**Lab 7 - Transactions and Spark**

**Transactions**

This section contains exercises on transactions. Here are some refreshers:

* Transaction is a group of statement executed together as one indivisible unit.
* Two important properties are **A**tomicity and **I**solation.
  + Atomicity: the statements inside the transaction are executed in all-or-nothing manner.
  + Isolation: multiple transactions are executed as if one after another.
* Write-Ahead-Logging is a technique for ensuring atomicity
  + Operation is written to a log first
  + The log is flushed to disk before the transaction commits.
  + Undo logging:
    - All data is flushed to disk before a COMMIT message is written to the log on disk.
  + Redo logging:
    - Data doesn't have to be flushed to disk when COMMIT message is written to the log on disk.
    - Data is flushed at a later time, e.g., when disk is not so busy.
* Once the COMMIT message appears in the log on disk, the transaction is considered as committed

**We assume very simple model for WAL**: there's only one single thread of execution. Also, the transaction does not deliberately abort.

* **Serializability** is the standard for isolation. An interleaving execution of multiple transaction is serializable when the final states of all the transactions can be achieved by one serial execution of the transactions. For example, consider 2 transaction T1, T2.
  + There are two serial execution orders: O1 = (T1 then T2) and O2 = (T2 then T1)
  + Any execution of T1 and T2 whose results is one of O1 or O2 is considered serializable.
* We **cannot** always check if an execution sequence is serializable.
* We can check if an execution sequence is conflict serializable.
  + If we can swap order of any two operation in the sequence without changing the conflict set, then the sequence is conflict serializable.
* If a sequence is conflict serializable, then it is serializable.
* Two phase locking (2PL): once you release any lock, you cannot acquire more locks.
* Strict two phase lock (S2PL): locks are released only when transaction finished (committed or aborted).
* Deadlock is a possibility, even with S2PL.

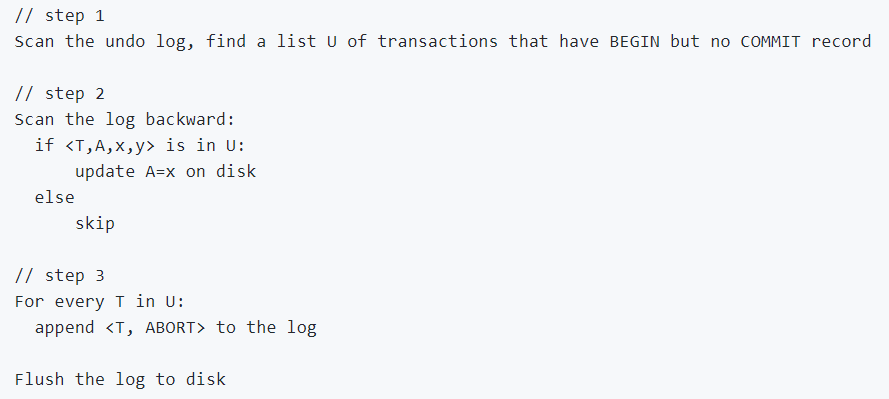
### Exercise 1

Given the content of the WAL below. Before this, the content on disk is A=0, B=0, C=0.

|  |  |
| --- | --- |
|  | **[Q1]** **Suppose undo logging is used. What are the content of A, B, C on disk:**  **[Q1a] If a crash happened at the end of line 10 and the system recovered.** - (Only T1 and T2 have been committed, so recovery must recover the changes they made. A=50, B=20, C=30 )  **[Q1b] If a crash happened at the end of line 7 and the system recovered.** - (Only T1 has been committed, so recovery must recover the change T1 made. A=10, B=20, C=0 )  **[Q2]** **Suppose redo logging is used. What are the content of A, B, C on disk:**  **[Q2a] Before the system crashes at line 10.**  (Since there's no FLUSHED message, the commited values of T1 may not have been persisted to disk. Note that T1 must be flushed before T2.  One answer is: A=0, B=0, C=0 Other possible answers: [A=10, B=0, C=0], [A=10, B=20, C=0]. Because T1 is in the middle of being flushed, and we don't know which values have been flushed yet. )  **[Q2b] Before the system crashes at line 7.**  (Same as crashing at line 10) |

### Exercise 2

A recovery algorithm for undo logging is given below.



**[Q1] Does this algorithm work if it fails in the middle of step 2? Why?**

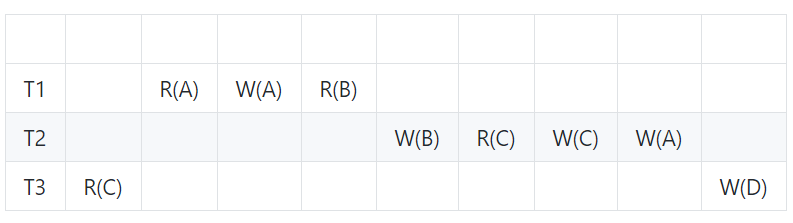
Yes, because it will reapply old values.

**[Q2] Does this algorithm work if step 2 scans forward instead of backward? Why?**

No, in the example above, when failing at line 7, if scanning forward the algorithm will update A to 40, which is not correct

### Exercise 3

Given the following execution schedule. Time goes from left to right.

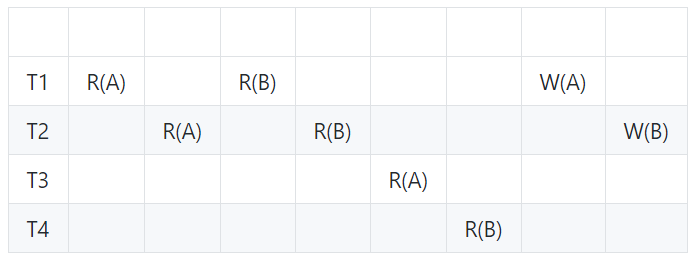


**[Q1] Is this schedule conflict serializable?**

Yes, move W(D) to the left. T3, T1, T2.

### Exercise 4

Given the following execution schedule. Time goes from left to right.

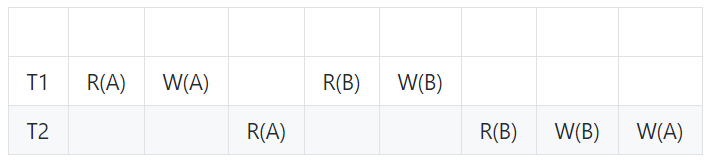


**[Q1] Is this schedule conflict serializable?**

No, operations on T1 and T2 cannot be moved. Moving T2 before T1 is not possible, because of the conflict order on B. Likewise, moving T2 after T1 is not possible, due to conflict order on A

### Exercise 5

Given the following execution schedule. Time goes from left to right.



Suppose each transaction commits right after the last operation.

**[Q1] Is this schedule possible under 2PL? If yes, insert lock and unlock operation at appropriate places.**

Yes, T1 locks both A and B first, then releases A after W(A), and releases B() after W(B). T2 acquires as soon as locks are released.

**[Q2] Is this schedule possible under strict 2PL? If yes, insert lock and unlock operation at appropriate places.**

No, locks on A and B cannot be released until last operation of T1, but T2 is waiting.