Deadlock and its avoidance

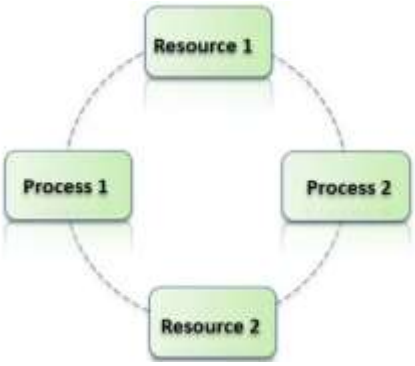
A Deadlock in DBMS can be termed as the undesirable condition which appears when a process waits for a resource indefinitely whereas the same resource is held by another process.

In order to understand the deadlock concept better, let us consider a transaction T1 which has a lock on a few rows in the table Employee and it requires to update some rows in another table Salary. Also, there exists another transaction T2 that has a lock on the table Salary and it also requires updating a few rows in the Employee table which already is held by the transaction T1.

In this situation both the transactions wait for each other to release the lock and the processes end up waiting for each other to release the resources. As a result of the above scenario, none of the tasks get completed and this is known as **deadlock**.

State of Deadlock

In the state of deadlock, no tasks get completed and they remain in the state of waiting for an indefinite time. We can understand the situation better by the below figure. Resource 2 is held by Process 1 and Process 1 needs Resource 1. Similarly, Resource 1 is held by Process 2 and Process 2 needs Resource 2. Deadlock should be avoided as because of it the entire system comes to rest.



Conditions in state of deadlock

One of the conditions is where a process waits for a resource, that is held by a second process and this second process is awaiting the third and as we go on, the last process is awaiting the first which makes the waiting a circular chain.

Another condition can be termed as Hold and Wait as the process which holds one resource can ask for extra resources that are held by the other processes.

Another condition for the occurrence of a deadlock is that a resource cannot be dynamically taken from a process because the process is only capable of freeing the resource held by it.

A deadlock may arise if at least one resource must exist which at a time cannot be used by more than one resource and this condition can be called mutual exclusion.

The handling of deadlock is costly, and it is better if we can prevent it rather than handling it.

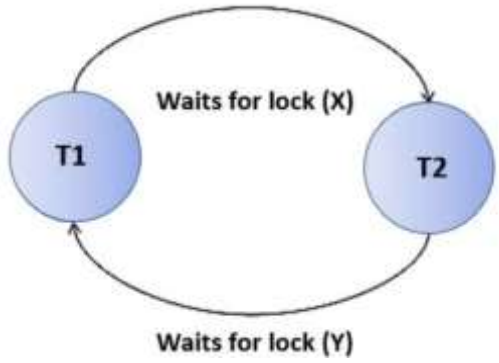
Detecting Deadlock in DBMS

The deadlock can be detected by the a who checks all the resources allocated to the different processes. The deadlock should be avoided instead of terminating and restarting the transaction so that both resources as well time is not wasted. One of the methods for detecting deadlock is Wait-For-Graph which is suitable for smaller databases.

a) Wait-For-Graph

A graph is created based on the transactions and locks on the resource in this method.

A deadlock occurs if the graph which is created has a closed-loop. For all transactions waiting the resources are maintained by DBMS and also are checked to see if there is any closed loop. Let us consider two transactions T1 and T2 where T1 requests for a resource held by T2. The wait-for-graph in this scenario draws an arrow from T1 to T2 and when the resource is released by T2, the arrow gets deleted. For example, T1 requests for a lock X on a resource, which is held by T2, a directed edge is created from T1 to T2. When T2 releases the resource X, the edge T1 locks the resource and the directed edge between T1 and T2 is dropped.



Prevention of Deadlock in DBMS?

In DBMS, all the operations are analyzed and inspected by it to find if there is any possibility of occurrence of deadlock and in case of the possibility of deadlock, the transaction is not allowed to be processed.

Primarily, the timestamp at which the transactions have begun is examined by the DMBS and so the transactions are ordered. The deadlock can be prevented by using schemes that use the timestamp of the transactions to calculate the occurrence of deadlock.

1. Wait- Die Scheme

In this scheme, when a transaction requests for the resource which is already held by another transaction, then the timestamps of the transactions are scanned by the DBMS and the older transaction waits till the resource becomes available. Let us consider two transactions T1 and T2 and the timestamp of the transaction be denoted by TS. If T1 requests for resources held by T2 and a lock exists on T2 by some other transaction,

Below are the steps followed:

Whether TS(T1) < TS(T2) is examined, and if T1 is the older transaction between T1 and T2 and some resource is held by the transaction T2, then it permits T1 to await the resource to be available for execution.

If T1 is the older transaction that has held some resource and T2 is waiting for the resource held by T1, then T2 gets killed and later it will be restarted with the same timestamp but with a random delay.

2. Wound Wait Scheme

In this scheme, if T1 is the older transaction in between transactions T1 and T2, and when T2 requests for the resource which is held by the transaction T1, then the younger transaction i.e. T2 waits until T1 releases the resource. But when the resource held by the younger transaction T2 is requested by the older transaction T1, T2 is forced by T1 to kill the transaction in order to release the resource held by it and afterward T2 is restarted with a delay but with the same timestamp.