Homework: Concurrency Control

-Rahul Kumar Nalubandhu
-nalubanr@oregonstate.edu

1. Consider the following classes of schedules: serializable and 2PL. For each of the following schedules, state which of the preceding classes it belongs to. If you cannot decide whether a schedule belongs in a certain class based on the listed actions, explain briefly. Also, for each 2PL schedule, identify whether a cascading rollback (abort) may happen. A cascading rollback will happen in a schedule if a given transaction aborts at some point in the schedule, and at least one other transaction must be aborted by the system to keep the database consistent (2 points).

   The actions are listed in the order they are scheduled and prefixed with the transaction name. If a commit or abort is not shown, the schedule is incomplete; assume that abort or commit must follow all the listed actions.

   1. T1:R(X), T2:R(Y), T3:W(X), T2:R(X), T1:R(Y)

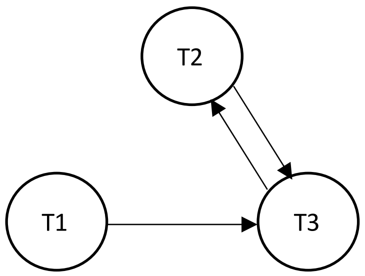
      The serialization graph, T1 → T3 → T2

      After looking at the data and the order of events, it's clear that there aren't any repeating patterns in the transactions. So, we can line them up one after another, just like in a Two-Phase Locking (2PL) schedule. Each transaction in this schedule has a part where it grows and a part where it shrinks.

T1	T2	T3
Shared lock (X) Shared lock (Y) R(X) Release lock (X)		
	Shared lock (X) R (Y)	Exclusive Lock (X) W(X) Release lock (X)
	Shared lock (X) R(X) Release lock (Y) Release lock (X)	
R(Y) Release lock (Y)		

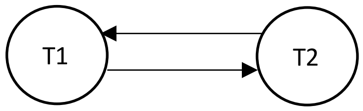
In this Two-Phase Locking (2PL) schedule, there could be a chain reaction of rollbacks or cancellations. If transaction T3 with W(X) gets cancelled, transaction T2 has to be cancelled too. This is because T2 read the new value of X - R(X) that was updated in transaction T3.

2. T1:R(X), T1:R(Y), T1:W(X), T2:R(Y), T3:W(Y), T1:W(X), T2:R(Y)



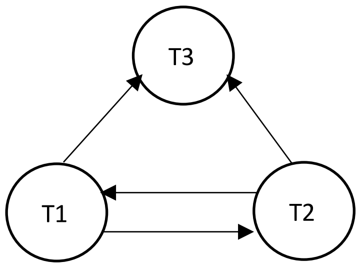
The chart showing the order of transactions has a repeating pattern or loop between transactions T2 and T3. This means we can't line them up one after another, so it's not a (2PL) schedule.

3. T1:W(X), T2:R(X), T1:W(X)



The chart showing the order of transactions has a repeating pattern or loop between transactions T1 and T2. This means we can't line them up one after another, so it's not a (2PL) schedule.

4. T1:R(X), T2:W(X), T1:W(X), T3:R(X)



The chart showing the order of transactions has a repeating pattern or loop between transactions T1 and T2. This means we can't line them up one after another, so it's not a (2PL) schedule.

2. Consider the following schedule.

	T1	T2
0	start	
1	read X	
2	write X	
3		start
4		read X
5		write X
6		Commit
7	read Y	
8	write Y	
9	Commit	

What are the maximum degrees of consistency for T1 and T2 in this schedule? You must find the maximum degrees of consistency for T1 and T2 that makes this schedule possible (1 point).

Looking at the details provided, we see that Transaction T1 begins first, reading and writing on item X. However, Transaction T2 can also start, read, and write on the same item X before T1 is complete. This means T1 doesn't hold onto item X for very long. It only has a short hold on X, which means its consistency level is 0. According to the schedule, no other transaction than T1 is using X before T2 finishes. So, T2 can hold onto item X for a long time, both in a shared and exclusive manner. Therefore, the highest level of consistency for transaction T2 is 3.

3. Consider the following protocol for concurrency control. The database system assigns each transaction a unique and strictly increasingly id at the start of the transaction. For each data item, the database system also keeps the id of the last transaction that has modified the data item, called the transaction-id of the data item. Before a transaction T wants to read or write on a data item A, the database system checks whether the transaction-id of A is greater than the id of T . If this is the case, the database system allows T to read/write A. Otherwise, the database system aborts and restarts T.

   (a) Does this protocol allow only serializable schedules for transactions? If not, you may suggest a change to the protocol so that all schedules permitted by this protocol are serializable. You should justify your answer. (1.5 points)

   No. This protocol does not prevent cycles in the serialization graph. The given protocol can lead to problems where a transaction reads or writes outdated data, causing incorrect results. This is because the protocol allows transactions to read or write data even if another transaction is still working with it. To make sure that the system only permits schedules where transactions don't interfere with each other (known as serializable schedules), we need to change the protocol.

   In the new protocol, a transaction can only read or write data if it has a higher id than the data's transaction-id. Also, when a transaction writes to a data item, it updates the data's transaction-id to its own id. This ensures a transaction only works with data from transactions that have fully finished and prevents conflicts. If a transaction tries to read or write data and it doesn't have a higher id than the data's transaction-id, the system shouldn't just restart it, but also give it a new id. This is to stop the transaction from getting stuck in a loop where it keeps getting restarted because its id is always lower.

   (b) Propose a change to this protocol or the modified version you have designed for part (a) that increases its degree of concurrency, i.e., it allows more serializable schedules. (1.5 points)

   In the existing protocol, read operations from different transactions on the same item need to happen in a specific order. However, in general, two read operations on the same item from different transactions don't conflict with each other, so the order doesn't really matter. Instead of forcing a particular order, we can treat read and write transactions separately for each item. If there's a conflict (when the transaction ids aren't in a specific order), then we restart the particular transaction.