**Why do we need Concurrency, anyway? Explain.**

*Simple. Imagine two users try to modify same data in different ways at the same time, and then the other user wants to access the data. What happens afterwards is anyone’s guess.*

*There is no way to apply their changes to the same data and same place. If DBMS allows both of their actions concurrently, it would be impossible to recognize which data is updated. Or, if DBMS choose one of the changes, the other one becomes meaningless. Then, it is sure to affect the results and confuses users as well since there is no error message or notification for this issue. Both of solutions make the database inconsistent and useless. This is why concurrency control is needed.*

*Actually, there is no “same time actions” in Computer Science so DBMS accepts one request and blocks or rejects the other one. This is the simplest way to make database concurrent. Most relational DBMS supports this function.*

*But it decrease performance of concurrent tasks because one task may be blocked by other one and in waiting status until it finished (and because it controls “concurrency”, literally!). Control of data concurrency and data consistency is critical as concurrent data execution need to produce meaningful and consistent results. If you don't need data to be updated consistently or you don't have any plan for UPDATE operation on that data but you want to do concurrent tasks quickly, you can choose DBMS product that supports disabling concurrency control or optimized only for concurrent*

**Why is testing multithreading / concurrent code so difficult?**

*If you can count on any mathematical experience, illustrate how a normal execution flow that is essentially deterministic becomes not just nondeterministic with several threads, but exponentially complex, because you have to make sure every possible interleaving of machine instructions will still do the right thing. A simple example of a lost update or dirty read situation is often an eye-opener.*

*"Slap a lock on it" is the trivial solution... it solves all your problems if you're not concerned about performance. Try to illustrate how much of a performance hit it would be if, for instance, Amazon had to lock the entire east coast whenever someone in Atlanta orders one book!*

**What is a Race Condition? Code an example, using whatever language you like.**

*A race condition occurs when two or more threads can access shared data and they try to change it at the same time. Because the thread scheduling algorithm can swap between threads at any time, you don't know the order in which the threads will attempt to access the shared data. Therefore, the result of the change in data is dependent on the thread scheduling algorithm, i.e. both threads are "racing" to access/change the data.*

*Problems often occur when one thread does a "check-then-act" (e.g. "check" if the value is X, then "act" to do something that depends on the value being X) and another thread does something to the value in between the "check" and the "act". E.g:*

*if (x == 5) // The "Check"*

*{*

*y = x \* 2; // The "Act"*

*// If another thread changed x in between "if (x == 5)" and "y = x \* 2" above,*

*// y will not be equal to 10.*

*}*

*The point being, y could be 10, or it could be anything, depending on whether another thread changed x in between the check and act. You have no real way of knowing.*

*In order to prevent race conditions from occurring, you would typically put a lock around the shared data to ensure only one thread can access the data at a time. This would mean something like this:*

*// Obtain lock for x*

*if (x == 5)*

*{*

*y = x \* 2; // Now, nothing can change x until the lock is released.*

*// Therefore y = 10*

*}*

*// release lock for x*

*https://stackoverflow.com/questions/34510/what-is-a-race-condition*

**What is a Deadlock? Would you be able to write some code that is affected by deadlocks?**

**What is Process Starvation? If you need, let's review its definition.**

*A lock occurs when multiple processes try to access the same resource at the same time.*

*One process loses out and must wait for the other to finish.*

*A deadlock occurs when the waiting process is still holding on to another resource that the first needs before it can finish.*

*So, an example:*

*Resource A and resource B are used by process X and process Y*

*X starts to use A.*

*X and Y try to start using B*

*Y 'wins' and gets B first*

*now Y needs to use A*

*A is locked by X, which is waiting for Y*

*The best way to avoid deadlocks is to avoid having processes cross over in this way. Reduce the need to lock anything as much as you can.*

*In databases avoid making lots of changes to different tables in a single transaction, avoid triggers and switch to optimistic/dirty/nolock reads as much as possible.*

**What is a Wait Free algorithm**

*If a program is lock-free, it basically means that at least one of its threads is guaranteed to make progress over an arbitrary period of time. If a program deadlocks, none of its threads (and therefore the program as a whole) cannot make progress - we can say it's not lock-free. Since lock-free programs are guaranteed to make progress, they are guaranteed to complete (assuming finite execution without exceptions).*

*Wait-free is a stronger condition which means that every thread is guaranteed to make progress over an arbitrary period of time, regardless of the timing/ordering of thread execution; and so we can say that the threads finish independently. All wait-free programs are lock-free.*

*I don't know offhand of any Java examples which illustrate this but I can tell you that lock-free/wait-free programs are typically implemented without locks, using low-level primitives such as CAS instructions.*