Exercise 5 Solution

1. Isolation property in ACID properties states that each transaction should run without interfering or being aware of other transactions in the system. In that case, we can run each transaction sequentially by locking the whole database for itself. Review why this may not be an ideal solution.

   Solution:

   This is in fact one type of, simplistic, concurrency control. That is, making sure that all transactions run in a sequence, in some order. But then we are not benefiting from the potential use of idle resources such as accessing disk while another transaction is using the CPU. So even under single CPU situations, concurrency can speed up things that we are not doing. In addition, if there is a long-running transaction in the system, holding up every other transaction till that long one finish is going to make the associated company's customers very unhappy. So in short, although we want our executions to be equal to the outcome of a serial execution of a given set of transactions, we do not really want to run them one after the other but rather in concurrency with each other for efficiency.

2. Given two transactions, per operation of each transaction, we can use locks to make sure concurrent access is done properly to individual objects that are used in both transactions i.e., they are not accessed at the same time. This is after all what the operating systems do, e.g., lock a file while one program is accessing it so others cannot change it at the same time. Give two transactions showing that this is not enough to achieve the isolation property of transactions for RDBMSs.

   Solution:

Answer (A is initially 0 on disk)	Transaction 1 at Process 1	Transaction 2 at Process 2
Operation 1	Lock(A)	
Operation 2	Read(A)	
Operation 3	Unlock(A)	
Operation 4	A = A +100	
Operation 5		Lock(A)
Operation 6		Read(A)
Operation 7		A = A + 50
Operation 8		Write(A)
Operation 9		Unlock(A)/Commit
Operation 10	Lock(A)	
Operation 11	Write(A)	
Operation 12	Unlock(A)/Commit	

   This execution order above is not equal to T1 running first nor to T2 running first and does not obey the Isolation property, although, for each disk access, we first obtained a lock properly. Thus we need something more for proper concurrency control in DBMSs.

3. What is the probability that a deadlock situation occurs?

   Solution: It can also be found in Section 7.11.5 of the reference book - Transaction Processing, Jim Gray and Andreas Reuter, Morgan Kaufmann, 1992.

   Assume, Number of transactions = n+1 ≈ n (if n is large).

   Each transaction access r number of locks exclusively

   Total number of records in the database = R

   On average, each transaction is holding r/2 locks approximately (minimum 0 and maximum r, hence average is r/2) – transaction that just commenced holds 0 locks, transaction that is just about to finish holds r locks.

   Average number of locks taken by the other n transactions = $n \times (r/2) =$

   The probability that a particular transaction waits for a lock = Requesting an exclusive lock on one of the nr/2 locks held by the other n Transactions out of R possible locks = $\frac{nr}{2}$ out of potential R records = $\frac{nr}{2R}$

   The probability that a particular transaction waits for a lock = (nr)/2R

   The probability that a particular transaction did not wait for a lock = 1- (nr)/2R

   The probability that a particular transaction did not wait (for any of the r locks), P = (1- (nr)/2R)^r ≈ 1- (nr2/2R) [note that $(1+\epsilon)^r = 1 + r \times \epsilon$ when ϵ is small]

   The probability that a transaction waits in its life time pw(T) = 1-P = 1–{$1 - (nr^2/2R)$} = $nr^2/2R$

   Probability that a particular Transaction T waits for some transaction T1 and T1 waits for T is = pw(T)* (pw(T1)/n) = $nr^4/4R^2$ [since the probability T1 waits for T is 1/n times the probability T1 waits for someone]

   Probability of any two transactions causing deadlock is = $n \times nr^4/4R^2 = n^2r^4/4R^2$ (This is a very small probability in practice)

4. Given the following two tasks where initial value of B is 50, and operations are labeled as Op1, Op2, etc, per task and we know that Task 1 has done its first operation already, please show all possible concurrent execution orders of the following tasks. Then state which orders make sense and which ones do not. If these were two transactions running concurrently this way, what property of transactions we would be violating with the invalid concurrent execution orders? The tasks are:

   - Task 1:
     - Op1) $B = B \times 2$
     - Op2) $B = B \times 2$
   - Task 2:
     - Op1) $B = B + 10$

   Solution:

   The orders are:
   Task 1 Op1, Task 1 Op2, Task 2 Op1; OR
   Task 1 Op1, Task 2 Op1, Task 1 Op2.

   The output of the first order is 210 for B and 220 for the second order. Only 210 makes sense because if Task 1 was done first and then Task 2 we would get 210 (and if Task 2 was done first and then Task 1 then we would get 240 and still not 220). We are not adhering to the Isolation property of transactions and the two transactions running in a peculiar way concurrently and it is the only way we can get 220 and hence they know about each other!