《数据库系统原理》实验报告（5）

题目：SQL 综合实验

学号	2352018	姓名	刘彦	日期	2025.5.8

实验环境：

Docker-desktop 4.41.2

MiniOB

实验步骤及结果截图：

⑴在 Docker 中建立 miniob 环境（参考第 0 章的教程 PPT 和第三节的内容）

git clone https://github.com/oceanbase/miniob.git

```
# git clone https://github.com/oceanbase/miniob.git
Cloning into 'miniob'...
remote: Enumerating objects: 7183, done.
remote: Counting objects: 100% (236/236), done.
remote: Compressing objects: 100% (163/163), done.
remote: Total 7183 (delta 100), reused 75 (delta 73), pack-reused 6947 (from 2)
Receiving objects: 100% (7183/7183), 43.09 MiB | 2.40 MiB/s, done.
Resolving deltas: 100% (3893/3893), done.
# cd miniob
# ls
benchmark  build.sh  cmake  CMakeLists.txt  CODE_OF_CONDUCT.md  CONTRIBUTING.md  deps  docker  docs  etc  License  NOTICE  README.md  src  test  tools  unittest
```

cd miniob

bash build.sh --make -j4

```
# bash build.sh --make -j4
THIRD_PARTY_INSTALL_PREFIX is /root/miniob/deps/3rd/usr/local
build.sh --make -j4
Build type: debug
create soft link for build_debug, linked by directory named build
cmake -DCMAKE_EXPORT_COMPILE_COMMANDS=1 --log-level=STATUS /root/miniob -DCMAKE_BUILD_TYPE=debug
-- The C compiler identification is GNU 13.2.0
-- The CXX compiler identification is GNU 13.2.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- This is Project source dir /root/miniob
-- This is PROJECT_BINARY_DIR dir /root/miniob/build_debug
-- Using build type: debug
-- HOME dir: /root
-- This is UNIX
-- CMAKE_CXX_COMPILER_ID is GNU
-- Instrumenting with Address Sanitizer
-- CMAKE_INSTALL_PREFIX has been set as /usr/local
-- Install target dir is /usr/local
```

（中间省略编译过程）

```
[ 99%] Linking CXX executable ../../bin/parser_test
[ 99%] Built target parser_test
[ 99%] Building CXX object unittest/observer/CMakeFiles/record_manager_test.dir/record_manager_test.cpp.o
[ 99%] Linking CXX executable ../../bin/persist_test
[ 99%] Built target persist_test
[ 99%] Building CXX object unittest/observer/CMakeFiles/ring_buffer_test.dir/ring_buffer_test.cpp.o
[ 99%] Linking CXX executable ../../bin/mvcc_trx_log_test
[ 99%] Built target mvcc_trx_log_test
[ 99%] Linking CXX executable ../../bin/pax_storage_test
[100%] Linking CXX executable ../../bin/ring_buffer_test
[100%] Built target ring_buffer_test
[100%] Built target pax_storage_test
[100%] Linking CXX executable ../../bin/record_manager_test
[100%] Built target record_manager_test
```

cd build

./bin/observer -s miniob.sock -f ../etc/observer.ini &

```
# ./bin/observer -s miniob.sock -f ../etc/observer.ini &
#
Welcome to the OceanBase database implementation course.

Copyright (c) 2021 OceanBase and/or its affiliates.

Learn more about OceanBase at https://github.com/oceanbase/oceanbase
Learn more about MiniOB at https://github.com/oceanbase/miniob

Successfully load ../etc/observer.ini

# ./bin/obclient -s miniob.sock

Welcome to the OceanBase database implementation course.

Copyright (c) 2021 OceanBase and/or its affiliates.

Learn more about OceanBase at https://github.com/oceanbase/oceanbase
Learn more about MiniOB at https://github.com/oceanbase/miniob

miniob >
```

./bin/obclient -s miniob.sock

```
# cd build
#  ./bin/observer -s miniob.sock -f ../etc/observer.ini &
#
Welcome to the OceanBase database implementation course.

Copyright (c) 2021 OceanBase and/or its affiliates.

Learn more about OceanBase at https://github.com/oceanbase/oceanbase
Learn more about MiniOB at https://github.com/oceanbase/miniob

Successfully load ../etc/observer.ini
```

⑵创建一张表，包括学号，姓名，绩点，学分

```
create table Student (
    No int,
    Name char(10),
    Grade float,
    Credit float,
    primary key (No)
);
```

```
miniob > create table Student (
    No int,
    Name char(10),
    Grade float,
    Credit float,
    primary key (No)
);
SUCCESS
```

⑶向该表插入几行数据，其中需要包含一条包含个人学号、姓名的数据

insert into Student values (235001, '张三', 4.8, 120);

insert into Student values (235002, '李四', 4.7, 124.5);

insert into Student values (235003, '王五', 4.2, 99.5);

insert into Student values (2352018, '刘彦', 4.9, 110);

```
miniob > insert into Student values (235001, '张三', 4.8, 120);
SUCCESS
miniob > insert into Student values (235002, '李四', 4.7, 124.5);
SUCCESS
miniob > insert into Student values (235003, '王五', 4.2, 99.5);
SUCCESS
miniob > insert into Student values (2352018, '刘彦', 4.9, 110);
SUCCESS
```

⑷使用 select 语句展示学号，绩点，学分

select No, Grade, Credit from Student;

```
miniob > select No, Grade, Credit
from Student;
No | Grade | Credit
235001 | 4.8 | 120
235002 | 4.7 | 124.5
235003 | 4.2 | 99.5
2352018 | 4.9 | 110
```

⑸尝试修改指定行的绩点或学分如下表所示，能否成功？为什么？

UPDATE Student

SET Grade = 4.82, Credit = 120

WHERE No = 2350001;

UPDATE Student

SET Grade = 4.65, Credit = 124.5

WHERE No = 2350002;

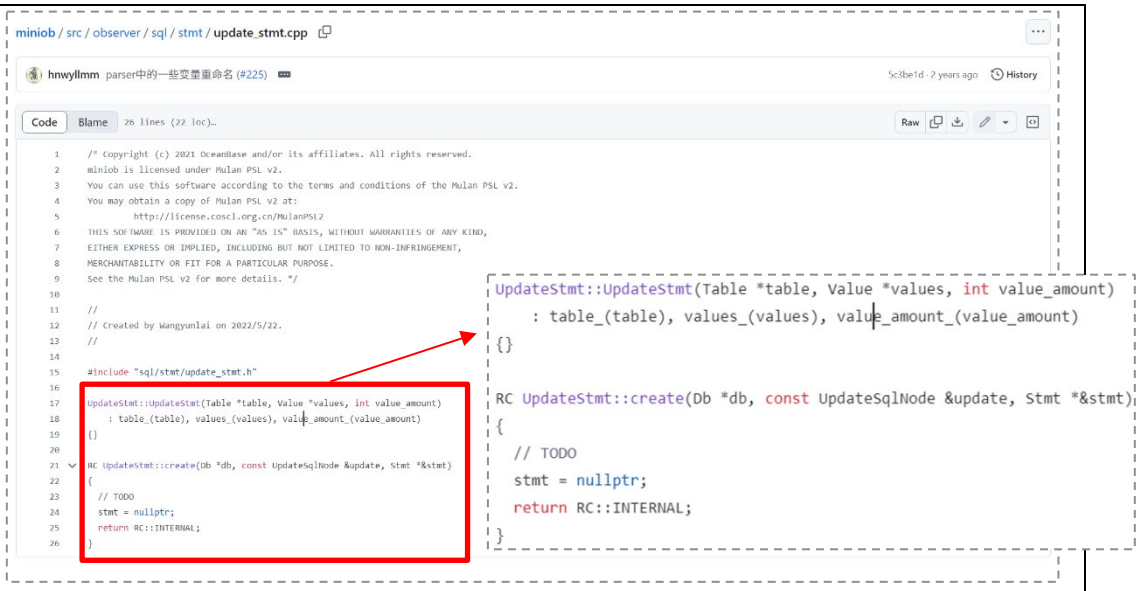
UPDATE Student

SET Grade = 4.2, Credit = 103.5

WHERE No = 2350003;

```
miniob > UPDATE Student
SET Grade = 4.82, Credit = 120
WHERE No = 2350001;
FAILURE
miniob > UPDATE Student
SET Grade = 4.65, Credit = 124.5
WHERE No = 2350002;
FAILURE
miniob > UPDATE Student
SET Grade = 4.2, Credit = 103.5
WHERE No = 2350003;
FAILURE
```

修改不能成功，经检查源码，源码 update 相关函数为空值。



⑹删除张三的记录

delete from Student where No = 2350001;

```
miniob > delete from Student where No = 235001;
SUCCESS
miniob > select No, Grade, Credit from Student;
No | Grade | Credit
235002 | 4.7 | 124.5
235003 | 4.2 | 99.5
2352018 | 4.9 | 110
```

⑺使用 select 语句展示你的学号

select No

from Student

where Name = '刘彦';

```
miniob > select No
from Student
where Name = '刘彦';
No
2352018
```

⑻对 miniob 源码进行阅读，主要选取一个功能（如 create table、insert、delete 等）进行分析理解，做简要报告（不超过两页）

选取 delete 功能进行分析理解，代码地址为 https://github.com/oceanbase/miniob/blob/main/src/observer/sql/stmt/delete_stmt.cpp。

DeleteStmt 类定义

构造函数：初始化 DeleteStmt 对象，接收一个 Table 指针（目标表）和一个 FilterStmt 指针（WHERE 条件的过滤语句）。

成员变量：

- table_：指向目标表的指针。
- filter_stmt_：指向过滤语句的指针，用于指定删除的行。

析构函数：释放 filter_stmt_指向的内存，避免内存泄漏。逻辑是检查 filter_stmt_是否非空，若非空则删除并置空指针。

```
DeleteStmt::~DeleteStmt()
{
  if (nullptr != filter_stmt_) {
    delete filter_stmt_;
    filter_stmt_ = nullptr;
  }
}
```

静态方法 create：根据输入的 DeleteSqlNode（DELETE 语句的语法树节点）创建 DeleteStmt 对象。

其实现的逻辑如下：

①参数校验

检查 db 和 table_name 是否为空，若为空则记录警告日志并返回 RC::INVALID_ARGUMENT。

```
const char *table_name = delete_sql.relation_name.c_str();
if (nullptr == db || nullptr == table_name) {
  LOG_WARN("invalid argument. db=%p, table_name=%p", db, table_name);
  return RC::INVALID_ARGUMENT;
}
```

②检查表是否存在

调用 db->find_table 查找目标表。若表不存在，记录警告日志并返回 RC::SCHEMA_TABLE_NOT_EXIST。

```
// check whether the table exists
Table *table = db->find_table(table_name);
if (nullptr == table) {
  LOG_WARN("no such table. db=%s, table_name=%s", db->name(), table_name);
  return RC::SCHEMA_TABLE_NOT_EXIST;
}
```

③创建表映射

创建一个 unordered_map，将表名映射到表对象。这里仅插入目标表，用于后续过滤语句的创建。

```
unordered_map<string, Table *> table_map;
table_map.insert(pair<string, Table *>(string(table_name), table));
```

④创建过滤语句

调用 FilterStmt::create 创建过滤语句，基于 delete_sql.conditions（WHERE 条件）。参数包括数据库、目标表、表映射、条件数组及其大小。若创建失败，记录警告日志并返回错误码。

```
FilterStmt *filter_stmt = nullptr;
RC          rc          = FilterStmt::create(
    db, table, &table_map, delete_sql.conditions.data(), static_cast<int>(delete_sql.conditions.size()), filter_stmt);
if (rc != RC::SUCCESS) {
  LOG_WARN("failed to create filter statement. rc=%d:%s", rc, strrc(rc));
  return rc;
}
```

⑤创建 DeleteStmt 对象

创建 DeleteStmt 对象，将表和过滤语句传递给构造函数。返回操作结果（RC::SUCCESS 表示成功）。

```
stmt = new DeleteStmt(table, filter_stmt);
return rc;
```

OceanBase miniob 项目的 DeleteStmt 类实现 SQL DELETE 语句解析，封装目标表和 WHERE 条件

（通过 FilterStmt），create 方法从 DeleteSqlNode 提取表名和条件，验证表存在性并生成 DeleteStmt 对象。代码模块化设计清晰，错误处理完善（返回码和日志），析构函数确保内存释放，但需调用者管理动态分配的 DeleteStmt。

可能的改进点如下：

- 空表名检查：当前仅验证 table_name 是否为 nullptr，未检查空字符串。建议添加 if (table_name [0] == '\0') { return RC::INVALID_ARGUMENT; }以防止空表名导致未定义行为。
- 表映射优化：unordered_map 仅存储目标表，作用有限。若 FilterStmt 不需多表支持，可直接传递 Table 指针，简化接口和代码。
- 异常安全性：使用 new 分配 DeleteStmt，未处理内存不足等异常。
- 资源管理：create 方法中若 FilterStmt::create 失败，未清理潜在的中间资源。建议确保所有失败路径均妥善清理。
- 接口一致性：FilterStmt::create 的表映射参数设计较为通用，但在此场景下显得冗余。建议为单表场景优化接口，减少不必要的复杂性。

出现的问题：

⑴建立表格时出现 SQL 语法解析失败

在建立 table 时，输入和 oceanbase 中相同的语句，出现报错。

```
miniob > CREATE TABLE Student (
    No INT PRIMARY KEY,
    Name CHAR(10),
    Grade FLOAT,
    Credit FLOAT
);
SQL_SYNTAX > Failed to parse sql
```

⑵插入多个值时只能保存首行问题

在表中插入值时如果连续插入多个值，虽然显示成功，但只能成功插入第一个。

```
miniob > insert into Student values
(235001, '张三', 4.8, 120),
(235002, '李四', 4.7, 124.5),
(235003, '王五', 4.2, 99.5),
(2352018, '刘彦', 4.9, 110);
SUCCESS
miniob > select No, Grade, Credit
from Student;
No | Grade | Credit
235001 | 4.8 | 120
```

解决方案：

⑴使建立表格时出现 SQL 语法解析失败的解决

一开始认为 MiniOB 不能使用大写，改为小写后依然失败。经检查，在 MiniOB 中，primary key 语句只能单列，不能跟在定义变量语句后面，改成单列后成功解决。

⑵插入多个值时只能保存首行问题的解决

在 MiniOB 中，插值操作只能一步一步进行，一条一条插入，这样就可以查到插入的 4 条语句。