《数据库系统实验》课程设计报告

说明：本模板为“Rucbase实验配置与存储管理”课程设计专用。

|  |  |  |  |
| --- | --- | --- | --- |
| **题目** | Rucbase实验配置与存储管理 | | |
| 小组成员信息 | | | |
| **姓名** | **学号** | **班级** | **分工** |
| **杨子昂** | 21307181 | 计算机科学与技  术（人工智能） | 任务1.3 缓冲池管理器 |
|  |  |  |  |
|  |  |  |  |

提交时间： 年 月 日

1. 开发环境与开发工具

具体实验目的

梁铭恩

1. 具体模块设计（40分）

各子任务的核心代码（一定要加适当注释），包括：

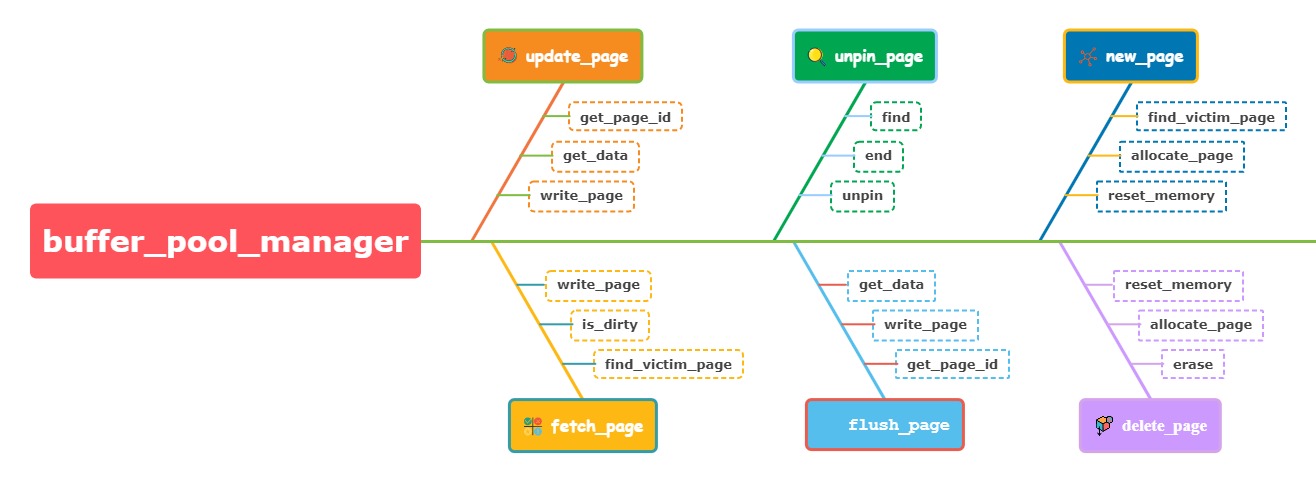
任务一 缓冲池管理器

任务1.1 磁盘存储管理器（10分）张序

任务1.2 缓冲池替换策略（10分）梁铭恩

任务1.3 缓冲池管理器（10分） 杨子昂

Overview：



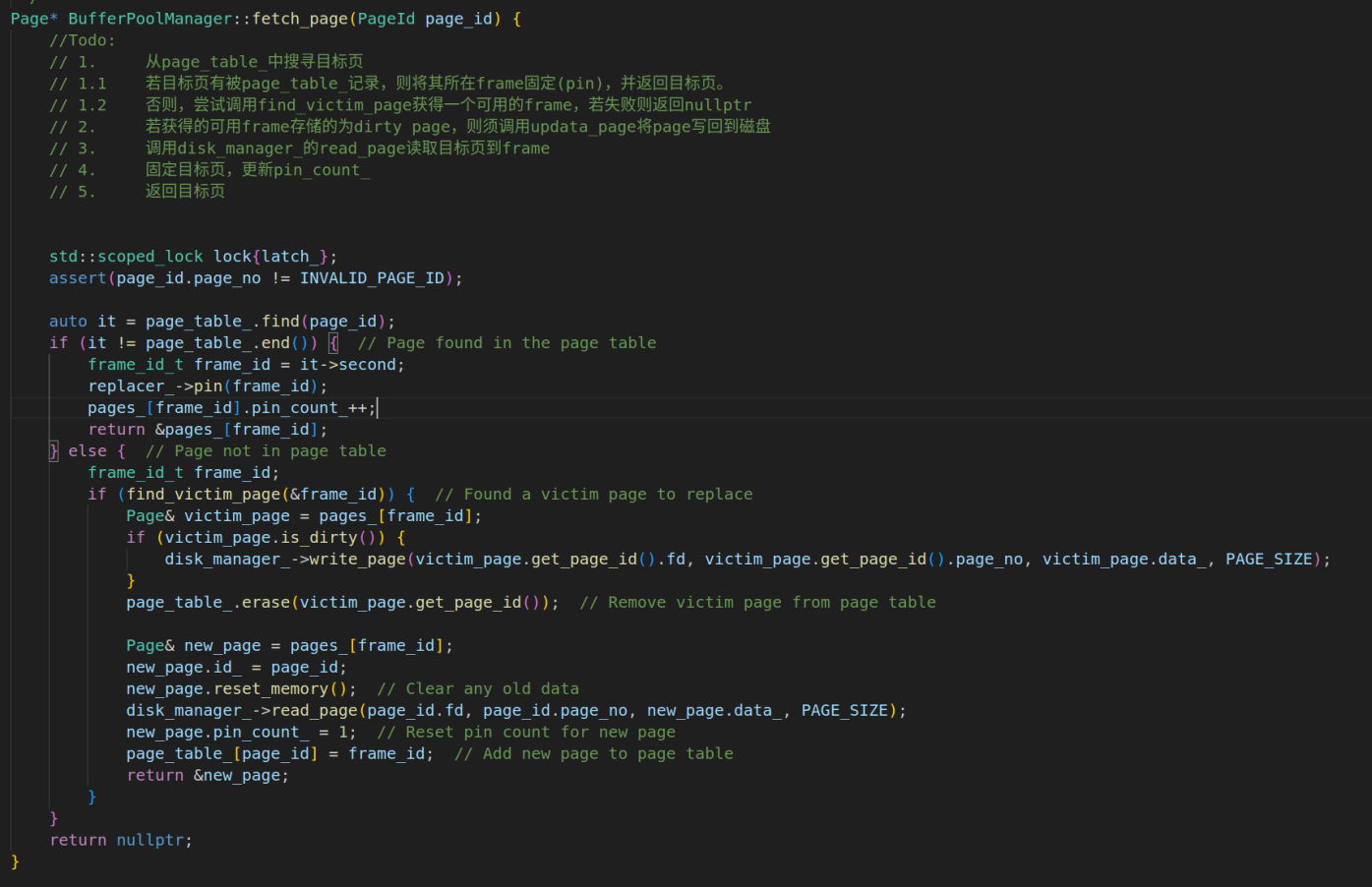
1. bool BufferPoolManager::find\_victim\_page(frame\_id\_t\* frame\_id): 用于在缓冲池管理器中查找一个可被替换的页面。参数是指向 frame\_id 的指针，返回值为布尔类型，表示是否找到了合适的页面。
2. void BufferPoolManager::update\_page(Page \*page, PageId new\_page\_id, frame\_id\_t new\_frame\_id): 用于更新缓冲池中的某个页面。接受页面指针、新的页面ID和帧ID作为参数，无返回值。
3. Page\* BufferPoolManager::fetch\_page(PageId page\_id): 用于获取缓冲池中的指定页面。参数为页面ID，返回值为指向该页面的指针。
4. bool BufferPoolManager::unpin\_page(PageId page\_id, bool is\_dirty): 用于“取消固定”缓冲池中的一个页面。接受页面ID和布尔值（表示页面是否已被修改）作为参数，返回布尔值表示操作是否成功。
5. bool BufferPoolManager::flush\_page(PageId page\_id): 用于将缓冲池中的特定页面刷新到存储中。参数为页面ID，返回布尔值表示操作是否成功。
6. Page\* BufferPoolManager::new\_page(PageId\* page\_id): 用于在缓冲池中创建一个新页面。参数为指向页面ID的指针，返回值为指向新创建页面的指针。
7. bool BufferPoolManager::delete\_page(PageId page\_id): 用于删除缓冲池中的一个页面。参数为页面ID，返回布尔值表示操作是否成功。
8. void BufferPoolManager::flush\_all\_pages(int fd): 用于将缓冲池中的所有页面刷新到存储中。参数为文件描述符，无返回值。

在该部分实验中，我们主要需要实现上面这8个函数，并通过最终的测试程序。

关键代码展示：

由于一共有八个函数且基本上每一个函数都是表层调用的级别，因此都十分重要，并且也会需要频繁调用前面两位同学的函数，比较繁琐而且代码篇幅很长，因此我会从中挑选几个比较有代表性的难度较大的函数进行分析：

（1）fetch\_page

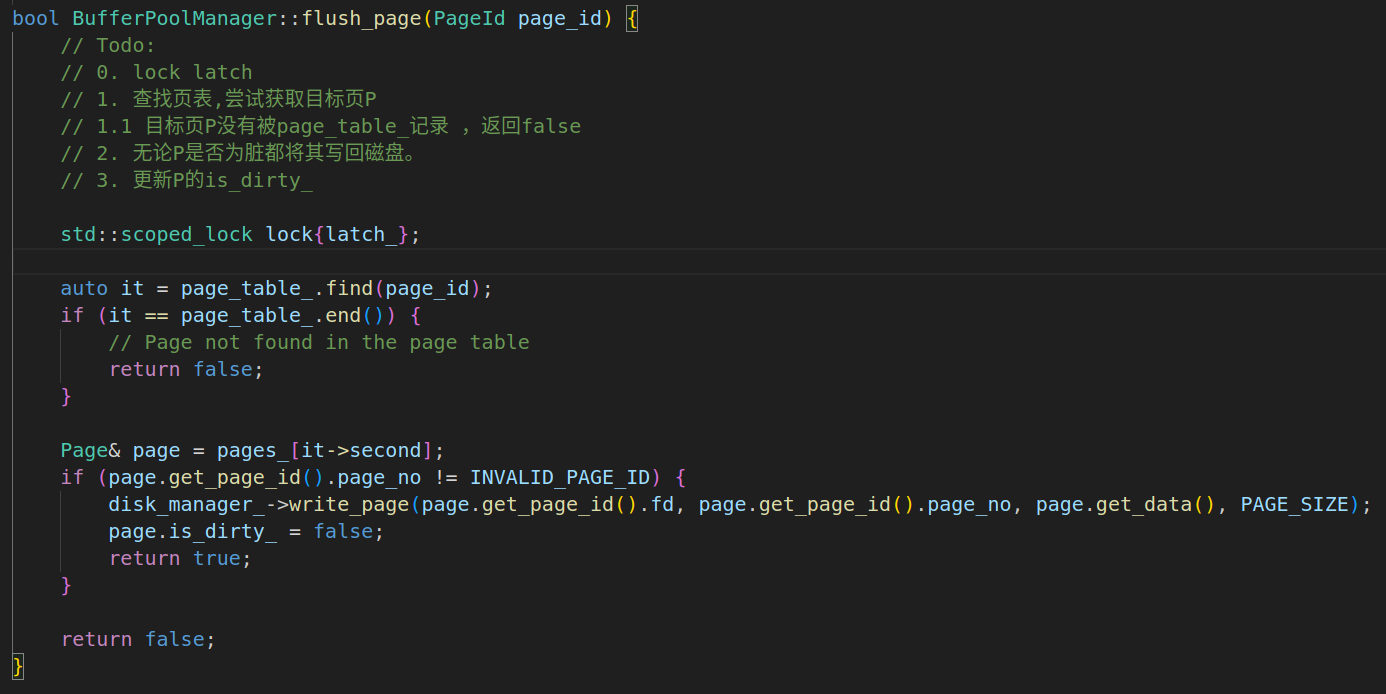


上面截图中的函数是fetch\_page函数

1. 搜索页表：首先在页表中搜索目标页面。
2. 页面存在：如果页面已在页表中，则固定该页面并返回它。
3. 页面不存在：如果页面不在页表中，则尝试找到一个可替换的页面（受害者页面）。
4. 脏页处理：如果找到的页面是脏页（已被修改），则先将其写回磁盘。
5. 读取新页面：从磁盘读取目标页面到缓冲池。
6. 更新页表：更新页表，记录新页面的位置。
7. 返回页面：返回指向新读取页面的指针。

总结来说，这个函数的主要目的是获取缓冲池中的一个指定页面。如果页面已存在于缓冲池中，则直接返回它；如果不存在，则尝试找到一个可用的帧（可能需要替换一个已存在的页面），将所需的页面从磁盘读入该帧，并返回它。这个过程涉及对页面的固定（pinning）、引用计数管理、脏页的写回以及页表的更新。

（2）flush\_page



1. 加锁: 使用 std::scoped\_lock 对象 lock 锁定，以确保线程安全。
2. 搜索页表: 在页表 page\_table\_ 中查找目标页面 page\_id。

如果页面不存在：如果目标页面没有被记录在页表中（it == page\_table\_.end()）， 函数返回 false。

1. 写回磁盘:

获取目标页面的引用（Page& page = pages\_[it->second]）。

无论页面是否脏（即是否被修改过），都将其内容写回磁盘 （disk\_manager\_->write\_page）。

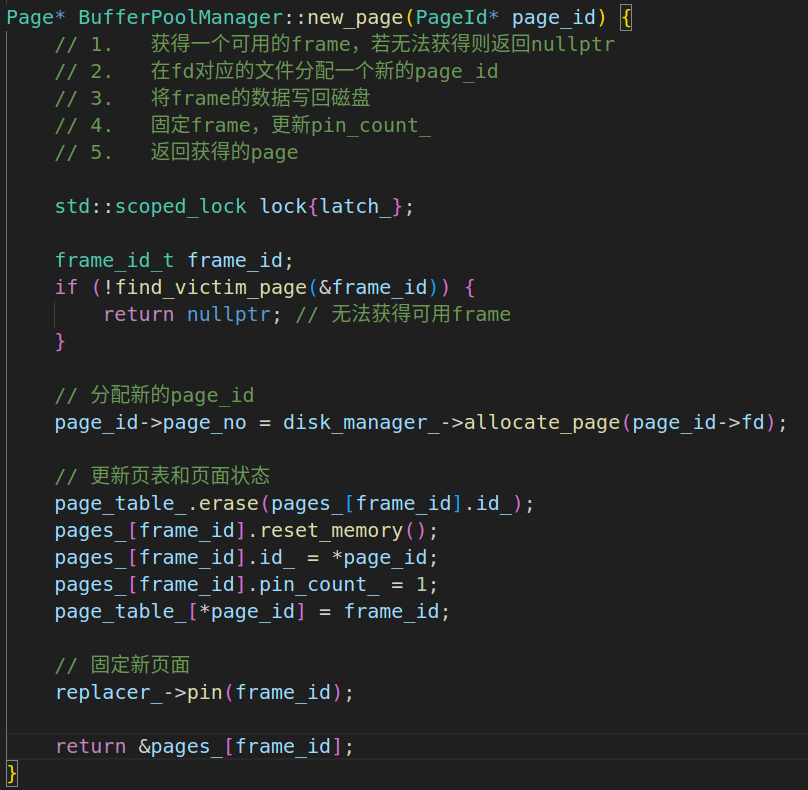
更新页面的脏标志（page.is\_dirty\_ = false），表示页面已同步到磁盘。

1. 返回结果:

如果成功执行了写回操作，函数返回 true。

如果页面的页面号无效（page.get\_page\_id().page\_no != INVALID\_PAGE\_ID 不成 立），函数返回 false。

（3）new\_page



1. 获取可用的帧:

使用 find\_victim\_page(&frame\_id) 尝试获得一个可用的帧（frame）。如果无 法获得（函数返回 false），则函数返回 nullptr。

1. 分配新的页面ID:

调用 disk\_manager\_->allocate\_page(page\_id->fd) 为新页面分配一个新的 page\_id。

1. 更新页表和页面状态:

从页表 page\_table\_ 中移除当前帧存储的旧页面的记录。

重置帧的内存（pages\_[frame\_id].reset\_memory()）。

将新的页面ID赋值给帧（pages\_[frame\_id].id\_ = \*page\_id）。

设置新页面的引用计数为1（pages\_[frame\_id].pin\_count\_ = 1）。

在页表中添加新页面的记录。

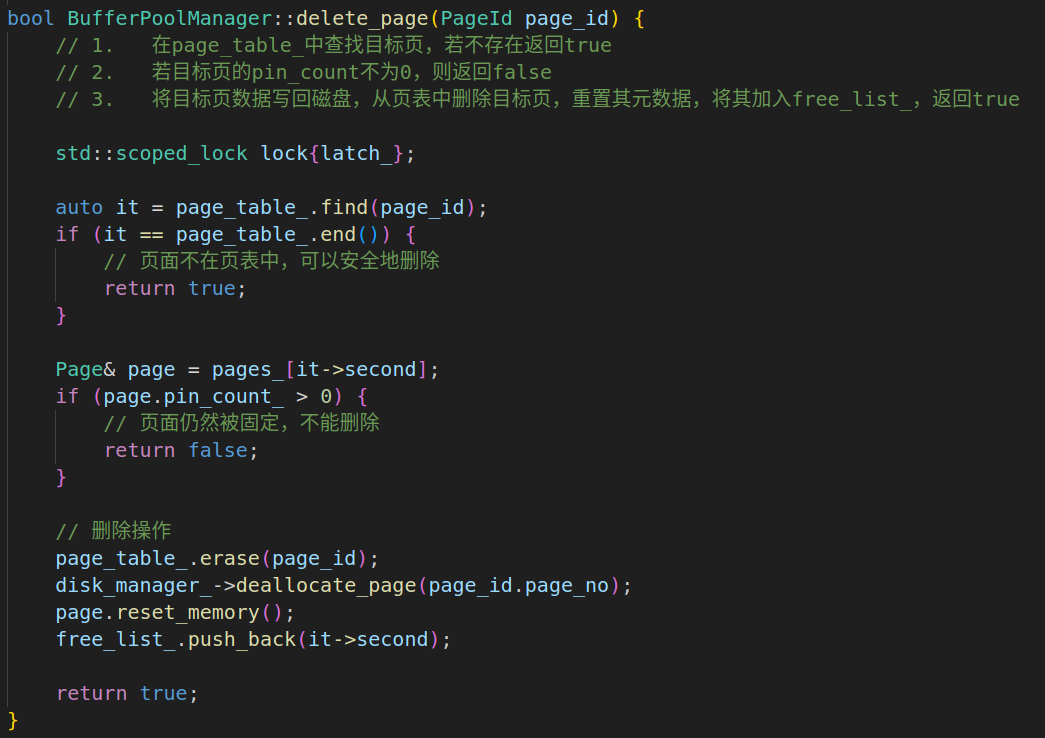
1. 固定新页面:

使用 replacer\_->pin(frame\_id) 固定新的页面，防止它被其他操作替换。

1. 返回新页面:

返回指向新分配页面的指针（&pages\_[frame\_id]）。

（4）delete\_page



1. 搜索页表:

在页表 page\_table\_ 中查找目标页面 page\_id。

如果页面不存在：如果页表中没有找到目标页面（it == page\_table\_.end()）， 函数返回 true，表示页面可以安全删除（因为它本来就不存在于缓冲池中）。

1. 检查页面固定状态:

获取目标页面的引用（Page& page = pages\_[it->second]）。

如果页面被固定：如果目标页面的 pin\_count\_ 大于0（即页面被固定，不能 被删除），函数返回 false。

1. 执行删除操作:

从页表中移除目标页面的记录（page\_table\_.erase(page\_id)）。

调用 disk\_manager\_->deallocate\_page(page\_id.page\_no) 在磁盘上释放页面。

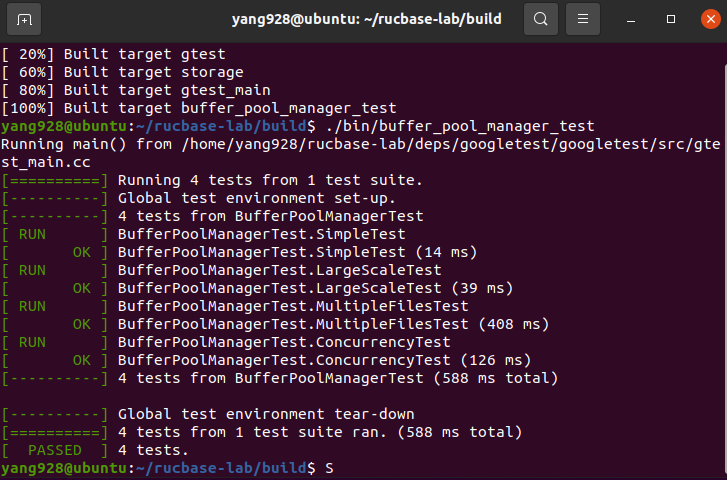
重置页面的内存（page.reset\_memory()）。

将页面的帧ID加入到空闲列表 free\_list\_ 中。

1. 返回成功:

删除操作完成后，函数返回 true。

1. 功能测试



可以看到全部都是绿色的pass测试点。

五. 总结

本课程设计中用到的《数据库系统原理》理论课概念与知识。

缓冲池管理（Buffer Pool Management）:

* 这是数据库管理系统中的一个核心组件，负责管理内存中的数据页，以减少磁盘I/O操作。
* 缓冲池提供了一种机制，使得频繁访问的数据可以保留在内存中，而不是每次都从磁盘读取。

页面置换算法（Page Replacement Algorithm）:

* 当缓冲池满时，需要选择一个页面进行替换。这涉及到复杂的算法，比如 LRU（最近最少使用）或其他策略，以确定哪个页面应该被移除。
* 例如，find\_victim\_page 函数可能就是用来实现这种页面置换策略的。

脏页的处理（Dirty Page Management）:

* 脏页指的是已经被修改但还没有写回磁盘的页面。
* 管理脏页，确保在适当的时机将其内容同步回磁盘，是缓冲池管理的重要部分。

页固定（Page Pinning）:

* 当一个页面被数据库的某个组件使用时，它会被“固定”，以防止被置换出缓冲池。
* 这在 unpin\_page 和其他函数中有所体现，确保在使用页面时不会丢失数据。

事务的持久性（Transaction Durability）:

* 确保事务的更改在提交后永久保存，即使在系统故障后也是如此。
* 函数如 flush\_page 和 delete\_page 都与确保数据的持久性和一致性有关。

锁和并发控制（Locking and Concurrency Control）:

* 使用锁（如 std::scoped\_lock）来确保在并发环境下数据的一致性和完整性。

这是数据库管理系统中处理多个同时运行的事务的重要机制。

任务二 记录管理器

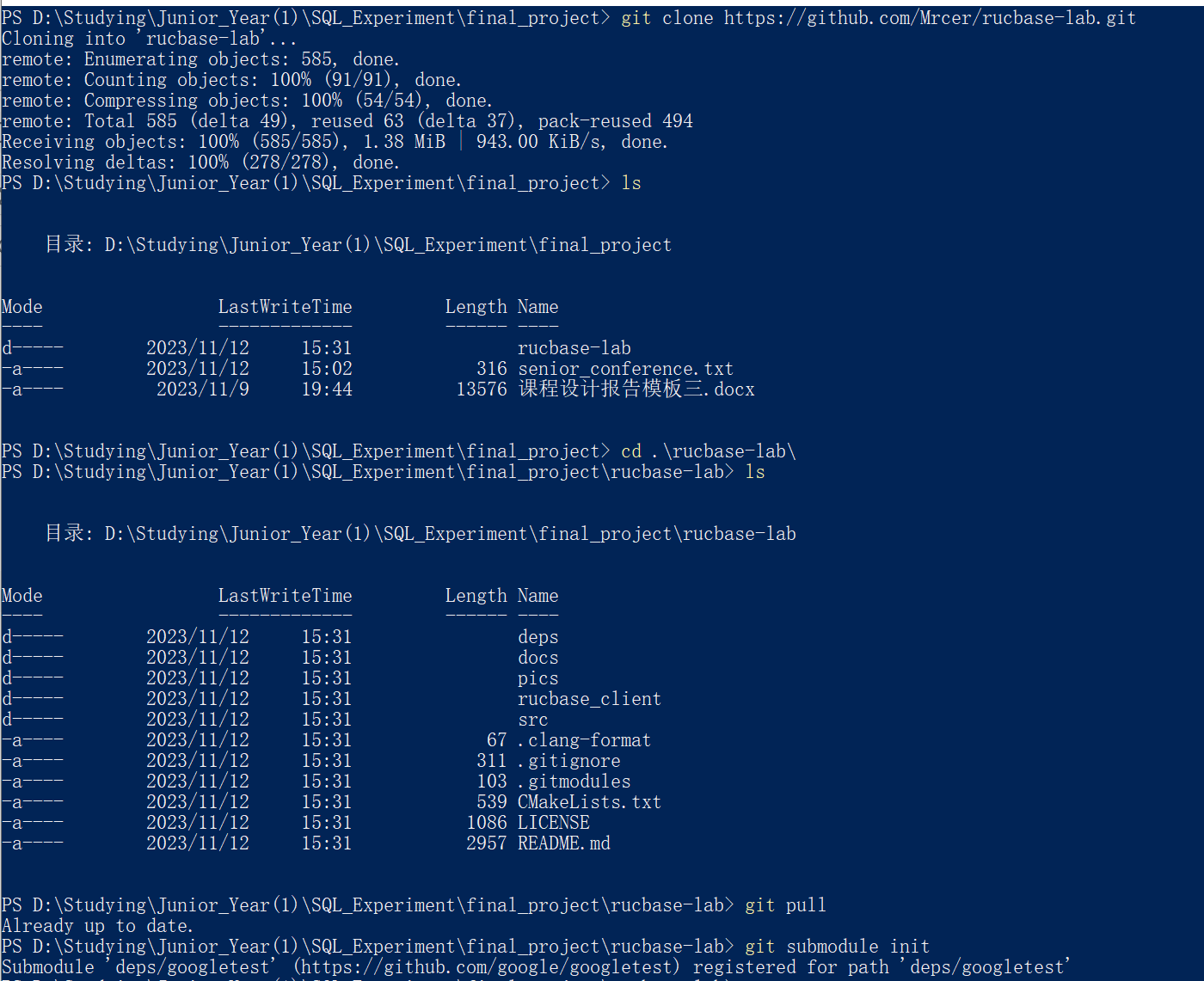
任务2.1 记录操作（5分）毕泽同

任务2.2 记录迭代器（5分）毕泽同

1. 功能测试（10分）

写出测试流程及相关截图。

五. 总结

 本课程设计中用到的《数据库系统原理》理论课概念与知识。