Task 1

1. RW conflict:

An exchange T2 could change the worth of an item A that has been perused by an exchange T1, while T1 is as yet in progress. The present circumstance messes two up.

First: if T1 attempts to peruse the worth of A once more, it will get an alternate outcome, despite the fact that it has not changed A meanwhile. The present circumstance couldn't emerge in a sequential execution of two exchanges; it is called an unrepeatable read.

Second: assume that both T1 and T2 read a similar worth of A, for example 5, and afterward T1 which needs to augment A by 1 changes it to 6 and T2 which needs to decrement A by 1 decrements the worth that it read (5) and changes a to . Running these exchanges in any chronic request ought to have A with a last worth of ; in this manner, the interleaved execution prompts a conflicting state. The hidden issue here is that in spite of the fact that T2's change isn't straightforwardly perused by T1, it nullify T1's supposition about the worth of A, which is the reason for a portion of T1's resulting activities

2. Assume that Harry and Larry are two representatives, and their pay rates should be kept equivalent. T1 sets their compensations to 1000 and T2 sets their pay rates to 2000. On the off chance that we execute these in the chronic request T1 followed by T2, both get the compensation 2000, the chronic request T2 followed by T1 gives each the compensation 1000. Both is adequate from a consistency outlook.

Task 3

Prove that the basic two-phase locking protocol guarantees conflict serializability of schedules. (Hint: Show that, if a serializability graph for a schedule has a cycle, then at least one of the transactions participating in the schedule does not obey the two-phase locking protocol.)

```
read_lock (X):
B: if LOCK (X)="unlocked"
then begin LOCK (X) <- "read-locked";
no_of_reads(X) <- 1
End
else if LOCK(X)="read-locked"
then no_of_reads(X) <- no_of _reads(X) + 1
else begin
```

```
wait (until LOCK (X)="unlocked" and
the lock manager wakes up the transaction);
go to B
end;
write_lock (X);
B: if LOCK (X)="unlocked"
then LOCK (X) <-"write-locked"
else begin
wait (until LOCK(X)="unlocked" and he lock manager wakes up the transaction);
go to B
end;
unlock_item (X):
if LOCK (X)="write-locked"
then begin LOCK (X) "unlocked;“
wakeup one of the waiting transactions, if any
End
else if LOCK(X)="read-locked"
then begin
no_of_reads(X) <- no_of_reads(X) –1;
if no_of_reads(X)=0
then begin LOCK (X)="unlocked";
wakeup one of the waiting transactions, if any
End
End;
```

Task 4

Assume that a non-serializable (non conflict) plan S for t1,t2,….,tn does happen. At that point the serialization chart for S must have a cycle. So there must be a few arrangement inside the plan of the frame: S: …; [o1(X);l …; o2(X);]…; [o2(Y)]…; …; o3(Y);] [on(Z);…;o1(Z);]…

Where each combine of operations between square brackets [o,o] are clashing ([w,r] or [r,w] or [w,w]) and needs to bolt thing Z (some time recently applying o1(Z) but must happen after Tn has opened it). Hense, a arrangement in T1 of the taking after shape happens: t1: …; o1(X); …; open (X); …; o1(Z); …

This suggests that t1 does not comply the two-phase locking convention (since lock(Z) takes after unlock(X)) contradicting our presumption that all exchanges in S take after the 2PL convention.