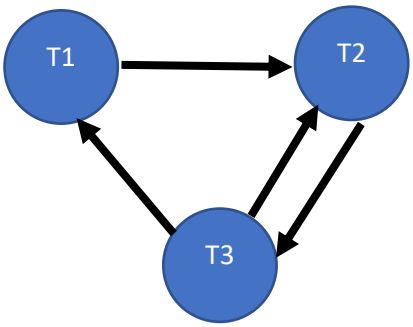
Practicum 3

Transactions Part

Question 1a: Draw the precedence graph of schedule S_2.

Answer 1a:



Question 1b: Is S_2 conflict serializable? Explain.

Answer 1b: No it is no conflict serializable because the precedence graph shows that S_2 is cyclic.

Question 1c: If is S_2 conflict serializable, write down all equivalent serial schedules.

Answer 1c: Not applicable.

Question 1d: If is S_2 not conflict serializable, make the schedule conflict serializable. Show all working required to make this schedule conflict serializable, include the new precedence graph and changes made to the schedule.

Answer 1d:

Step 1: Original schedule

	S2	
T1	T2	T3
read(P)		
	read(R)	
		read(P)
read(R)		
	read(Q)	
		read(Q)
write(P)		
	write(R)	
		write(Q)
	write(Q)	

Step 2: Swap T2 read(Q) and T3 read(Q)

	S2	
T1	T2	T3
read(P)		
	read(R)	
		read(P)
read(R)		
		read(Q)
	read(Q)	
write(P)		
	write(R)	
		write(Q)
	write(Q)	

Step 3: Swap T2 read(Q) and T1 write(P)

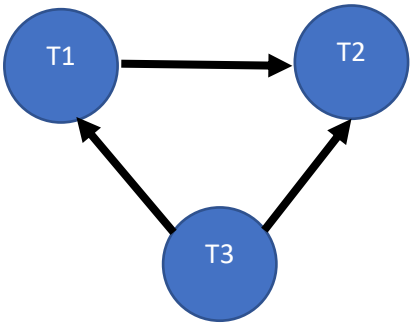
	S2	
T1	T2	T3
read(P)		
	read(R)	
		read(P)
read(R)		
		read(Q)
write(P)		
	read(Q)	
	write(R)	
		write(Q)
	write(Q)	

Step 4: Swap T2 read(Q) and T2 write(R)

	S2	
T1	T2	T3
read(P)		
	read(R)	
		read(P)
read(R)		
		read(Q)
write(P)		
	write(R)	
	read(Q)	
		write(Q)
	write(Q)	

Step 5: Swap T2 read(Q) and T3 write(Q)

	S2	
T1	T2	T3
read(P)		
	read(R)	
		read(P)
read(R)		
		read(Q)
write(P)		
	write(R)	
		write(Q)
	read(Q)	
	write(Q)	



Question 2a: For the program specified above, what atomicity problems could arise if it was not run as a transaction?

Answer 2a: There will be no atomicity problems.

There are a few scenarios that exist. One scenario is if the user login already exists in the table, an error message will be displayed and the rest of the procedure will not be executed. So the entire procedure fails and no new values are inserted into the users table.

One scenario is that if the user login does not exist in the table, the else statement would run and assign the user inputs to variables, this is then followed by line 7 which has a CHECK constraint so if the password fails the CHECK constraint, no new values are inserted into the users table.

Another scenario is that if the user login does not exist in the table, the else statement would run and assign the user inputs to variables, this is then followed by line 7. If the password passes the CHECK constraint, the new values are inserted into the users table.

From the above scenarios, the whole procedure would either fully execute or not execute at all, therefore there are no atomicity problems. However, it would be prudent to place the INSERT statement in line 7 under the else statement in line 5 rather than outside of the else statement.

Question 2b: For the program specified above, what isolation problems could arise if it was not run as a serializable transaction?

Answer 2b: If it was not run as a serializable transaction, it could run into the problems of dirty reads, non-repeatable reads and phantom read.

Dirty reads occurs when a transaction is allowed to read data from a row that has been modified by another running transaction and not yet committed. Non-Repeatable reads occurs when, during the course of a transaction, a row is retrieved twice and the values within the row differ between reads. Phantom reads occurs when, in the course of a transaction, new rows are added or removed by another transaction to the records being read.

This would potentially allow different transactions to read the users table and have different conclusion as to whether the login already exists in the users table or not. This might potentially allow the INSERT task to run when it should not but it can still abort if the login already exists in the users table because of the PRIMARY KEY constraint placed on login which means it must be unique.

Question 2c: Compare what would happen if the program above was run as a transaction with isolation level **SERIALIZABLE** compared to if it was run with isolation level **READ COMMITTED**. Point out benefits and drawbacks of the two choices and give a recommendation for the best isolation level for this particular problem.

Answer 2c: If the program was running as a transaction with isolation level SERIALIZABLE, it will not run into problems like dirty reads, non-repeatable reads, phantom reads. However, because the transactions cannot be run concurrently, it would result in slower overall speed of transactions especially if you have short transactions, running them serially will make short transactions wait too long to complete. It can also result in deadlocks which causes transactions to be even slower as you have to wait for the lock wait timeout to finish.

If the program was running as a transaction with isolation level READ COMMITTED, it will allow for transactions to be run concurrently which helps improve throughput and resource utilization and reduce waiting time. However, it will run into problems like non-repeatable reads, phantom reads. If it does run into such problems, it will usually require rollbacks which requires additional time to correct for resulting in lower overall efficiency.

One particular concern might be that if there are 2 transactions, and both transaction reads that a particular login does not exist in the users table, both transactions will allow the insert into the table, this might seem like a problem, however because the DDL lists the login as a primary key, if both transactions try to enter the same login to register as a new user, the transaction that commits first will have the new login registered and when the other transaction tries to commit, it will fail and abort and rollback. So although it does cause rollbacks, it will not cause a problem with the users table. Therefore although READ COMMITTED still can cause rollbacks, it is still preferable as it allows concurrency.