SNS COLLEGE OF TECHNOLOGY, COIMBATORE –35

(An Autonomous Institution)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Deadlock – Transaction Recovery

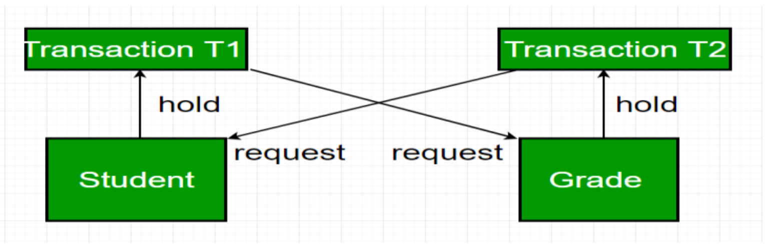
In a multi-process system, deadlock is an unwanted situation that arises in a shared resource environment, where a process indefinitely waits for a resource that is held by another process.

For example, assume a set of transactions {T_0, T_1, T_2, ...,T_n}. T_0 needs a resource X to complete its task. Resource X is held by T_1, and T_1 is waiting for a resource Y, which is held by T_2. T_2 is waiting for resource Z, which is held by T_0. Thus, all the processes wait for each other to release resources. In this situation, none of the processes can finish their task. This situation is known as a deadlock.

Deadlocks are not healthy for a system. In case a system is stuck in a deadlock, the transactions involved in the deadlock are either rolled back or restarted.

Example – let us understand the concept of Deadlock with an example : Suppose, Transaction T1 holds a lock on some rows in the Students table and **needs to update** some rows in the Grades table. Simultaneously, Transaction **T2 holds** locks on those very rows (Which T1 needs to update) in the Grades table **but needs** to update the rows in the Student table **held by Transaction T1**.

Now, the main problem arises. Transaction T1 will wait for transaction T2 to give up the lock, and similarly, transaction T2 will wait for transaction T1 to give up the lock. As a consequence, All activity comes to a halt and remains at a standstill forever unless the DBMS detects the deadlock and aborts one of the transactions.



Deadlock in DBMS

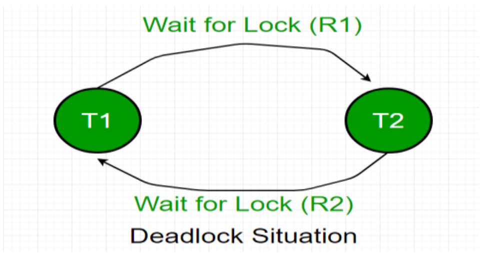
Deadlock Avoidance: When a database is stuck in a deadlock, It is always better to avoid the deadlock rather than restarting or aborting the database. The deadlock avoidance method is suitable for smaller databases whereas the deadlock prevention method is suitable for larger databases.

One method of avoiding deadlock is using application-consistent logic. In the above-given example, Transactions that access Students and Grades should always access the tables in the same order. In this way, in the scenario described above, Transaction T1 simply waits for transaction T2 to release the lock on Grades before it begins. When transaction T2 releases the lock, Transaction T1 can proceed freely.

Another method for avoiding deadlock is to apply both the row-level locking mechanism and the READ COMMITTED isolation level. However, It does not guarantee to remove deadlocks completely.

Deadlock Detection: When a transaction waits indefinitely to obtain a lock, The database management system should detect whether the transaction is involved in a deadlock or not. **Wait-for-graph** is one of the methods for detecting the deadlock situation. This method is suitable for smaller databases. In this method, a graph is drawn based on the transaction and its lock on the resource. If the graph created has a closed loop or a cycle, then there is a deadlock.

For the above-mentioned scenario, the Wait-For graph is drawn below:



Deadlock prevention: For a large database, the deadlock prevention method is suitable. A deadlock can be prevented if the resources are allocated in such a way that a deadlock never occurs. The DBMS analyzes the operations whether they can create a deadlock situation or not, If they do, that transaction is never allowed to be executed.

The following table lists the differences between Wait – Die and Wound -Wait scheme prevention schemes:

Wait – Die	**Wound -Wait**

Wait – Die	**Wound -Wait**
It is based on a non-preemptive technique.	It is based on a preemptive technique.
In this, older transactions must wait for the younger one to release its data items.	In this, older transactions never wait for younger transactions.
The number of aborts and rollbacks is higher in these techniques.	In this, the number of aborts and rollback is lesser.

Features of deadlock in a DBMS:

Mutual Exclusion: Each resource can be held by only one transaction at a time, and other transactions must wait for it to be released.

Hold and Wait: Transactions can request resources while holding on to resources already allocated to them.

No Preemption: Resources cannot be taken away from a transaction forcibly, and the transaction must release them voluntarily.

Circular Wait: Transactions are waiting for resources in a circular chain, where each transaction is waiting for a resource held by the next transaction in the chain.

Indefinite Blocking: Transactions are blocked indefinitely, waiting for resources to become available, and no transaction can proceed.

System Stagnation: Deadlock leads to system stagnation, where no transaction can proceed, and the system is unable to make any progress.

Inconsistent Data: Deadlock can lead to inconsistent data if transactions are unable to complete and leave the database in an intermediate state.

Difficult to Detect and Resolve: Deadlock can be difficult to detect and resolve, as it may involve multiple transactions, resources, and dependencies.