数据库专题训练 · Lab3

计01　容逸朗　2020010869

实验目的

1. 初步了解基于 MVCC 技术的多线程事务处理；
2. 对数据库系统中并发控制机制有进一步的理解。

基础实验内容

1. 添加隐藏列操作接口

- 和 `RecordFactory` 的 `SetRid` 、`GetRid` 函数类似，增加对创建版本号（`CreateXID`）和删除版本号（`DeleteXID`）的隐藏列操作接口即可。

2. 多线程场景下的记录页面的操作

- **插入：**和原来的 `InsertRecord` 一致，同时需要调用 `SetCreateXID` 设置创建版本号；
- **删除：**和原来的 `DeleteRecord` 相似，需要调用 `SetDeleteXID` 设置删除版本号，同时由于 MVCC 删除不能直接清除数据，因此不需要对位图进行操作。
- **查找：**需要排除无效数据，规则如下：
  - 未开始的插入：记录的 `CreateXID` 在 `uncommit_xids` 中，或者比当前事务的编号更大；
  - 已提交的删除：`DeleteXID` 存在但不在 `uncommit_xids` 中，且值不大于当前事务编号。

3. 多线程场景下的表操作

- **插入：**插入数据时改用 `PageHandle::InsertRecord(Record*, XID)` 的接口即可；
- **删除：**删除数据时使用 `PageHandle::DeleteRecord(SlotID, XID, bool)` 接口，同时由于数据没有被真正删除，故不需要更新 `meta` 的 `first_free_` 项；
- **更新：**采用删除旧数据并插入新数据的方法，流程如下：
  - 首先记录删除日志，删除旧数据；
  - 然后找到新的位置，在 meta 中保存 Rid 信息，然后增加插入日志，最后再插入新数据；
  - 最后维护页面的空闲信息即完成更新操作。

4. Checkpoint 恢复当前事务编号

- 在 `CheckpointLog` 的 `Store` 和 `Load` 函数中分别调用 `TxManager` 的 `GetXID` 和 `SetXID` 接口取得（设置）当前事务的编号即可。

高级功能

本次实验中，我还实现了 MVCC 的垃圾回收功能。

1. 设计方案

- 引入日志机制后，我们很难在数据库运行期间进行垃圾收集；
- 因此我选择了在数据库关闭时进行垃圾回收。具体而言，在 `SystemManager` 执行 `CloseDatabase` 关闭数据库，且调用 `FlushAll` 函数前，做如下操作：
  - 遍历 `tables_` 中的所有表，然后访问从 0 到 `table_end_page_` 中的所有页面，执行回收；
  - 回收的逻辑放在 `PageHandle` 中，方法和 `PageHandle::LoadRecords` 类似，但是判断的标准改为是否有合法的 `DeleteXID` 项，若有则可以删除（把对应位置的位图置空）并把页面记为脏页；
  - 对于所有成功回收垃圾的页，我们可以用一个双向链表把他们连接起来，为此需要在 `PageHeader` 中加入 `last_page` 表示上一个空页面，原有的 `next_page` 不变，记为下一个空页；
  - 然后顺序遍历成功回收页集合，维护 `last_page` 和 `next_page` 项即可；
- 接下来还需要更改分配页的逻辑：
  - 为此需要更改页满后的处理方式，我加入了 `Table::FindNextPage(PageHandle)` 函数。
  - 该函数通过当前页面的 `PageHandle` 取得对应的 `last_page` 和 `next_page` 项；
  - 先判断 `last_page` 是否为 NULL_PAGE，若是则把 `first_free_` 置为 `next_page`；
  - 否则把 `last_page` 的后继页设为 `next_page`，然后把 `first_free_` 置为 `last_page` 即可。

2. 测例生成

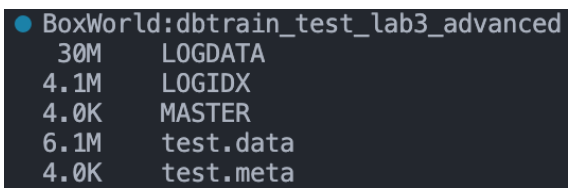
- 由于我们仅在数据库关闭时才进行垃圾回收，为了测试的便利性，我利用了本来的 `test.sh` 脚本，使得我们可以简便地测试多个测例。
- 测试的思路如下，首先建立一张表，然后往里插入 10000 条数据，最后统一删除，重复此操作 20 次即可得到较明显的性能差异，同时也测试了程序的正确性。
- 具体代码如下：

```python
import random
N = 10000

# 先生成数据库和表
with open('lab3/test/40_setup.sql', 'w') as f:
    f.write('drop database if exists dbtrain_test_lab3_advanced;\n')
    f.write('create database dbtrain_test_lab3_advanced;\n')
```

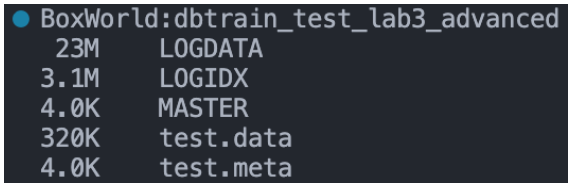
```
    f.write('use dbtrain_test_lab3_advanced;\n')
    f.write('create table test(id int, score float);\n')

with open('lab3/result/40_setup.result', 'w') as f:
    f.write('SUCCESS\n\n' * 4)
    f.write('Bye\n')

# 重复 20 轮
for i in range(0, 20):
    success_count = 0
    with open('lab3/test/{}_very_large_table.sql'.format(i + 41), 'w')
as f:
        f.write('use dbtrain_test_lab3_advanced;\n')
        success_count += 1

        bias = 100
        # 先把 10000 条数据插入数据库中
        for idx in range(0, N, bias):
            f.write('insert into test values')
            f.write(','.join(['(%d, %f)' % (idx + ofs,
random.random()) for ofs in range(0, bias)]))
            f.write(';\n')
            success_count += 1

        # 最后统一删除即可
        f.write('delete from test where id < {};\n'.format(N))
        success_count += 1

    with open('lab3/result/{}_very_large_table.result'.format(i + 41),
'w') as f:
        f.write('SUCCESS\n\n' * success_count)
        f.write('Bye\n')
```

3. 测试结果

- 在没有引入垃圾回收机制前，数据大小 `test.data` 达到了 6.1 MB 的规模[1]：



- 引入垃圾回收机制后，数据大小仅有 320KB[2]：



- 注(1)：20 轮操作后，需要 320KB × 20 ÷ 1024 = 6.25MB 的空间；
- 注(2)：这是因为已分配的页面不会被删除，以一万条数据而言，所需页面数约为 10000 ÷ 144 = 70 页。

总结

- 高级功能 Commit ID：`edb47fc61d04a241cf25235f7c87b3b3863f778c`（位于 `ch3a` 分支）
- 用时：
  - 基础功能用时 6 小时；
  - 高级功能用时 6 小时；
- 合计 12 小时。