实验6：并发控制实现

邹兆年，哈尔滨工业大学计算学部，znzou@hit.edu.cn

一、实验目的

1. 掌握Rucbase事务管理器的实现方法。
2. 掌握Rucbase锁管理器的实现方法。
3. 掌握Rucbase中两阶段锁协议的实现方法。

二、相关知识

1. 事务
2. 锁
3. 两阶段锁协议

三、实验内容

本实验包括2项任务。

任务1：事务管理器实现

补全`TransactionManager`类，实现事务管理器，具体完成下列任务。

（1）准备工作

在进行本实验之前，取消`rmdb.cpp`文件中`client_handler()`函数中对如下语句的注释：

- 第121行：

```
SetTransaction(&txn_id, context);
```

- 第183～186行：

```
if(context->txn_->get_txn_mode() == false)
{
    txn_manager->commit(context->txn_, context->log_mgr_);
}
```

（2）阅读代码

阅读`src/transaction`目录下的代码。

- `src/transaction/txn_defs.h`
- `src/transaction/transaction.h`
- `src/transaction/transaction_manager.h`

- `src/transaction/transaction_manager.cpp`

了解`WriteRecord`、`Transaction`和`TransactionManager`类的设计，并回答下列问题：

1. `WriteRecord`的作用是什么?
2. 全局静态成员变量`txn_map`有什么作用?

（3）实现`TransactionManager::begin`函数

函数声明：

```
Transaction* TransactionManager::begin(Transaction* txn, LogManager* log_manager);
```

功能：启动事务。

实现：参考代码注释。基本实现逻辑如下：如果是新事务，则需要创建一个`Transaction`对象，正确设置`Transaction`对象的状态，并把该对象的指针加入到全局事务表中`txn_map`中。

（4）实现`TransactionManager::commit`函数

函数声明：

```
void TransactionManager::commit(Transaction* txn, LogManager* log_manager);
```

功能：提交事务。

实现：参考代码注释。基本实现逻辑如下：如果并发控制协议需要申请和释放锁，则需要在提交阶段完成锁的释放。

（5）实现`TransactionManager::abort`函数

函数声明：

```
void TransactionManager::abort(Transaction* txn, LogManager* log_manager);
```

功能：中止事务。

实现：参考代码注释。基本实现逻辑如下：撤销事务的所有写操作，这些写操作记录在`txn->write_set_`中。如果并发控制协议需要申请和释放锁，则需要在中止阶段完成锁的释放。在回滚删除操作时，是否必须将元组插入到原位置？如果没有插入到原位置，会出现什么问题?

（6）单元测试

单元测试代码在`src/test/transaction/transaction_unit_test.py`文件中。执行下列命令，进行单元测试。

```
cd src/test/transaction
python transaction_unit_test.py commit_test
python transaction_unit_test.py abort_test
python transaction_unit_test.py commit_index_test
python transaction_unit_test.py abort_index_test
```

如果提示`command not found: python`，执行下列命令，进行单元测试。

```
cd src/test/transaction
python3 transaction_unit_test.py commit_test
python3 transaction_unit_test.py abort_test
python3 transaction_unit_test.py commit_index_test
python3 transaction_unit_test.py abort_index_test
```

任务2：锁管理器实现

补全`LockManager`类，实现锁管理器，具体完成下列任务。

（1）阅读代码

阅读`src/transaction`目录下的代码。

- `src/transaction/txn_defs.h`

了解`LockDataId`和`TransactionAbortException`类的设计。

阅读`src/transaction/concurrency`目录下的代码。

- `src/transaction/concurrency/lock_manager.h`
- `src/transaction/concurrency/lock_manager.cpp`

了解`LockManager`、`LockRequest`和`LockRequestQueue`类的设计，并回答下列问题：

1. `LockManager`类的成员变量`lock_table`的作用是什么？

（2）回顾知识

回顾相关理论知识。

1. 行级S锁、行级X锁、表级S锁、表级X锁、表级意向锁IS、表级意向锁IX之间的相容关系是什么？画出锁相容矩阵，再进行加锁解锁流程的梳理。
2. 如果并发事务发生死锁，该如何解决？

（3）实现`LockManager::lock_shared_on_record`函数

函数声明：

```
bool LockManager::lock_shared_on_record(Transaction* txn, const Rid& rid, int tab_fd);
```

功能：请求对元组加行级S锁。该操作需要被阻塞直到申请成功或失败。如果申请成功，则返回`true`，否则返回`false`。

实现：参考代码注释。

（4）实现`LockManager::lock_exclusive_on_record`函数

函数声明：

```
bool LockManager::lock_exclusive_on_record(Transaction* txn, const Rid& rid, int
tab_fd);
```

功能：请求对元组加行级X锁。该操作需要被阻塞直到申请成功或失败。如果申请成功，则返回`true`，否则返回`false`。

实现：参考代码注释。

（5）实现`LockManager::lock_shared_on_table`函数

函数声明：

```
bool LockManager::lock_shared_on_table(Transaction* txn, int tab_fd);
```

功能：请求对表加表级S锁。该操作需要被阻塞直到申请成功或失败。如果申请成功，则返回`true`，否则返回`false`。

实现：参考代码注释。

（6）实现`LockManager::lock_exclusive_on_table`函数

函数声明：

```
bool LockManager::lock_exclusive_on_table(Transaction* txn, int tab_fd);
```

功能：请求对表加表级X锁。该操作需要被阻塞直到申请成功或失败。如果申请成功，则返回`true`，否则返回`false`。

实现：参考代码注释。

（7）实现`LockManager::lock_IS_on_table`函数

函数声明：

```
bool LockManager::lock_IS_on_table(Transaction* txn, int tab_fd);
```

功能：请求对表加表级IS锁。该操作需要被阻塞直到申请成功或失败。如果申请成功，则返回`true`，否则返回`false`。

实现：参考代码注释。

（8）实现`LockManager::lock_IX_on_table`函数

函数声明：

```
bool LockManager::lock_IX_on_table(Transaction* txn, int tab_fd);
```

功能：请求对表加表级IX锁。该操作需要被阻塞直到申请成功或失败。如果申请成功，则返回`true`，否则返回`false`。

实现：参考代码注释。

（9）实现`LockManager::unlock`函数

函数声明：

```
bool LockManager::unlock(Transaction* txn, LockDataId lock_data_id);
```

功能：解锁。需要更新锁表。如果解锁成功，则返回`true`，否则返回`false`。

实现：参考代码注释。

（10）解锁

- `unlock(Transaction *, LockDataId)`：解锁操作。
- 需要更新锁表，如果解锁成功则返回true，否则返回false。

注意事项

注意对`txn_map`等全局共享数据结构的竞争访问。

任务3：两阶段锁协议实现

Rucbase采用两阶段锁并发控制协议，实现可串行化隔离级别，并用no-wait策略来解决死锁问题。该任务需要在相关函数中添加代码，调用任务2中锁管理器的加锁和解锁接口，在合适的地方申请行级锁和意向锁，并在合适的地方解锁，从而实现两阶段锁协议。具体完成下列任务。

（1）在`RmFileHandle::get_record`函数中添加请求加锁的代码

修改`src/record/rm_file_handle.cpp`文件，在`RmFileHandle::get_record`函数中添加代码，申请对应表上的IS锁和对应元组上的S锁。

（2）在`RmFileHandle::insert_record`函数中添加请求加锁的代码

修改`src/record/rm_file_handle.cpp`文件，在`RmFileHandle::insert_record`函数中添加代码，申请对应表上的IX锁和对应元组上的X锁。

（3）在`RmFileHandle::delete_record`函数中添加请求加锁的代码

修改`src/record/rm_file_handle.cpp`文件，在`RmFileHandle::delete_record`函数中添加代码，申请对应表上的IX锁和对应元组上的X锁。

（4）在`SmManager::create_table`函数中添加请求加锁的代码

修改`src/system/sm_manager.cpp`文件，在`SmManager::create_table`函数中添加代码，申请对应表上的X锁。

（5）在`SmManager::drop_table`函数中添加请求加锁的代码

修改`src/system/sm_manager.cpp`文件，在`SmManager::drop_table`函数中添加代码，申请对应表上的X锁。

（6）在`SmManager::create_index`函数中添加请求加锁的代码

修改`src/system/sm_manager.cpp`文件，在`SmManager::create_index`函数中添加代码，申请对应表上的S锁。

（7）在`SmManager::drop_index`函数中添加请求加锁的代码

修改`src/system/sm_manager.cpp`文件，在`SmManager::drop_index`函数中添加代码，申请对应表上的S锁。

（8）在`SeqScanExecutor::begin_tuple`函数中添加请求加锁的代码

修改`src/execution/seq_scan_executor.cpp`文件，在`SeqScanExecutor::begin_tuple`函数中添加代码，申请对应表上的S锁。

（9）在`IndexScanExecutor::begin_tuple`函数中添加请求加锁的代码

修改`src/execution/index_scan_executor.cpp`文件，在`IndexScanExecutor::begin_tuple`函数中添加代码，申请对应表上的S锁。

（7）单元测试

单元测试代码在文件`src/test/concurrency/concurrency_unit_test.py`中。

执行下列命令，进行单元测试。

```
cd src/test/concurrency
python concurrency_unit_test.py concurrency_read_test
python concurrency_unit_test.py dirty_write_test
python concurrency_unit_test.py dirty_read_test
python concurrency_unit_test.py lost_update_test
python concurrency_unit_test.py repeatable_read_test
python concurrency_unit_test.py repeatable_read_test_hard
```

如果提示`command not found: python`，执行下列命令，进行单元测试。

```
cd src/test/concurrency
python3 concurrency_unit_test.py concurrency_read_test
python3 concurrency_unit_test.py dirty_write_test
python3 concurrency_unit_test.py dirty_read_test
python3 concurrency_unit_test.py lost_update_test
python3 concurrency_unit_test.py repeatable_read_test
python3 concurrency_unit_test.py repeatable_read_test_hard
```