Write Ahead Log System

---

id: 519021910861

name: huidong xu

简介

介绍

Wal-sys是一个简单的 **WAL** 系统，用于更改银行中心数据库，包含用于错误崩溃回复的重做记录。

Wal-sys 创建两个文件，"LOG" 文件用于记录操作条目，"DB" 文件用于包含所有数据库的所有改变。

前置要求

- Python 解释器和 Python 环境
- 改变权限 `chmod +x wal-sys.py`

运行

将命令放在 "cmd.in" 文件中并直接从该文件中读取输入，每一行（包括最后一行）命令后都需要有换行符，"reset" 命令代表是否抛弃之前的 "LOG" 和 "DB" 文件。

```
./wal-sys.py -reset < cmd.in
```

命令

- `begin action_id` 开启一个**可恢复动作**，action_id 是正数且唯一。
- `create_account action_id account_name starting_balance` 利用 action_id 的可恢复动作完成一个用户的创建，account_name 是用户姓名（任意字符无空格），starting_balance 是初始账户金额。
- `credit_account action_id account_name credit_amount` 向用户账户中添加金额。*这条命令会进行记录式执行。*
- `debit_account action_id account_name debit_amount` 向用户账户减少金额。*这条命令会进行记录式执行。*
- `commit action_id` 提交该**可恢复动作**，并记录下一条 `commit` 记录。
- `checkpoing` 记录下一条 `checkpoint`记录，用于 debug？
- `end action_id` 结束该**可恢复动作**，将所有**可恢复结果**写入数据库 "DB"，并记录下一条 `end` 记录。
- `show_state` 显示当前 "LOG" 和 "DB" 文件内容，用于 debug。
- `crash` 系统崩溃并退出，是本次 lab 唯一退出系统的方式。

知识点回顾 - **Journaling**

- Write-ahead Logs：A concept from database
- Record before update
    a. Record changes in journal
    b. commit journal
    c. update
- crash before commit: No data is changed, discard journal.
- crash after commit: Journal is complete, redo changes in journal.

Using wal-sys

我将命令写入文件 `cmd1.in` 中，每次运行 `./wal-sys.py -reset < cmd1.in` 。

Q1: What do you boserve in on-disk DB contents? Why doesn't the database show studentC?

我在数据库文件 "DB" 中只看到了 $studentA$ 账户中有 1000 余额，而没有 $studentB$ 和 $studentC$ 的信息。数据库不显示 $studentC$ 的信息是因为 $action_id$ 为 3 的进程还没有结束，即没有进行 `end action_id` 操作，所以数据还没有写入数据到磁盘上。

```
----------------------
On-disk DB contents:
Account: studentA Value: 1000
----------------------
```

Q2: Which account should exist, and what values should they contain when the database recovers?

理论上分析，$studentA$、$studentB$ 的账户信息应该在系统恢复后存在且分别为 1100 和 2000 ，而 $studentC$ 的账户信息则不应该存在。因为 $studentA$ 账户是在系统崩溃前完成了所有操作包括写入磁盘，所以本身就应该存在。而 $studentB$ 的账户所在的 $action$ 在系统崩溃前已经 `commit` 了，所以在系统恢复后应该 `redo` 所有相关记录，因此也应该存在。$studentC$ 账户同理，因为没有进行 `commit` 操作，所以在系统恢复后应该 `undo` 所有相关记录，因此不应该存在。

Q3: Can you explain why the "DB" file does not contain a record for $studentB$ and contains the balance for $studentA$ before `credit_account` is issued?

因为任何 $action$ 在 `end action_id` 操作前的操作都不会真正写入磁盘，而是写入 "LOG" 文件便于系统崩溃后恢复。而 $studentB$ 的 `create_account` 和 $studentA$ 的 `credit_account` 操作对应的相应 $action$ 都还没有 `end`，因此在 "DB" 文件中没有相应数据。

Recovering the database

我将这一块的命令写入文件 `cmd2.in` 中，并连续运行两条命令。

```
./wal-sys.py -reset < cmd1.in
./wal-sys.py < cmd2.in
```

Q4: What do you expect the state of "DB" to be after wal-sys recovers? Why?

系统恢复后，预想中 "DB" 文件内容是：

$$studentB = 2000$$
$$studentA = 1100$$

和实际恢复后 "DB" 文件相符。

```
-------------------------
On-disk DB contents:
Account: studentB Value: 2000
Account: studentA Value: 1100
-------------------------
```

原因：分别分析 $studentA$ 和 $studentB$ 和 $studentC$ 账户。

- 对于 $studentA$ 来说，`create_account` 操作所在 `action (id = 1)` 已经 `end`，所以 $studentA$ 初始余额为 1000。且 `credit_account` 操作所在 `action (id = 2)` 已经 `commit` 了，即系统恢复后会 `redo` 这个操作，因此 $studentA$ 余额会增加 100 。所以现在 $studentA$ 在 "DB" 中显示的余额是 1100 。
- 对于 $studentB$ 来说，`create_account` 操作所在 `action (id = 2)` 已经

`commit` 了，即系统恢复后会 `redo` 这个操作。因此 $studentB$ 初始余额为 2000 。

- 对于 $studentC$ 来说，`create_account` 操作和 `debit_account` 操作所在 `action (id = 3)` 还没有 `commit` ，因此系统恢复后会 `undo` 这些操作，因此 "DB" 文件中不存在 $studentC$ 账户信息。

Q5: If you issue another wal-sys command to recover the database again, what would the "DB" file contain after the second recovery? Why?

预想中和实际操作后的结果一致，即无论重复连续恢复数据库多少次，"DB" 文件内容都和第一次恢复后是一样的。因为写入 "LOG" 文件的操作都应该是幂等的，例如赋值操作等而不能是自增操作等，这样可以保证数据的恢复与恢复次数无关。

Q6: During recovery, wal-sys reports the $action\_ids$ of those recoverable actions that are "Losers", "Winners", and "Done". What is the difference between these categories?

我们根据 `commit` 操作和 `end` 操作来划分 $action$ 的角色。

- $action$ 还没有 `commit` 也没有 `end` ，则它是 "Loser"。字面意义上理解就是它是个失败者，即系统恢复时应该撤销有关它的所有操作，即 `undo` 它在 `begin` 之后进行的操作。
- $action$ 已经 `commit` 但还没有 `end` ，则它是 "Winner"。字面意义上理解就是它是个成功者，即系统恢复时需要重做有关它的所有操作，即 `redo` 它在 `begin` 到 `commit` 之间的所有操作。
- $action$ 已经 `commit` 且已经 `end` ，则它是 "None"。字面意义上理解就是它是虚无的，即系统恢复时不需要考虑它的任何操作，因为它的操作已经在系统崩溃前全部写入 "DB" 文件了。