

学号	姓名	邮箱	完成题目
205220025	郑凯琳	<a href="mailto:205220025@smail.nju.edu.cn">205220025@smail.nju.edu.cn</a>	1/2/3/f1/f2

### **t1. LRU**

#### **实验细节：**

##### **(1) Victim**

1. 遍历 `lru_list_` 中的帧，寻找标记为 **可淘汰** 的帧。
2. 找到后，从链表和哈希表中删除，并更新当前大小。
3. 如果没有可淘汰的帧，返回 `false`。

##### **(2) Pin**

检查帧是否已存在：

1. 若存在，从链表中移除。
2. 将帧重新插入链表尾部，并标记为不可淘汰。

##### **(3) Unpin**

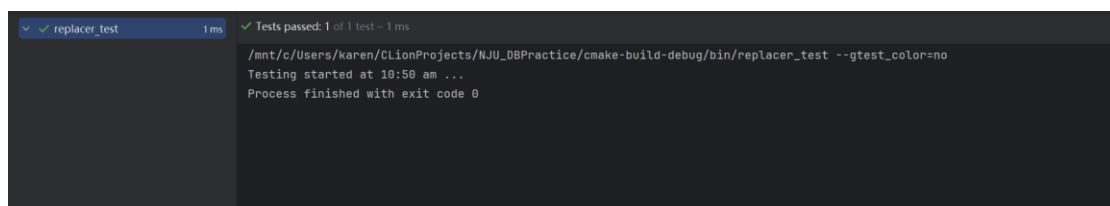
检查帧是否已存在：

1. 若存在且为固定状态，修改为可淘汰状态。
2. 若帧不存在，直接插入链表尾部，标记为可淘汰。

##### **(4) Size**

返回成员变量 `cur_size_` 的值。

#### **实验结果：**



```
✓ replacer_test 1 ms ✓ Tests passed: 1 of 1 test - 1 ms
/mnt/c/Users/karen/CLionProjects/NJU_DBPractice/cmake-build-debug/bin/replacer_test --gtest_color=no
Testing started at 10:50 am ...
Process finished with exit code 0
```

### **t2. Buffer Pool Manager**

#### **实现思路：**

##### **(1) FetchPage**

检查页面是否已经在缓冲池中，通过 `page_frame_lookup_` 查找 `fid_pid_t` 键值。如果在缓冲池中，则固定页面，并通过 `replacer_` 通知页面被固定，返回页面。如果不在缓冲池中，通过 `GetAvailableFrame()` 获取一个可用框架，将页面从磁盘读取到此框架，并更新映射表。

##### **(2) UnpinPage**

检查页面是否在缓冲池中且被固定，若引用计数大于 0 则减少引用计数。如果页面是脏页，则标记为脏，并将页面写回磁盘。通知替换策略页面已被解除固定。

### (3) DeletePage 和 DeleteAllPages

检查页面是否在缓冲池中，如果页面没有被引用且存在，直接删除。如果是脏页，则先写回磁盘，再重置框架并更新映射表。

### (4) FlushPage 和 FlushAllPages

检查页面是否脏页，如果是则写回磁盘，并重置脏标记。遍历所有页面，将属于指定文件的脏页统一写回磁盘。

### (5) GetAvailableFrame

优先从 free\_list\_ 获取空闲框架。如果没有空闲框架，则使用 Victim() 从替换器中选择牺牲框架。如果框架脏，则写回磁盘并重置框架。

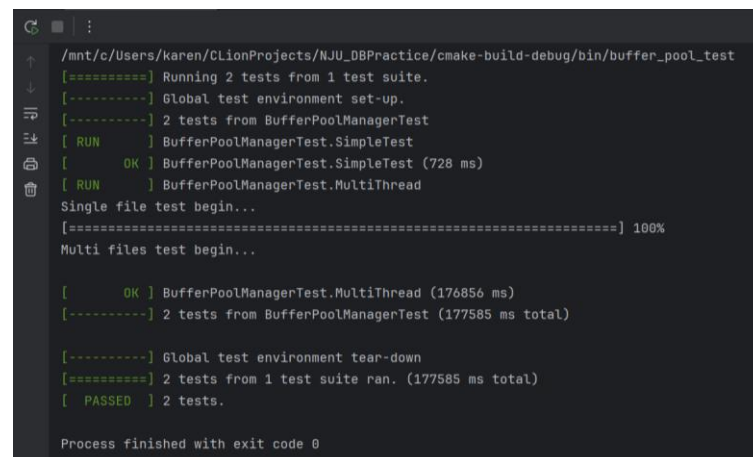
### (6) UpdateFrame

读取新的页面数据并将其存储在指定框架中。如果框架先前的页面是脏页，先写回磁盘再更新。更新映射表，并通知替换器页面被固定。

### 优化技巧：

- 高效的线程安全处理：通过 std::scoped\_lock 实现锁的自动管理，避免死锁并提高代码可读性。
- 替换策略选择：通过 LRU 替换算法的适配提升性能。
- 脏页标记优化：避免不必要的磁盘写操作，减少 I/O 开销。

### 实验结果：



```
/mnt/c/Users/karen/CLionProjects/NJU_DBPractice/cmake-build-debug/bin/buffer_pool_test
[=====] Running 2 tests from 1 test suite.
[-----] Global test environment set-up.
[-----] 2 tests from BufferPoolManagerTest
[ RUN ] BufferPoolManagerTest.SimpleTest
[ OK ] BufferPoolManagerTest.SimpleTest (728 ms)
[ RUN ] BufferPoolManagerTest.MultiThread
Single file test begin...
[=====] 100%
Multi files test begin...

[ OK ] BufferPoolManagerTest.MultiThread (176856 ms)
[-----] 2 tests from BufferPoolManagerTest (177585 ms total)

[-----] Global test environment tear-down
[=====] 2 tests from 1 test suite ran. (177585 ms total)
[ PASSED ] 2 tests.

Process finished with exit code 0
```

**思考：**脏页管理的重要性：显著降低了磁盘 I/O 开销，这是数据库系统优化的关键之一。

## t3. Table Handle

**实现思路：**大致上按照头文件的步骤实现即可。

### 实验细节：

#### (1) InsertRecord

- 获取空槽位：FindFirst 搜索第一个可用槽位

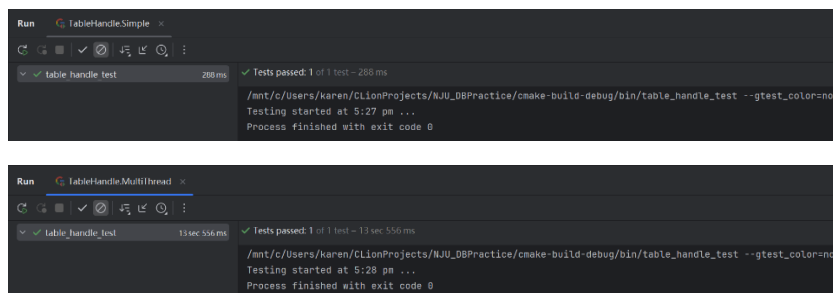
- 更新位图和页面头信息：
  - SetBit 函数标记为 true 表示槽位已占用
  - 全局记录计数 tab\_hdr.rec\_num 递增：增加表级的总计数
  - GetRecordNum()：获取改页当前记录的数量
  - SetRecordNum()：将页内记录数自增并更新到当前页

→ void InsertRecord：针对特定位置插入

(2) DeleteRecord 的更新操作与 InsertRecord 相反（false，递减）

优化技巧：位图优化、预先检查槽位状态，避免不必要的读写。

实验结果：



## f1. LRU K Replacer

实验思路/细节：

AddHistory：向节点的历史时间戳列表中添加当前时间戳，并保持列表长度不超过 K。如果列表超过长度，移除最早（最老）的时间戳。

GetBackwardKDistance：计算当前时间戳与历史中的第 K 个时间戳之间的差值。如果历史中的时间戳少于 K 个，返回+inf。

两个辅助函数：**FetchOldestTimestamp** 和 **ResetTimestampHistory**，用于获取最早的时间戳和重置历史时间戳列表。

Victim：

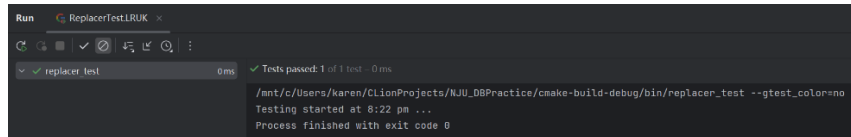
- 遍历所有可驱逐页面，选择“逆向 K 距离”最大的页面作为淘汰候选，若多个页面距离相同，则选择时间戳最早的页面。
- 特殊情况处理：若逆向 K 距离不可计算，则按最早时间戳选取。
- 淘汰后重置节点时间戳历史，标记为不可驱逐，并更新驱逐计数。

其他大致上与 LRU 相同。

优化技巧：

- 减少冗余操作：使用 emplace 构造并插入节点，避免二次查找；仅在必要时修改 cur\_size\_。
- 现代 C++ 特性：使用 std::numeric\_limits 处理最大值，简化代码；引用减少对象拷贝，提高性能。

实验结果：



## f2. PAX Page Handle

实验思路/细节：

### (1) TableHandle 类中的构造函数和 GetChunk 方法

初始化 TableHandle 对象，包括设置表 ID 和计算字段偏移量，特别是在 PAX 存储模型中。从指定页面读取数据块，使用页面句柄的 ReadChunk 方法，并在完成后释放页面资源。

### (2) PAXPageHandle 类中的 ReadSlot 和 WriteSlot 方法

两者都操作页面中的记录槽位，使用字段偏移量数组 offsets\_ 来定位字段数据，都涉及到 null map 的处理，以及基于字段大小和记录数的数据操作。

ReadSlot 从页面读取数据到提供的缓冲区，而 WriteSlot 将数据从缓冲区写入页面。ReadSlot 需要验证槽位是否为空，而 WriteSlot 需要标记槽位为已占用。

### (3) PAXPageHandle 类中的 ReadChunk 方法

构建一个数据块，包含多个记录的字段值，使用 lambda 表达式处理每个字段的读取。

**优化技巧：**使用 std::vector 避免手动内存管理；预先分配向量大小，减少动态内存分配；使用 lambda 表达式提高代码复用性和可读性。

实验结果：

