* **Timestamp Based Protocol**

• The time stamp ordering protocol is a scheme in which the order of transaction' is decided in advance based on their timestamps. Thus the schedules are serialized according to their timestamps.

• The timestamp-ordering protocol ensures that any conflicting read and write operations are executed in timestamp order.

• A larger timestamp indicates a more recent transaction or it is also called as younger transaction while lesser timestamp indicates older transaction.

• Assume a collection of data items that are accessed, with read and write operations, by transactions.

• For each data item X the DBMS maintains the following values:

      • **RTS(X):** The Timestamp on which object X was last read (by some transaction T1, i.e., RTS(X):=TS(T;)) [Note that: RTS stands for Read Time Stamp]

      • **WTS(X):** The Timestamp on which object X was last written (by some transaction T,, i.e., WTS(X):=TS(T)) [Note that: WTS stands for Write Time Stamp]

• For the following algorithms we use the following assumptions: - A data item X in the database has a RTS(X) and WTS(X). These are actually the timestamps of read and write operations performed on data item X at latest time.

• A transaction T attempts to perform some action (read or write) on data item X on some timestamp and we call that timestamp as TS(T).

• By timestamp ordering algorithm we need to decide whether transaction T has to be aborted or T can continue execution.

**Basic Timestamp Ordering Algorithm**

**Case 1 (Read):** Transaction T issues a read(X) operation

i) If TS(T) < WTS(X), then read(X) is rejected. T has to abort and be rejected.

ii) If WTS(X) ≤ TS(T), then execute read(X) of T and update RTS(X).

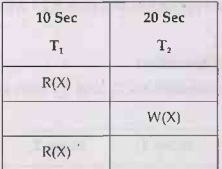
**Case 2 (Write):** Transaction T issues a write(X) operation

i) If TS(T) < RTS(X) or if TS(T) < WTS(X), then write is rejected

ii) If RTS(X)≤ TS(T) or WTS(X) ≤ TS(T), then execute write(X) of T and update WTS(X).

**Example for Case 1 (Read operation)**

(i) Suppose we have two transactions T1 and T2 with timestamps 10 sec and 20 sec respectively.



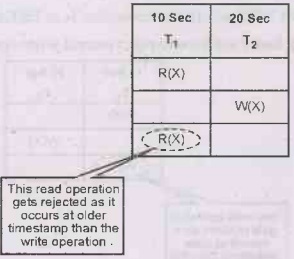
RTS(X) and WTS(X) is initially = 0

Then RTS(X)=10, when transaction T1 executes

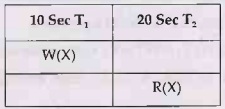
After that WTS(X) =20 when transaction T2 executes

Now if Read operation R(X) occurs on transaction T1 at TS(T1) = 10 then

TS(T1) i.e. 10 <WTS(X) i.e. 20, hence we have to reject second read operation on T1 i.e.



(ii) Suppose we have two transactions T1 and T2 with timestamps 10 sec and 20 sec respectively.



RTS(X) and WTS(X) is initially = 0

Then WTS(X) =10 as transaction T1 executes.

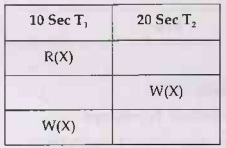
Now if Read operation R(X) occurs on transaction T2 at TS(T2) = 20 then

TS(T2) i.e. 20 >WTS(X) which is 10, hence we accept read operation on T2. The transaction T2 will perform read operation and now RTS will be updated as

RTS(X)=20

**Example for Case 2 (Write Operation)**

(i) Suppose we have two transactions T1 and T2 with timestamps 10 sec and 20 sec respectively.

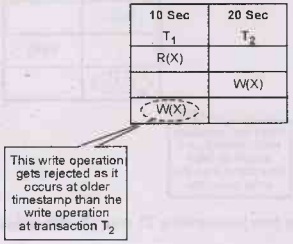


RTS(X) and WTS(X) is initially = 0

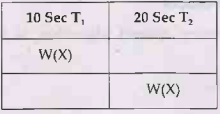
Then RTS(X)=10, when transaction T1 executes

After that WTS(X) =20 when transaction T2 executes

Now if Write operation W(X) occurs on transaction T1 at TS(T1) = 10 then TS(T1) i.e. 10 <WTS(X), hence we have to reject second write operation on T1 i.e.



(ii) Suppose we have two transactions T1 and T2 with timestamps 10 sec and 20 sec respectively.



RTS(X) and WTS(X) is initially = 0

Then WTS(X) =10 as transaction T1 executes.

Now if write operation W(X) occurs on transaction T2 at TS(T2) = 20 then

TS(T2) i.e. 20 >WTS(X) which is 10, hence we accept write operation on T2. The transaction T2 will perform write operation and now WTS will be updated as

**WTS(X) = 20**

**Advantages and disadvantages of time stamp ordering**

**Advantages**

(1)Schedules are serializable

(2) No waiting for transaction and hence there is no deadlock situation.

**Disadvantages**

(1) Schedules are not recoverable once transactions occur.

(2) Same transaction may be continuously aborted or restarted.

* **Strict Timestamp Ordering**

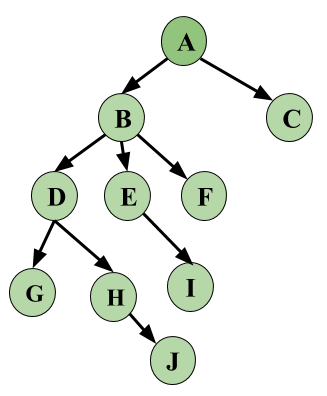
strict timestamp ordering is a variation of basic timestamp ordering. Strict timestamp ordering ensures that the transaction is both strict and conflicts serializable. In Strict timestamp ordering a transaction T that issues a Read\_item(X) or Write\_item(X) such that TS(T) > W\_TS(X) has its read or write operation delayed until the Transaction T‘ that wrote the values of X has committed or aborted.

* **Graph Based Protocol :**
* In graph-based concurrency control, transactions are represented as nodes in a graph, and the conflicts between transactions are represented as edges between the nodes.
* A conflict between two transactions occurs when they access the same data item, and at least one of the transactions performs a write operation.
* When a transaction starts, a node is created in the graph representing the transaction.
* When a transaction accesses a data item, it acquires a shared or exclusive lock on the item, and the corresponding node is added to the graph.
* If a transaction tries to acquire an exclusive lock on an item that is already locked by another transaction, a conflict edge is added between the nodes representing the two transactions.
* When a transaction completes, all the edges incident on its node are removed from the graph.
* Before granting a lock to a transaction, the protocol checks the graph for any cycles. If a cycle exists, it means that there is a conflict between transactions, and one of them needs to be rolled back to break the cycle.
* The advantage of graph-based concurrency control is that it can handle more complex transaction dependencies than other protocols, such as two-phase locking.
* Graph-Based Protocols are used as an alternative to 2-PL. *Tree Based Protocols is a simple implementation of Graph Based Protocol*.

### ****Tree Based Protocol:****

* Partial Order on Database items determines a tree-like structure.
* Only Exclusive Locks are allowed.
* The first lock by Ti may be on any data item. Subsequently, a data Q can be locked by Ti only if the parent of Q is currently locked by Ti.
* Data items can be unlocked at any time.

 following is a *Database Graph* which will be used as a reference for locking the items subsequently.



Let’s look at an example based on the above Database Graph. We have three Transactions in this schedule and this is a skeleton example, i.e, we will only see how Locking and Unlocking work, let’s keep this simple and not make this complex by adding operations on data.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **T1** | **T2** | **T3** |
| **1** | **Lock-X(A)** |  |  |
| **2** | **Lock-X(B)** |  |  |
| **3** |  | **Lock-X(D)** |  |
| **4** |  | **Lock-X(H)** |  |
| **5** |  | **Unlock-X(D)** |  |
| **6** | **Lock-X(E)** |  |  |
| **7** | **Lock-X(D)** |  |  |
| **8** | **Unlock-X(B)** |  |  |
| **9** | **Unlock-X(E)** |  |  |
| **10** |  |  | **Lock-X(B)** |
| **11** |  |  | **Lock-X(E)** |
| **12** |  | **Unlock-X(H)** |  |
| **13** | **Lock-X(B)** |  |  |
| **14** | **Lock-X(G)** |  |  |
| **15** | **Unlock-X(D)** |  |  |
| **16** |  |  | **Unlock-X(E)** |
| **17** |  |  | **Unlock-X(B)** |
| **18** | **Unlock-X(G)** |  |  |

From the above example, first see that the schedule is Conflict Serializable. Serializability for Locks can be written as **T2 –> T1 –> T3**.   
Data items Locked and Unlocked are following the same rule as given above and follow the Database Graph.

Thus, let’s revise once more what are the key points of Graph-Based Protocols.

**Advantage:**

* Ensures Conflict Serializable Schedule.
* Ensures Deadlock Free Schedule
* Unlocking can be done anytime

With some advantages come some disadvantages also.

**Disadvantage:**

* Unnecessary locking overheads may happen sometimes, like if we want both D and E, then at least we have to lock B to follow the protocol.
* ***Cascading Rollbacks*** is still a problem. We don’t follow a rule of when Unlock operation may occur so this problem persists for this protocol. Overall this protocol is mostly known and used for its unique way of implementing Deadlock Freedom.
* **Dead Lock Handling :**

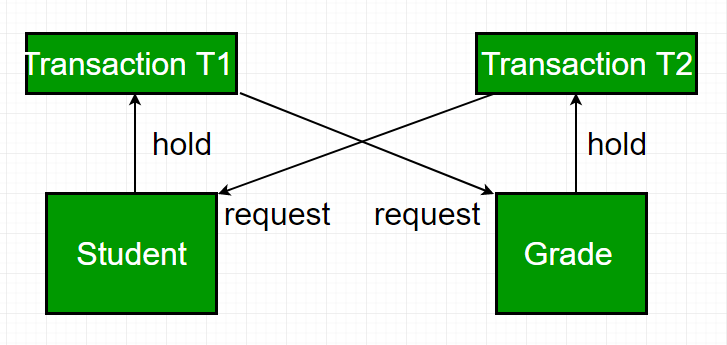
## What is Deadlock?

The Deadlock is a condition in a multi-user database environment where transactions are unable to the complete because they are each waiting for the resources held by other transactions. This results in a cycle of the dependencies where no transaction can proceed.

Basically, **Deadlocks occur when two or more transactions wait indefinitely for resources held by each other.**

**Characteristics of Deadlock**

* Mutual Exclusion: Only one transaction can hold a particular resource at a time.
* Hold and Wait: The Transactions holding resources may request additional resources held by others.
* No Preemption: The Resources cannot be forcibly taken from the transaction holding them.
* Circular Wait: A cycle of transactions exists where each transaction is waiting for the resource held by the next transaction in the cycle.
* **Example –** let us understand the concept of deadlock 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.



## ****What is 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.

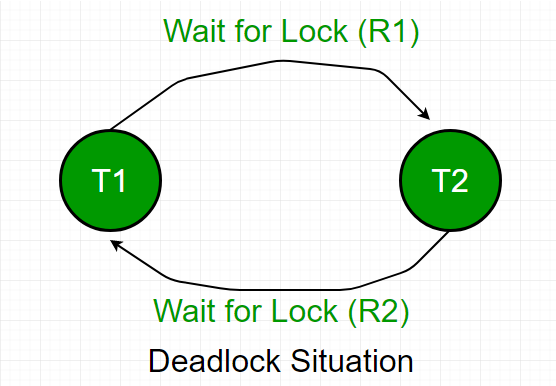
One method of avoiding deadlock is using application-consistent logic. In the above-given example, Transactions that access Students and  Grades should always access the tables in the same order. In this way, in the scenario described above, Transaction T1 simply waits for transaction T2 to release the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

## ****What is 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

Avoiding one or more of the above-stated Coffman conditions can lead to the prevention of a deadlock. Deadlock prevention in larger databases is a much more feasible situation rather than handling it.

The DBMS is made to efficiently analyze every database transaction, whether they can cause a deadlock or not, if any of the transactions can lead to a deadlock, then that transaction is never executed.

1. **Wait-Die Scheme:** When a transaction requests a resource that is already locked by some other transaction, then the DBMS checks the timestamp of both the transactions and makes the older transaction wait until that resource is available for execution.
2. **Wound-wait Scheme:** When an older transaction demands a resource that is already locked by a younger transaction (a transaction that is initiated later), the younger transaction is forced to kill/stop its processing and release the locked resource for the older transaction's own execution. The younger transaction is now restarted with a one-minute delay, but the timestamp remains the same. If a younger transaction requests a resource held by an older one, the younger transaction is made to wait until the older one releases the resource.