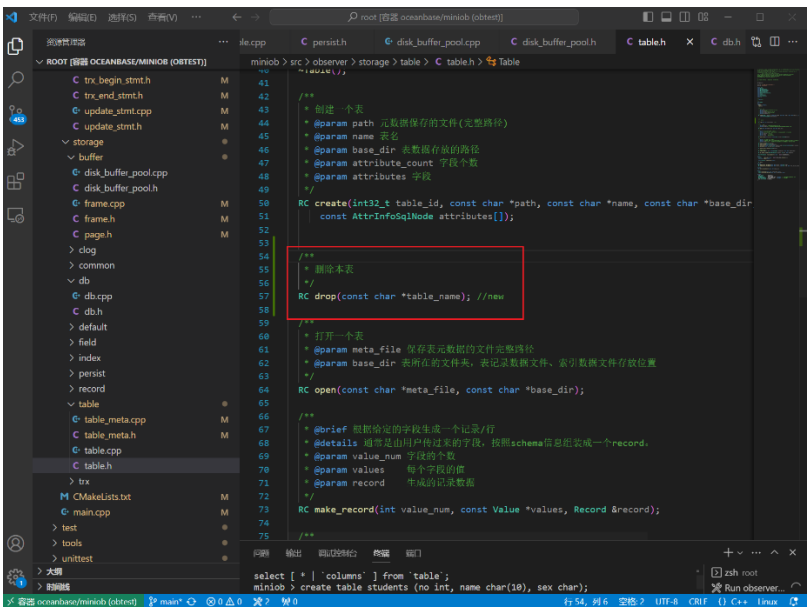
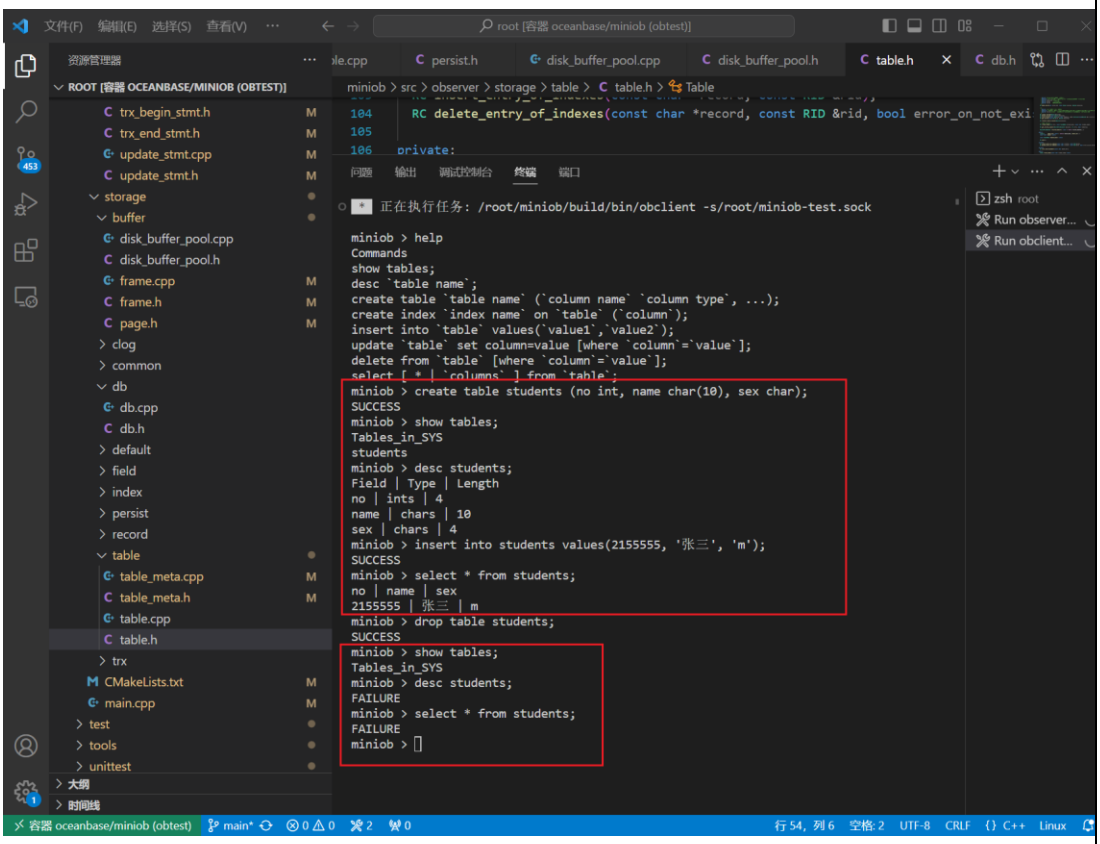


table.h
/**
* 创建一个表
* @param path 元数据保存的文件(完整路径)
* @param name 表名
* @param base_dir 表数据存放的路径
* @param attribute_count 字段个数
* @param attributes 字段
*/
RC create(int32_t table_id, const char *path, const char *name, const char *base_dir
const AttrInfoSqlNode attributes[]);
/**
* 删除本表
*/
RC drop(const char *table_name); //new
/**
* 打开一个表
* @param meta_file 保存表元数据的文件完整路径
* @param base_dir 表所在的文件夹，表记录数据文件、索引数据文件存放位置
*/
RC open(const char *meta_file, const char *base_dir);
/**
* @brief 根据给定的字段生成一个记录/行
* @details 通常是由用户传过来的字段，按照schema信息组装成一个record。
* @param value_num 字段的个数
* @param values 每个字段的值
* @param record 生成的记录数据
*/
RC make_record(int value_num, const Value *values, Record &record);
select [* | `columns`] from `table`;
miniob > create table students (no int, name char(10), sex char);

验证功能实现：


miniob > help
Commands
show tables;
desc `table name`;
create table `table name` (`column name` `column type`, ...);
create index `index name` on `table` (`column`);
insert into `table` values(`value1`,`value2`);
update `table` set column=value [where `column`=`value`];
delete from `table` [where `column`=`value`];
select [* | `columns`] from `table`;
miniob > create table students (no int, name char(10), sex char);
SUCCESS
miniob > show tables;
Tables_in_SYS
students
miniob > desc students;
Field | Type | Length
no | ints | 4
name | chars | 10
sex | chars | 4
miniob > insert into students values(2155555, '张三', 'm');
SUCCESS
miniob > select * from students;
no | name | sex
2155555 | 张三 | m
miniob > drop table students;
SUCCESS
miniob > show tables;
Tables_in_SYS
miniob > desc students;
FAILURE
miniob > select * from students;
FAILURE
miniob >
正在执行任务: /root/miniob/build/bin/obclient -s/root/miniob-test.sock