实验报告—实验1：存储管理

学号	姓名	邮箱	完成题目
221220067	刘思远	221220067@smail.nju.edu.cn	1/2/3/f1

1 完成情况

完成了`l1.t1`、`l1.t2`、`l1.t3`、`l1.f1`，没有完成`l1.f2`。

在考察了总评分数占比组成和实验一的各子题目占比组成后，发现`l1.f2`即使不做，对总评的影响也可以忽略不计；做的话可能需要投入大量的时间，且功利性的回报和付出看起来并不成正比。综上，选择放弃`l1.f2`。

2 实验详述

2.1 t1 LRU-Replacer

2.1.1 思路

首先要明确一点的是，`lru_list_`中存放的不只有可以淘汰的frame，事实上，只要pin过且pin的时候`lru_list_`中还有存放空间，就会被存放进去，因此一般来说不能用`lru_list_.size()`来直接代表可以驱逐的frame的数量。

其次，自己实现过程中有两个易错点：

- `Pin`函数中，当`lru_hash_`中没有`frame_id`对应项的时候，需要向`lru_list`中加入`frame_id`。此时需要判断其大小是否已经达到了`max_size_`，达到的话需要先`Victim()`一下，剔除一个元素。
- `Victim`函数中，直接`return`的条件应该是`cur_size_ == 0`，而非`lru_list_.empty()`。

2.1.2 优化

- 由于采用头插的方式，故淘汰时应该从`lru_list`的尾部开始遍历，所以需要两次`reverse()`操作。
- 在`Pin`中，当`frame_id`已经在`lru_hash_`中的时候，直接调用STL的`splice`操作，来实现对于其在`lru_list`中存储位置的修改。

2.1.3 实验结果

```
root@LAPTOP-HPEFQ5QA:/home/sylaw/NJU_DBPractice/build# ./bin/replacer_test
[==========] Running 2 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 2 tests from ReplacerTest
[ RUN      ] ReplacerTest.LRU
[       OK ] ReplacerTest.LRU (0 ms)
```

2.2 t2 buffer pool manager

2.2.1 思路

思路上没什么特别需要说的，按照头文件给出的步骤一步步实现即可。

2.2.2 优化和问题

问题：（1）在`DeleteAllPages`中，由于已经在`DeletePage`中加过锁了，所以没有必要再在删除所有page的时候加锁（`FlushAllPages`同理）。（2）在`GetAvailableFrame`的时候，由于当`free_list`为空的时候，需要使用`replacer`淘汰frame，淘汰的时候记得把它从`page_frame_look_up`当中清除。（3）`UpdateFrame`中，在用新的`Page`数据更新`frame`之前，记得先把`page`进行`clear`操作，将其中存放的数据重置。

优化：在这里没有作过多的优化。

2.2.3 经验教训

由于后面的lab会调用之前的lab实现的接口，所以当此lab无法通过的时候，不一定是这个lab本身的问题，要回头看。

2.2.4 实验结果

```
root@LAPTOP-HPEFQ5QA:/home/sylaw/NJU_DBPractice/build# ./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 (12 ms)
[ RUN      ] BufferPoolManagerTest.MultiThread
Single file test begin...
[============================================================================] 100%
Multi files test begin...
[============================================================================] 100%
[       OK ] BufferPoolManagerTest.MultiThread (18669 ms)
[----------] 2 tests from BufferPoolManagerTest (18682 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 1 test suite ran. (18682 ms total)
[  PASSED  ] 2 tests.
```

2.3 t3 Table Handle

2.3.1 思路

思路同上，没有特别需要说的，按照头文件给出的大致步骤，一步步实现即可，但是有几个需要特别注意的地方。放在下一部分“细节”中叙述。

2.3.2 细节

（1）`GetRecord`中，当slot为空时，记得throw异常；（2）在两个`InsertRecord`中，更新`bitmap`和记录的数量时，记得把元数据中的总记录数也更新；（3）`DeleteRecord`的时候，如果删除记录后，变成空页，则需要加入空页的链表；（4）对两个不同的`InsertRecord`的不同处理方式：

- `auto TableHandle::InsertRecord(const Record &record) -> RID`由于找空白页时是直接找的首空白页，所以如果插入记录后页面满，直接把非空页头结点变为第二个就行；
- `void TableHandle::InsertRecord(const RID &rid, const Record &record)`由于给定了`rid`，则需要插入指定页面。那么，当插入记录后页面满，就需要分类讨论：
  - 指定页面本来就是`first_free_page_`，同上；
  - 指定页面不是`first_free_page_`，则需要遍历到指定页面的前一个页面`X`，把`X`的后一个空闲页面变为指定页面的后一个空闲页面。

```
auto prev = FetchPageHandle(tab_hdr_.first_free_page_);
while (prev->GetPage()->GetNextFreePageId() != page->GetPageId(){
    prev = FetchPageHandle(prev->GetPage()->GetNextFreePageId());
}
prev->GetPage()->SetNextFreePageId(page->GetNextFreePageId());
page->SetNextFreePageId(INVALID_PAGE_ID);
```

2.3.3 实验结果

```
root@LAPTOP-HPEFQ5QA:/home/sylaw/NJU_DBPractice/build# ./bin/table_handle_test
[==========] Running 3 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 3 tests from TableHandle
[ RUN      ] TableHandle.Simple
[       OK ] TableHandle.Simple (4 ms)
[ RUN      ] TableHandle.MultiThread
[       OK ] TableHandle.MultiThread (1019 ms)
```

2.4 f1 LRUK-Replacer

2.4.1 思路

由于在任意时刻的当前访问索引都是一样的，则可以把最大的distance转化为最小的backward。那么在`GetBackwardDistance`的时候，直接返回得到的backward的值，如果出现次数还没到k，就返回`unsigned long long`能表示的最小值0。

在所给代码基础上新增一个数据结构`lru_list`，元素为`std::pair<frame_id, unsigned long long>`来存已经访问的序列。

其余的思路跟LRU-Replacer的是一致的，只不过在一些细节处的实现有变化。

2.4.2 优化

1. 把distance最大转化为backward最小
2. 更新`lru_list`的时候，为了保证插入是有序的，采用`std::upper_bound`来解决，按照其backward排序。由于`std::upper_bound`的性质，可以保证`lru_list`是按照backward从小到大排序的，backward相等的，后来的在后面。

2.4.3 实验结果

```
[ RUN      ] ReplacerTest.LRUK
[       OK ] ReplacerTest.LRUK (0 ms)
[----------] 2 tests from ReplacerTest (1 ms total)
```