Chapter 10: Transaction Processing and Concurrency Control

10.1 Transaction:

- A transaction is an action carried out by a single user or an application program which reads or updates the contents of the database.
- A transaction operations must be done entirely (commit) or aborted (Rollback), no intermediated state are acceptable.
- During transaction execution, the database may be temporarily inconsistent however when the transaction completes successfully i.e. commits, the database must be consistent. After a transaction commits, the changes it has made to the database persist, even if there is system failure.

10. 2 ACID Properties

A transaction is a very small unit of a program and it may contain several groups of tasks. A transaction in a database system must maintain Atomicity, Consistency, Isolation, and Durability − commonly known as ACID properties − in order to ensure accuracy, completeness, and data integrity.

1. ***Atomicity:*** This property states that a transaction must be treated as an atomic unit, that is, either all of its operations are executed or none. There must be no state in a database where a transaction is left partially completed. Atomicity is maintained in the presence of deadlock, software failure, disk failure etc.
2. ***Consistency:*** The consistency property of transaction implies that if database was in consistent state before the initiation of a transaction, then at the end of the transaction the database will also be in a consistent state. This means that a transaction can't break the rules or integrity constraints of the database.
3. ***Isolation:*** The isolation property of a transaction indicates that the steps of operations performed by a transaction should be isolated from other transaction i.e. the execution of a transaction should not be interfered by any other transaction being executed simultaneously. Intermediate results of any transaction must be hidden from other concurrently executing transactions.
4. ***Durability:*** This property ensures that once the transaction has completed execution, the updates and modifications to the database are stored in and written to disk and they

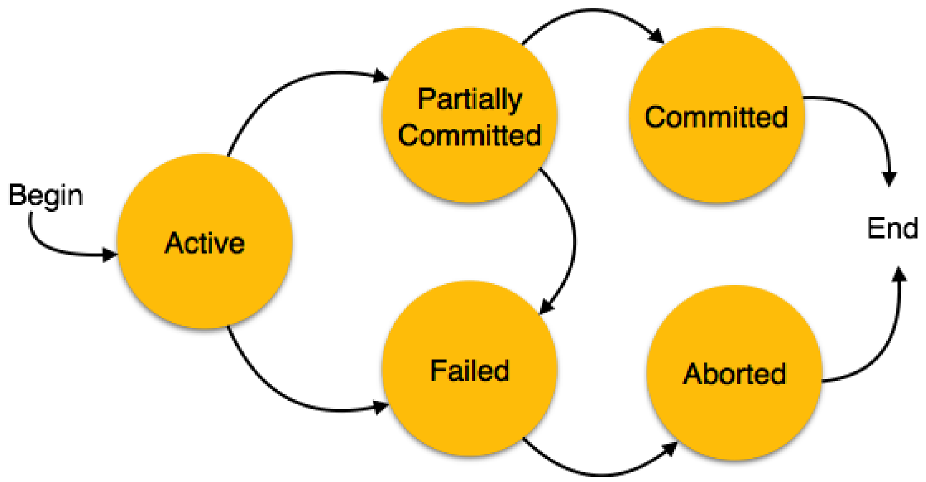
persist even is system failure occurs. These updates now become permanent and are stored in a non-volatile memory. The effects of the transaction, thus, are never lost.

The ACID properties, in totality, provide a mechanism to ensure correctness and consistency of a database in a way such that each transaction is a group of operations that acts a single unit, produces consistent results, acts in isolation from other operations and updates that it makes are durably stored.

Example: Transaction to transfer Rs 1000 from account A to account B

1. Read (A)
2. A: = A – 1000
3. Write (A)
4. Read (B)
5. B: = B + 1000
6. Write (B)

10.3 Transaction States/ States of a Transaction



A transaction in a database can be in one of the following states:

1. Active − In this state, the transaction is being executed. This is the initial state of every transaction.
2. Partially Committed − When a transaction executes its final operation, it is said to be in a partially committed state.
3. Failed − A transaction is said to be in a failed state if any of the checks made by the database recovery system fails. A failed transaction can no longer proceed further.

4. Aborted − If any of the checks fails and the transaction has reached a failed state, then the recovery manager rolls back all its write operations on the database to bring the database back to its original state where it was prior to the execution of the transaction. Transactions in this state are called aborted. The database recovery module can select one of the two operations after a transaction aborts :

   Re-start the transaction

   Kill the transaction

5. Committed − If a transaction executes all its operations successfully, it is said to be committed. All its effects are now permanently established on the database system.

10.4 Concurrent Executions/Schedules:

- Multiple transactions are allowed to run concurrently in the system. This helps to
  1. Increased processor and disk utilization
  2. Reduced average response time
- Concurrency control schemes: It is a mechanism to achieve isolation that is to control interaction among the concurrent transactions in order to prevent them from destroying the consistency of database.
- **Schedules:** A schedule is series of operations from one or more transactions. There are four types of schedule:
  1. Serial schedule
  2. Non-serial schedule
  3. Equivalent schedule
  4. Conflict schedule

  → A schedule S is called ***serial schedule*** if for any two transaction Ti and Tj participating in S either all operation of Ti precedes all operation of Tj or vice versa. Hence in serial schedule, only one transaction is active at a time. The commit or abort of a active transaction initiates execution of next transaction. No interleaving occurs in serial schedule

  → A schedule S is called ***non-serial schedule*** if for any two transaction Ti and Tj participating in S, the operation of each transaction are executed non-consecutively with interleaved operations from the other transaction.

→ Two schedules A and B are said to be ***equivalent schedules*** if the execution of first schedule is identical to execution of second schedule.

→ A schedule S is called ***conflict schedule*** if for any two transaction Ti and Tj there is some action ti of Ti and an action tj of Tj accessing the same object and at least one of the actions is write.

Schedule1

T1	T2
ead (A)	
A := A – 50	
Write (A)	
Read (B)	
B := B + 50	
Write (B)	
	Read (A)
	Temp := A *0.1
	A := A – Temp
	Write (A)
	Read (B)
	B := B + Temp
	Write (B)

Schedule2

T1	T2
	Read (A)
	Temp := A *0.1
	A := A – Temp
	Write (A)
	Read (B)
	B := B + Temp
	Write (B)
Read (A)	
A := A – 50	
Write (A)	
Read (B)	
B := B + 50	
Write (B)	

In the above example, schedule1 and schedule2 are serial schedule because in schedule1 Transaction T1 is followed by Transaction T2 and in schedule2 Transaction T2 is followed by Transaction T1.

Schedule3

T1	T2
Read (A)	
A := A – 50	
Write (A)	
	Read (A)
	Temp := A *0.1
	A := A – Temp
	Write (A)
Read (B)	
B := B + 50	
Write (B)	
	Read (B)
	B := B + Temp
	Write (B)

Schedule4

T1	T2
Read (A)	
A := A – 50	
	Read (A)
	Temp := A *0.1
	A := A – Temp
	Write (A)
	Read (B)
Write (A)	
Read (B)	
B := B + 50	
Write (B)	
	B := B + Temp
	Write (B)

Schedule3 is non-serial schedule because there is interleaving of operations between transactions T1 and T2. However the schedule3 is equivalent schedule to schedule1 and schedule2 since the sum (A+B) is preserved. Schedule4 is conflict schedule because it doesn’t preserve the value of (A+B)

10.5 Serializability

Serial schedule are generally considered unacceptable in practice because in serial schedule, if a transaction waits for I/O operations to complete, it make waste of CPU time making while other transactions are ready for execution. So interleaving i.e. non serial schedule could improve the use of CPU cycle. But some non-serial schedule produces erroneous result while some produce correct result. So this introduces the concept of serializability.

- → A schedule is serializable if it is equivalent to serial schedule.
- → A concurrent execution of N transaction is called serializable, if the execution is computationally equivalent to a serial execution. When more than one transaction is being executed concurrently, we must have serializability in order to have same effect on the database as same serial execution does.

Different forms of schedule equivalence give rise to the notions of: There are two types of serializability:

1. Conflict serializability
2. View serializability

1. Conflict serializability

If two operations in a schedule satisfy all these three conditions then operations are said to be conflict.
i. They belong to different transaction.
ii. They access the same item.
iii. At least one of the operations is write operation.

Instructions li and lj of transactions Ti and Tj respectively, conflict if and only if there exists some item Q accessed by both li and lj, and at least one of these instructions wrote Q.

1. li = read(Q), lj = read(Q). li and lj don't conflict.
2. li = read(Q), lj = write(Q). They conflict.
3. li = write(Q), lj = read(Q). They conflict.
4. li = write(Q), lj = write(Q). They conflict.

- ✓ Conflicting operations pair (R1 (A), W2 (A)) because they belong to two different transactions on same data item A and one of them is write operation.
- ✓ Similarly, (W1 (A), W2 (A)) and (W1 (A), R2 (A)) pairs are also conflicting.
- ✓ On the other hand, (R1 (A), W2 (B)) pair is **non-conflicting** because they operate on different data item.
- ✓ Similarly, ((W1 (A), W2 (B)) pair is non-conflicting.

- If a schedule is a conflict equivalent to its serial schedule then it is called Conflict Serializable Schedule.
- Two schedules are said to be conflict equivalent if all the conflicting operations in both the schedule get executed in the same order.

If a schedule S can be transformed into a schedule S´ by a series of swaps of non-conflicting instructions, we say that S and S´ are conflict equivalent.
So a schedule S is conflict serializability if it is conflict equivalent to a serial schedule.

- It is easier to achieve & is based on the precedence graph.

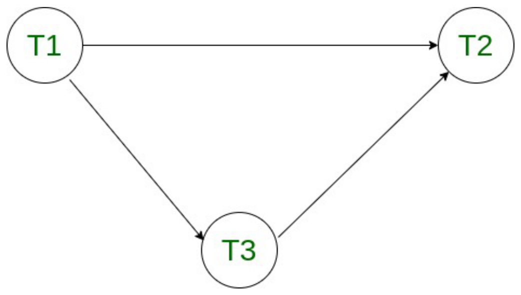
Example with precedence graph

T1	T2	T3
	R2(A)	
		R3(A)
W1(B)		
	W2(A)	
		R3(B)
	W2(B)	

Two operations are said to be conflicting if the belong to different transaction, operate on same data and at least one of them is a write operation.

1. R3(A) and W2(A) [T3 -> T2]
2. W1(B) and R3(B) [T1 -> T3]
3. W1(B) and W2(B) [T1 -> T2]
4. R3(B) and W2(B) [T3 -> T2]

Constructing the precedence graph, we see there are no cycles in the graph. Therefore, the schedule is Conflict Serializable.



The serializable schedule is, (On the basis of less **in degree** values of the vertices)

T1 -> T3 -> T2

2. View serializability

- There may be some schedules that are not Conflict-Serializable but still gives a consistent result because the concept of Conflict-Serializability becomes limited when the Precedence Graph of a schedule contains a loop/cycle.
- If a schedule is view equivalent to its serial schedule then it is called **View Serializable Schedule**

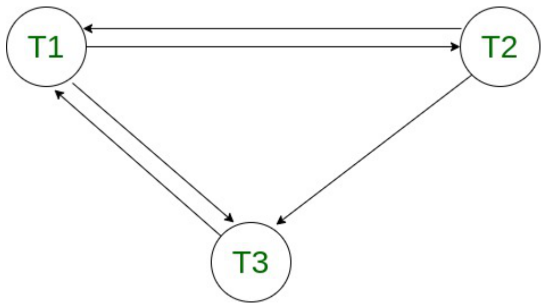
- If the order of initial read, final write and update operations is the same in both the schedules, then this two schedules are view equivalent.

  Example:

T1	**T2**	**T3**
R1(A)		
	W2(A)	
		R3(A)
W1(A)		
		W3(A)

The conflicting operations for this schedule are

1. R1(A) and W2(A) [T1 -> T2]
2. R1(A) and W2(A) [T1 -> T3]
3. W2(A) and R3(A) [T2 -> T3]
4. W2(A) and W1(A) [T2 -> T1]
5. W2(A) and W3(A) [T2 -> T3]
6. R3(A) and W1(A) [T3 -> T1]
7. W1(A) and W3(A) [T1 -> T3]



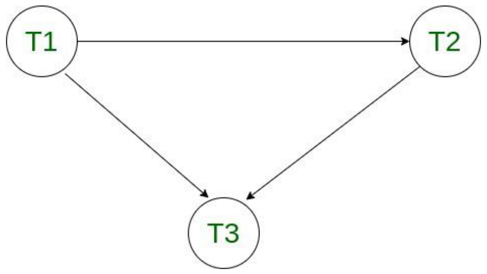
As we can see that there is a cycle in the precedence graph, it means that the given schedule is not Conflict Serializable.

In order to check for View Serializability, we will draw a **Dependency Graph** of the schedule.

From the given schedule we gather the following points:

- T1 reads A before T2 updates A thus, T1 must execute before T2.
- T3 does the final update on A thus, it must execute in the end.

Constructing the dependency graph.



As there exists no cycle in the graph, we can say that the given schedule is View Serializable. The serializable schedule is T1 -> T2 -> T3.

10.6 Concurrently Control Technique

Concurrency control is the process of managing simultaneous operations like inserts, updates, deletes on the database without having them interfere with one another. Simply concurrency control is the management of concurrent transaction execution. DBMS implements concurrency control technique to ensure serializability and isolation of transaction in order to guard the consistency and integrity of database. There are several concurrency control techniques which are explained below.

1. Lock based protocols
2. Two Phase Locking Protocol (2PL)
3. Time Stamp Based Ordering Scheme

Explanation:

1. *Lock Based Protocols*

One way to ensure serializability is to require that data items be accessed in mutually exclusive manner i.e. while one transaction is accessing a data item s, no other transaction can modify that data items. In this scheme, each data item has a lock associated with it. When a transaction T wants to access the data item, it first examines the associated lock.
If no other transaction hold the lock, the lock scheduler locks the data items for transaction T. now if any other transaction Ti wants to access the same data item then Ti has to wait until T releases the lock. So any transaction must obtain a read lock or write lock on data item before it can perform a read or write operation.

Mode of locking

i. Shared Lock(S) or Read Lock

   If Ti holds shared mode lock on Q, it can only read and cannot write. Multiple shared mode lock can exist for the same data item Q.

ii. Exclusive Lock(X) or Write Lock

If T_i holds exclusive mode lock on Q, then it can both read as well as write. There can exist only one exclusive mode lock on a data item Q

Based on these locking modes, we can define a compatibility matrix.

	S	X
S	True	False
X	False	False

For example: let us consider we have two transactions as below.

T1	T2
T1: Read (Y)	T2: Read (X)
Read (X)	Read (Y)
X = X + Y	X=X – Y
Write(X)	Write(X)

Using locking, this transaction can be written as,

T1	T2
T1: Readlock (Y)	T2: Writelock (X)
Read (Y)	Read (X)
Unlock (Y)	Readlock(Y)
Writelock (X)	Read (Y)
Read (X)	Unlock (Y)
X = X + Y	X = X - Y
Write (X)	Write (X)
Unlock (X)	Unlock (X)

2. *Two phase locking protocol*

   This is a protocol which ensures conflict-serializable schedules. This protocol requires that each transaction issue lock and unlock requests in different phases. There are two phases as discussed below:

   Phase 1: Growing Phase

   - transaction may obtain locks

- transaction may not release locks

Phase 2: Shrinking Phase

- transaction may release locks
- transaction may not obtain locks

Example: Two phase locking guarantee serializability, but it doesn't prevent deadlock.

	T1	**T2**
1	Lock T(A)	
2		Lock T(D)
3	Lock T(B)	
4	…….	……
5	Unlock(A)	
6		Lock T(C)
7	Unlock(B)	
8		Unlock(D)
9		Unlock(C)

Example: Transaction T1 and T2 are the example of two phase locking protocol. Initially a transaction is in growing phase and can acquire locks as per its need. Then the transaction reaches its lock point i.e. it is the point where a transaction acquires its final lock. Finally when transaction unlocks any data item, it enters into shrinking phase and no more locks can be acquired again.

Transaction T1:

- The growing Phase is from steps 1-3.
- The shrinking Phase is from steps 5-7.
- Lock Point at 3

Transaction T2:

- The growing Phase is from steps 2-6.
- The shrinking Phase is from steps 8-9.
- Lock Point at 6

The protocol assures serializability. It can be proved that the transactions can be serialized in the order of their lock points (i.e. the point where a transaction acquired its final lock). Two-phase locking does not ensure freedom from deadlocks.

Cascading roll-back is possible under two-phase locking. To avoid this, follow a modified protocol called **strict two-phase locking**. Here a transaction must hold all its exclusive locks till it commits/aborts.

Rigorous two-phase locking is even stricter: here all locks are held till commit/abort. In this protocol transactions can be serialized in the order in which they commit.

Note:

Deadlock: Deadlock is a set of blocked transactions each holding a resource and waiting to acquire a resource held by another process in the set. The simplest example of deadlock is given below:

a. Transaction T1 request resource A and receive it.
b. Transaction T2 request resource B and receive it.
c. Transaction T1 request resource B and is queued up waiting the release of B by transaction T2.
d. Transaction T2 request resource A and is queued up waiting the release of A by transaction T1

Condition of deadlock

i. **Mutual Exclusion**
    - ✓ There must exist at least one resource in the system which can be used by only one process at a time.
    - ✓ If there exists no such resource, then deadlock will never occur.
    - ✓ Printer is an example of a resource that can be used by only one process at a time.

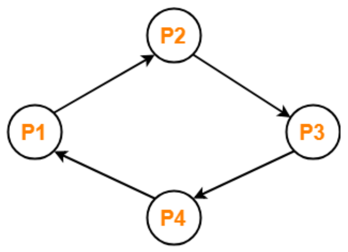
ii. **Hold & wait**
    - ✓ There must exist a process which holds some resource and waits for another resource held by some other process.

iii. **No preemption**

- ✓ Once the resource has been allocated to the process, it can not be preempted.
- ✓ It means resource cannot be snatched forcefully from one process and given to the other process.
- ✓ The process must release the resource voluntarily by itself.

iv. Circular Wait

- ✓ All the processes must wait for the resource in a cyclic manner where the last process waits for the resource held by the first process.



Circular Wait

3. <u>Time Stamp Based Ordering Scheme</u>

In this scheme, with each transaction Ti in the system we associated a unique fixed timestamp denoted by Ts (Ti). This time stamp is assigned by the DBMS before the transaction Ti starts execution. If Ti has been assigned timestamp Ts (Ti) and a new transaction Tj enters the system then Ts (Ti) < Ts (Tj). There are two simple methods for implementing this scheme.

1. Use the value of system clock as the timestamp i.e. a transaction's timestamp value is equal to the value of clock when the transaction enters the system.
2. Use the logical counter that is incremented after a new timestamp has been assigned i.e. transactions timestamp is equal to the values o the counters when transaction enters the system.

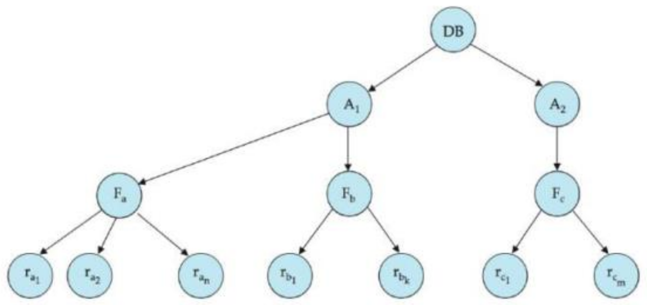
The protocol manages concurrent execution such that the time-stamps determine the serializability order. In order to assure such behavior, the protocol maintains for each data Q two timestamp values:

- W-timestamp(Q) is the largest time-stamp of any transaction that executed write(Q) successfully.
- R-timestamp(Q) is the largest time-stamp of any transaction that executed read(Q) successfully.

Multiple Granularity

Granularity is the size of data item being locked. It is considered as the major factor concurrency control because it can effect the performance of concurrency and recovery. Because, if the granularity of a data item is very large then the overhead of the lock is very low. For example, if we select the data item as whole database, then a transaction T while accessing even a small portion of database will lock the whole database and no concurrent access by other transactions would be allowed.

Multiple granularity allows data item to be of various size and define a hierarchy of data granularities where the small granularities are nested within the larger one. It can be represented graphically as a tree.



The highest level represents the entire database. Below it are nodes of type area. Each area in turn has node of type file as its children. Finally each file has node of type records. In addition to shared lock and exclusive lock, there are three more additional lock modes with it.

Questions:

1. Define transaction. Explain about different ACID properties of a database.
2. What do you mean by ACID? Explain different states in a transaction.
3. Define schedule. What are the different types of schedules?
4. What is concurrency control? Explain different concurrency control techniques.
5. What is deadlock? Explain about 2PL and time stamp ordering protocol.
6. Write short notes:
    a) 2PL locking protocol
    b) Deadlock
    c) Concurrency Control
    d) Multiple Granularity
    e) Serializability