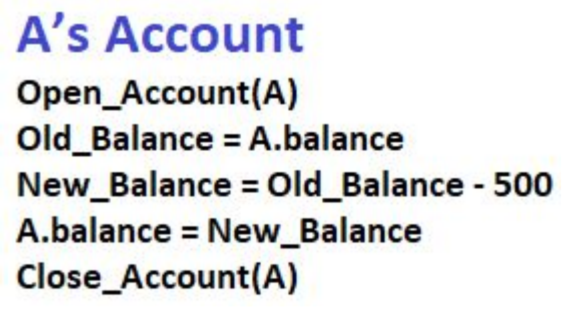
Unit 4 Chapter 2 Transaction Management

Syllabus Topics

- Basic concept of a Transaction
- ACID Properties
- State diagram
- Concept of Schedule
- Serializability – Conflict and View
- Concurrency Control Protocols
- Recovery techniques

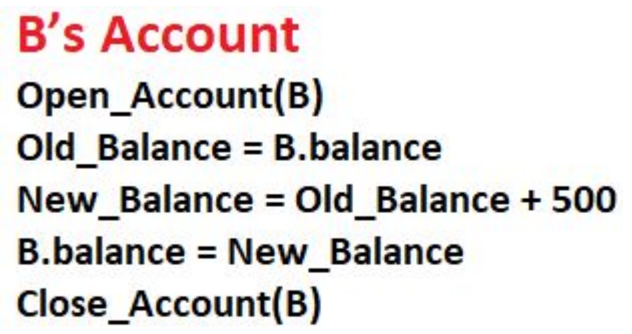
What is Transaction in DBMS?

- Transactions are a set of operations that are used to perform some logical set of work.
- A transaction is made to change data in a database which can be done by inserting new data, updating the existing data, or by deleting the data that is no longer required.
- For example, you are transferring money from your bank account (**Say Account A**) to your friend's account(**Say Account B**), the set of operations would be like this:
- **Simple Transaction Example**
- **Read your account balance**
- **Deduct the amount from your balance**
- **Write the remaining balance to your account**
- **Read your friend's account balance**
- **Add the amount to his account balance**
- **Write the new updated balance to his account**



A's Account

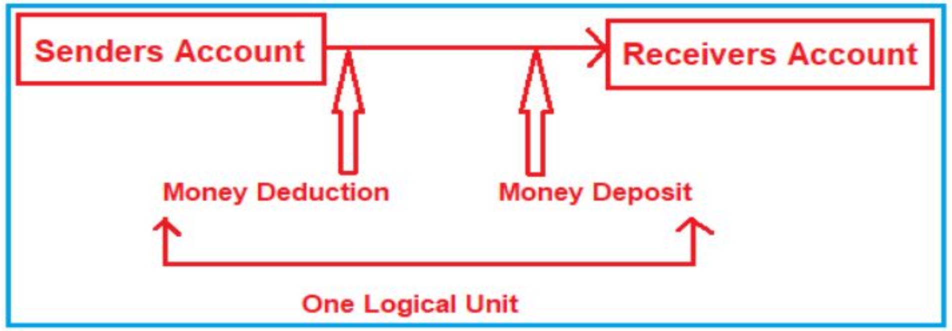
```
Open_Account(A)
Old_Balance = A.balance
New_Balance = Old_Balance - 500
A.balance = New_Balance
Close_Account(A)
```



B's Account

```
Open_Account(B)
Old_Balance = B.balance
New_Balance = Old_Balance + 500
B.balance = New_Balance
Close_Account(B)
```

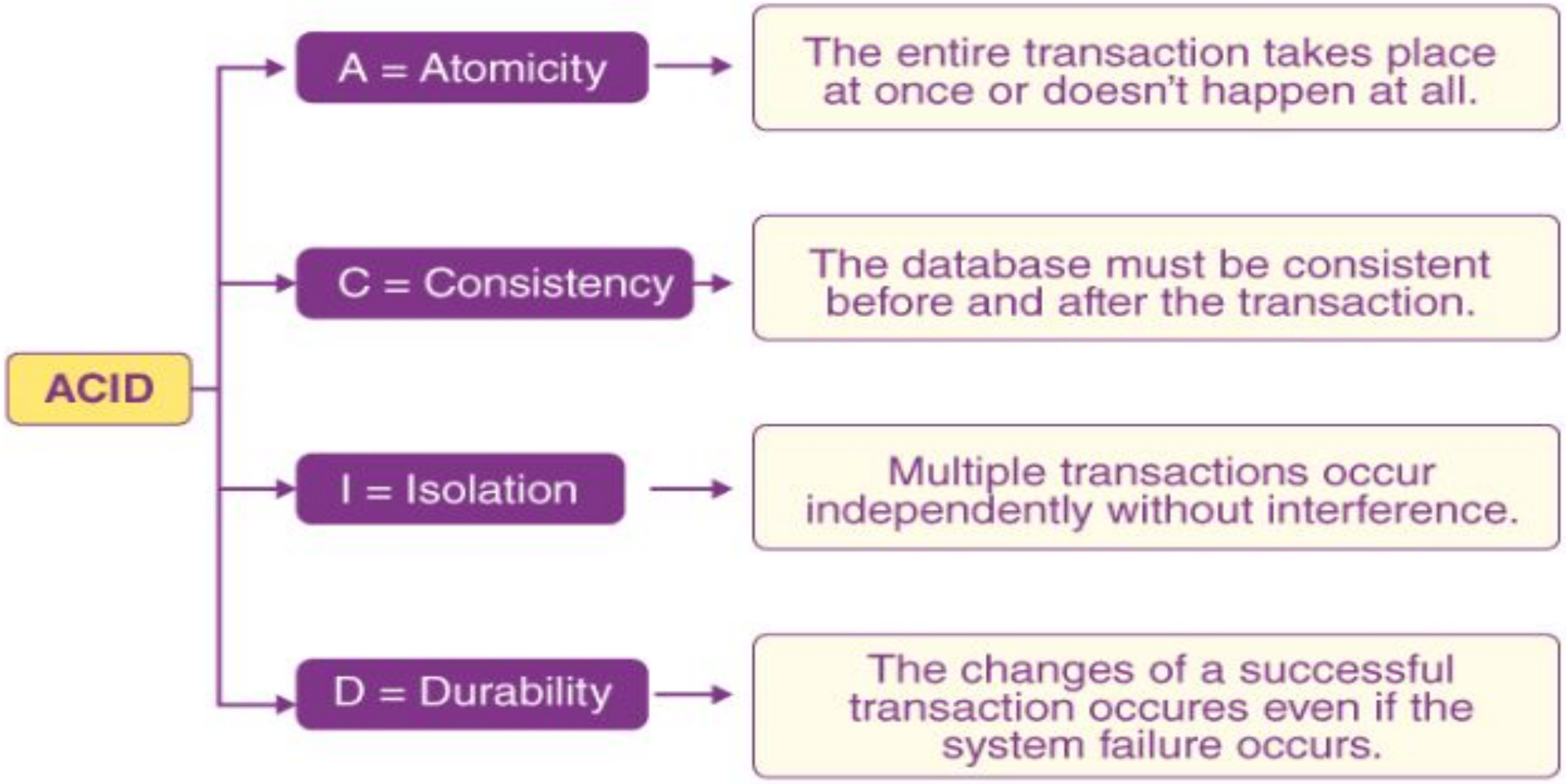
What is Transaction in DBMS?



- In this case, we need to perform at least two updates.
- The first update is happening in the sender's account from where the money is sending and the other update is happening in the receiver's account who is receiving the money.
- Both of these updates should either get committed or get rollback if there is an error. We don't want the transaction to be in a half-committed state.
- **Operations of Transaction:**
- Following are the main operations of transaction:
- **Read(X):** Read operation is used to read the value of X from the database and stores it in a buffer in main memory.
- **Write(X):** Write operation is used to write the value back to the database from the buffer.

ACID properties

- To ensure the **integrity and consistency of data** during a transaction, the database system maintains **four properties**. These properties are widely known as **ACID properties.**


ACID
A = Atomicity
The entire transaction takes place at once or doesn't happen at all.
C = Consistency
The database must be consistent before and after the transaction.
I = Isolation
Multiple transactions occur independently without interference.
D = Durability
The changes of a successful transaction occures even if the system failure occurs.

ACID properties

- **Atomicity**
- This property **ensures that either all the operations of a transaction reflect in database or none.** Either it executes completely or it doesn’t, there shouldn’t be a partial execution.
- **Suppose Account A has a balance of 400$ & B has 700$. Account A is transferring 100$ to Account B.**
- This is a transaction that has two operations
- **a) Debiting 100$ from A’s balance**
- **b) Crediting 100$ to B’s balance.**
- **Let’s say first operation passed successfully while second failed**, in this case A’s balance would be 300$ while **B would be having 700$ instead of 800$.**
- This is unacceptable in a banking system. Either the transaction should fail without executing any of the operation or it should process both the operations. The Atomicity property ensures that.

ACID properties

- **Atomicity**
- There are two key operations are involved in a transaction to maintain the atomicity of the transaction.
- **Rollback : If there is a failure in the transaction, abort the execution and rollback the changes made by the transaction.**
- **Commit:** If transaction executes successfully, **commit the changes to the database.**
- **Consistency**
- Database **must be in consistent state before and after the execution of the transaction.** This ensures that there are no errors in the database at any point of time. Application programmer is responsible for maintaining the consistency of the database.

ACID properties

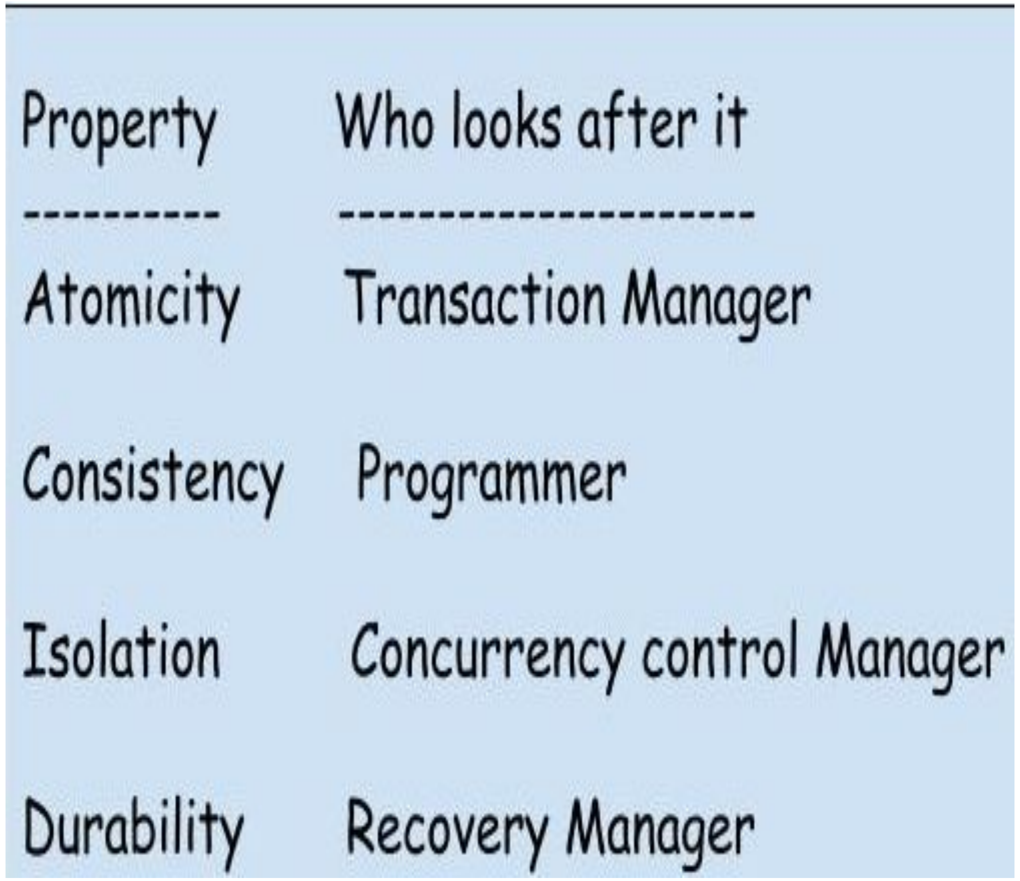
- **Consistency**
- **Example:**
  A transferring 1000 dollars to B. A's initial balance is 2000 and B's initial balance is 5000.
- **Before the transaction:**
  Total of A+B = 2000 + 5000 = 7000$
- **After the transaction:**
  Total of A+B = 1000 + 6000 = 7000$
- There is **no other transaction which change values of A or B during execution of T,** then the **new values of A and B will be 1000 and 6000** i.e., **(A+B=7000 after the transaction)**.
- The **data is consistent before and after the execution of the transaction** so this example maintains the consistency property of the database.

ACID properties

- **Isolation**
- A transaction **shouldn’t interfere with the execution of another transaction**.
- To preserve the consistency of database, the execution of transaction should take place in isolation (**that means no other transaction** should run concurrently when there is a transaction already running).
- For example account A is having a balance of 400$ and it is transferring 100$ to account B & C both.
- So we have two transactions here. Let’s say these transactions run concurrently and both the transactions read 400$ balance, in that case the final balance of A would be 300$ instead of 200$. **This is wrong.**
- **If the transaction were to run in isolation then the second transaction would have read the correct balance 300$ (before debiting 100$)** once the first transaction went successful.

ACID properties

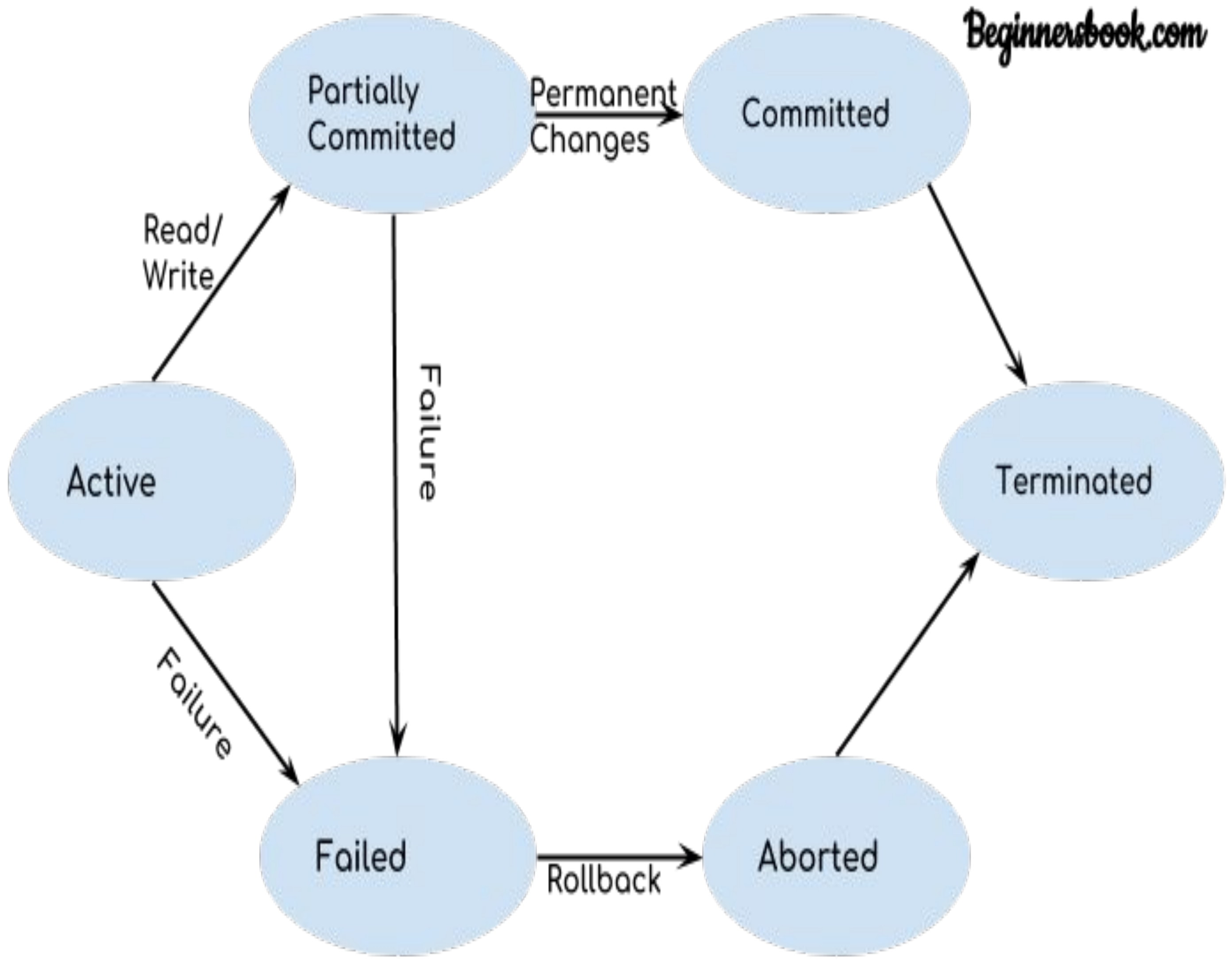
- **Durability**
- Once a transaction completes successfully, the **changes it has made into the database should be permanent even if there is a system failure**.
- The **recovery-management component of database systems ensures the durability of transaction.**
- **ACID properties** are the **backbone of a database management system.**
- These properties ensure that even though there are multiple transaction reading and writing the data in the database**, the data is always correct and consistent.**



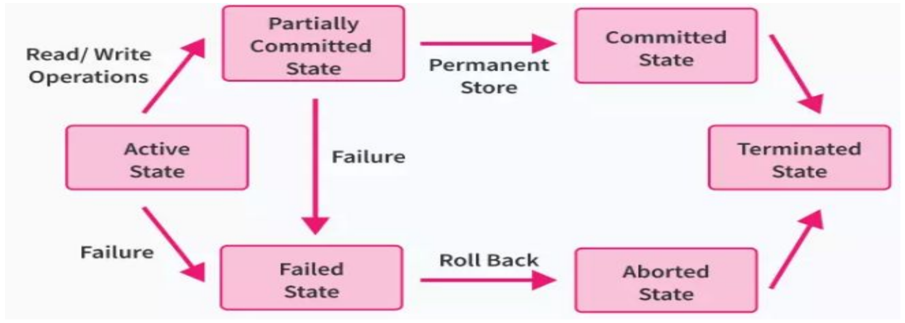
Property	Who looks after it
Atomicity	Transaction Manager
Consistency	Programmer
Isolation	Concurrency control Manager
Durability	Recovery Manager

State diagram

- During the lifetime of a transaction, there are a lot of states to go through. **These states update the operating system about the current state of the transaction.**
- These **states decide the regulations which decide the fate of a transaction whether it will commit or abort.**

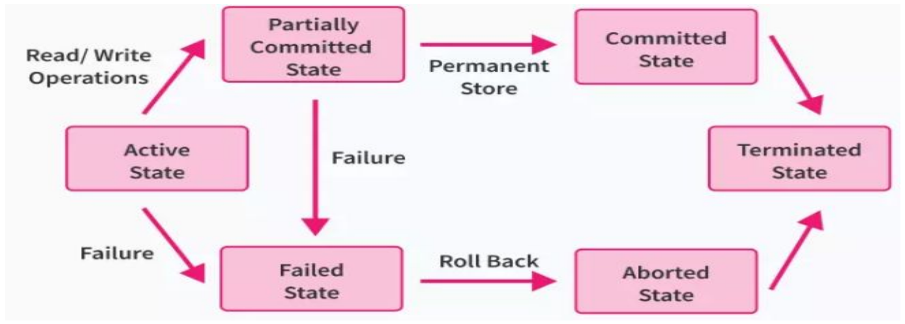


State diagram



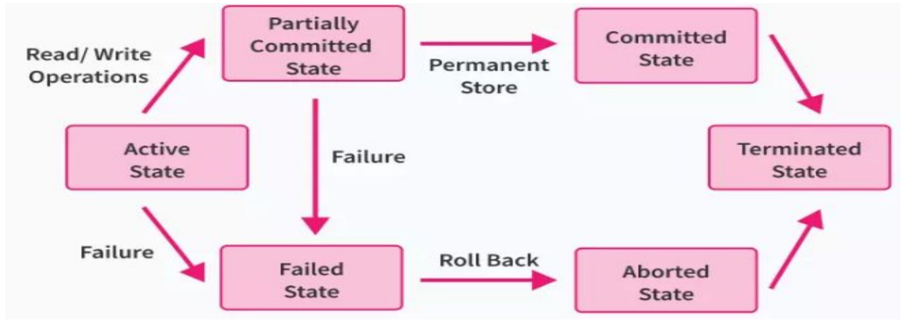
- **Active State**
- If a **transaction is in execution then it is said to be in active state**.
- During this state read or write operations can be performed.
- **Partially Committed State**
- Once the whole transaction is successfully executed, the transaction goes into partially committed state **where we have all the read and write operations performed on the main memory (local memory) instead of the actual database.**
- The reason why we have this state is **because a transaction can fail during execution so if we are making the changes in the actual database instead of local memory, database may be left in an inconsistent state in case of any failure.**
- **This state helps us to rollback the changes made to the database in case of a failure during execution.**

State diagram



- **Failed State**
- A transaction considers failed when **any one of the checks fails or if the transaction is aborted while it is in the active state.**
- If a transaction is executing and a failure occurs, **either a hardware failure or a software failure then the transaction goes into failed state** from the active state.
- **Committed State**
- If a transaction completes the execution successfully **then all the changes made in the local memory during partially committed state are permanently stored in the database.**

State diagram



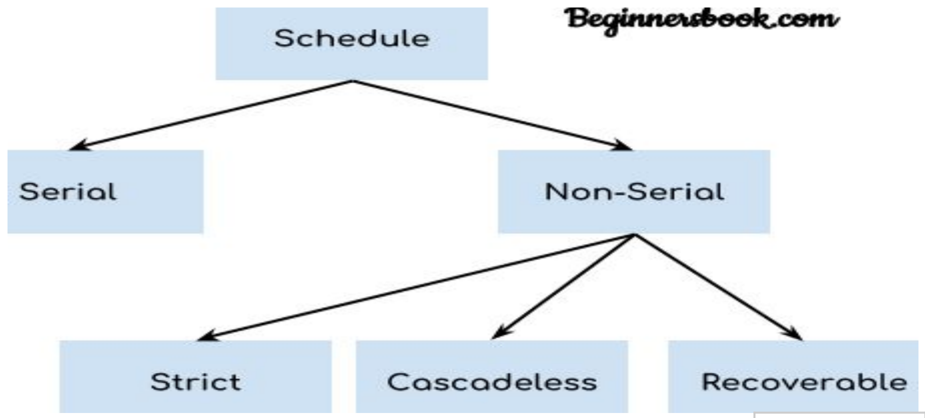
- **Aborted State**
- If the transaction fails during its execution, **it goes from *failed state* to *aborted state*** and because in the previous states all the changes were only made in the main memory, **these uncommitted changes are either deleted or rolled back.**
- The transaction **at this point can restart and start afresh from the active state.**
- **Terminated State**
- State of transaction reaches terminated state when certain transactions which are leaving the system can’t be restarted.

DBMS Schedule

- A **series of operation from one transaction to another transaction** is known as schedule.
- It is **used to preserve the order of the operation** in each of the individual transaction.
- When multiple transactions are running concurrently then there needs to be a sequence in which the operations are performed **because at a time only one operation can be performed on the database.**

Time	Transaction T1	Transaction T2
t1	Read(A)	
t2	A=A+50	
t3	Write(A)	
t4		Read(A)
t5		A+A+100
t6		Write(A)
t7	Read(B)	
t8	B=B+100	
t9	Write(B)	
t10		Read(B)
t11		B=B+50
t12		Write(B)

DBMS Schedule

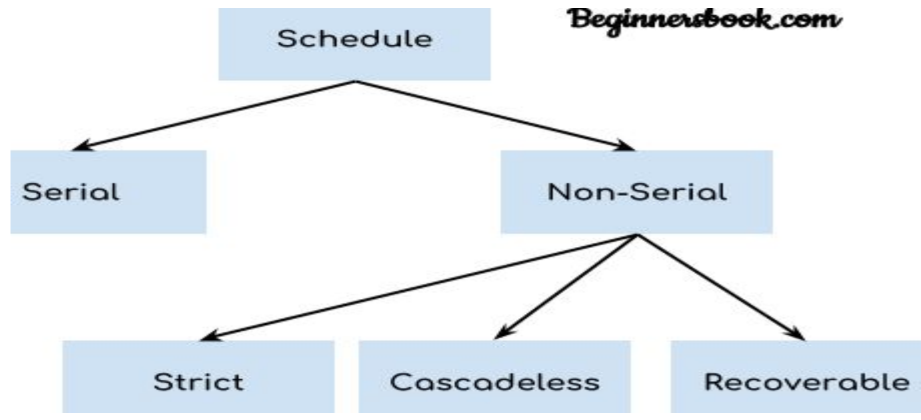


- **Types of Schedules in DBMS**
- **Serial Schedule**
- In **Serial schedule**, **a transaction is executed completely before starting the execution of another transaction.** In other words, you can say that in serial schedule, a transaction does not start execution until the currently running transaction finished execution.
- **For example:**
- Suppose a schedule S with two transactions T1 and T2.
- **If all the instructions of T1 are executed before T2** or **all the instructions of T2 are executed before T1**, then S is said to be a serial schedule.

Time	Transaction T1	Transaction T2
t1	Read(A)	
t2	A=A+50	
t3	Write(A)	
t4	Read(B)	
t5	B=B+100	
t6	Write(B)	
t7		Read(A)
t8		A+A+100
t9		Write(A)
t10		Read(B)
t11		B=B+50
t12		Write(B)

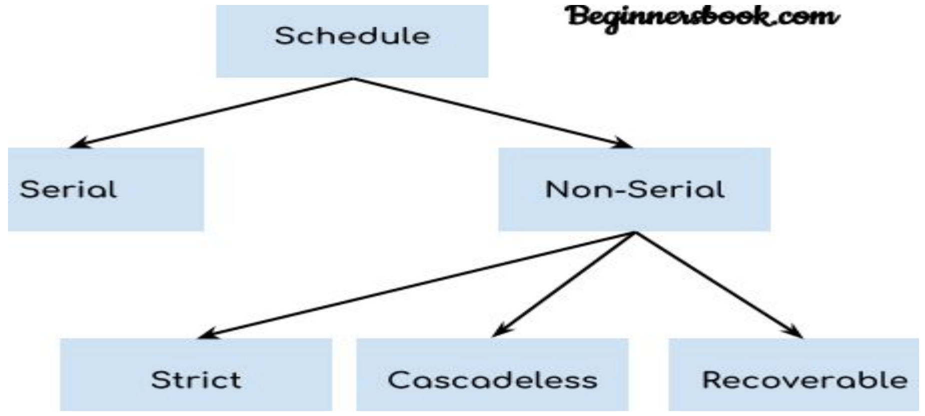
DBMS Schedule

- **Non-Serial Schedule**
- It **contains many possible orders in which the system can execute the individual operations** of the transactions.
- The non-serial schedule is a type of schedule where the **operations of multiple transactions are interleaved.**
- Unlike the serial schedule, this schedule **proceeds without waiting for the execution of the previous transaction to complete.**
- This schedule **may give rise to the problem of concurrency.** In a non-serial schedule multiple transactions are executed concurrently.

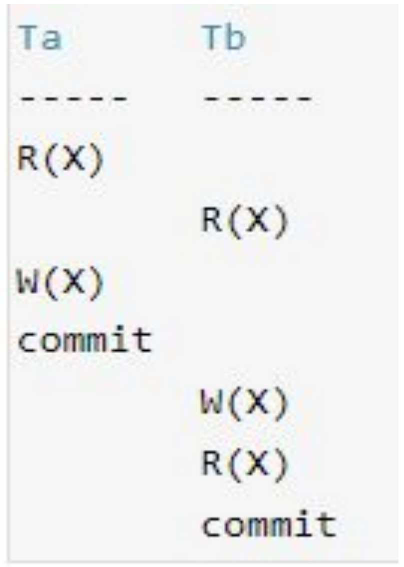


Time	Transaction T1	Transaction T2
t1	Read(A)	
t2		Read(A)
t3	A=A+50	
t4		A+A+100
t5	Write(A)	
t6		Write(A)

DBMS Schedule



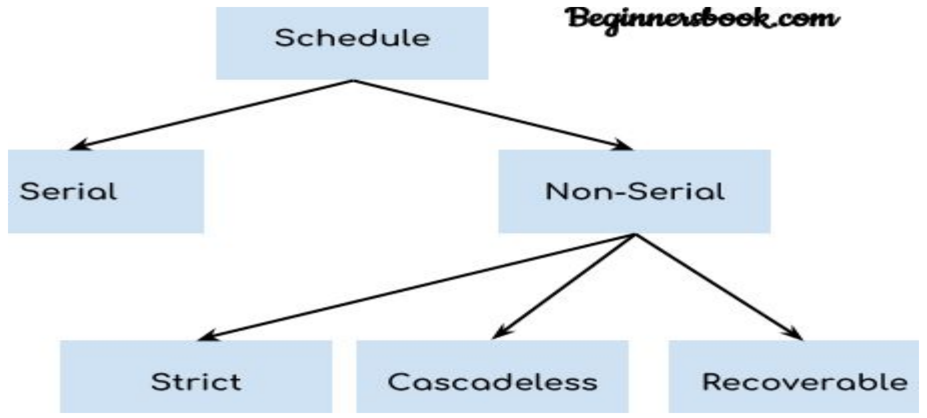
- **Non-Serial Schedule**
- **Strict Schedule**
- In Strict schedule, **if the write operation of a transaction precedes a conflicting operation (Read or Write operation) of another transaction then the commit or abort operation of such transaction should also precede the conflicting operation of other transaction.**
- **Example**
- Lets say we have two transactions Ta and Tb.
- The write operation of transaction Ta precedes the read or write operation of transaction Tb, so the commit or abort operation of transaction Ta should also precede the read or write of Tb.

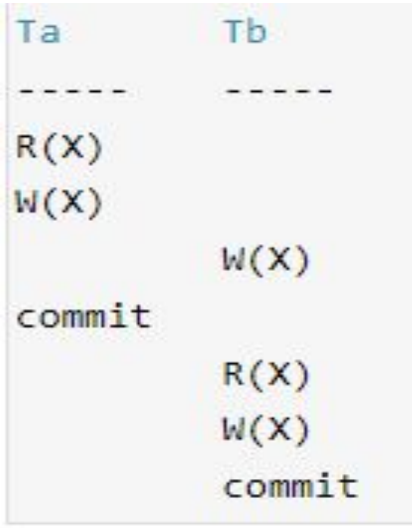


```
Ta          Tb
-----       -----
R(X)
            R(X)
W(X)
commit
            W(X)
            R(X)
            commit
```

- **Here the write operation W(X) of Ta precedes the conflicting operation (Read or Write operation) of Tb so the conflicting operation of Tb had to wait the commit operation of Ta.**

DBMS Schedule

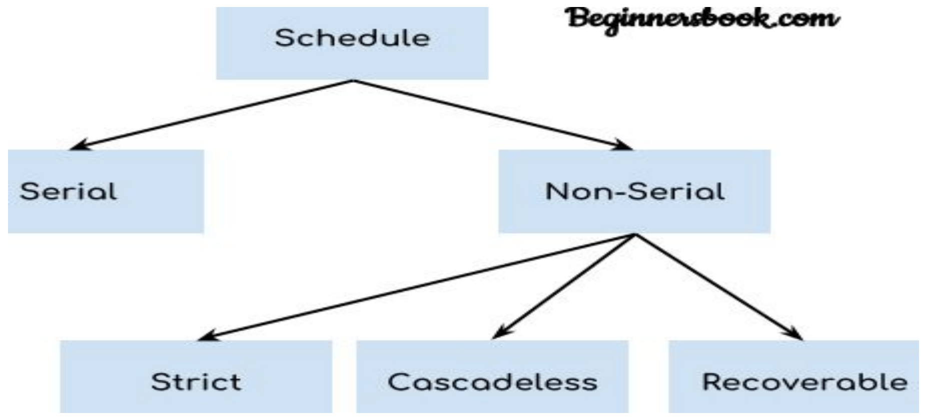




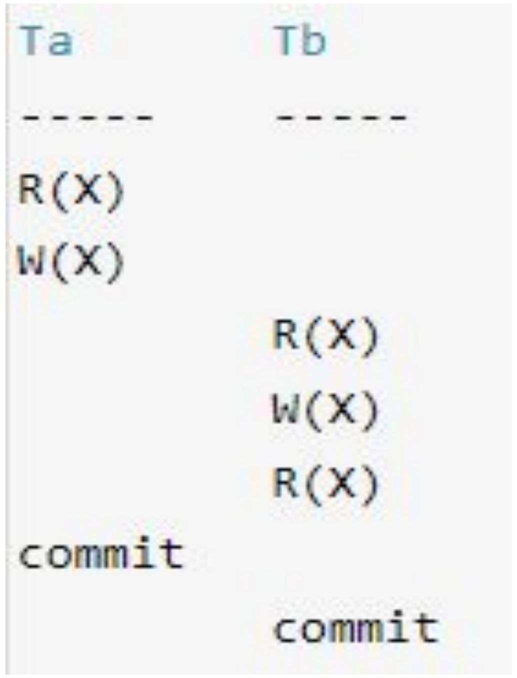
```
Ta          Tb
-----       -----
R(X)
W(X)
            W(X)
commit
            R(X)
            W(X)
            commit
```

- **Non-Serial Schedule**
- **Cascadeless Schedule**
- In Cascadeless Schedule, **if a transaction is going to perform read operation on a value, it has to wait until the transaction who is performing write on that value commits.**
- For example, lets say we have two transactions Ta and Tb. Tb is going to read the value X after the W(X) of Ta then Tb has to wait for the commit operation of transaction Ta before it reads the X.

DBMS Schedule



- **Non-Serial Schedule**
- **Recoverable Schedule**
- In Recoverable schedule, **if a transaction is reading a value which has been updated by some other transaction then this transaction can commit only after the commit of other transaction which is updating value.**
- Here Tb is performing read operation on X after the Ta has made changes in X using W(X) so Tb can only commit after the commit operation of Ta.



```
Ta          Tb
-----       -----
R(X)
W(X)
            R(X)
            W(X)
            R(X)
commit
            commit
```

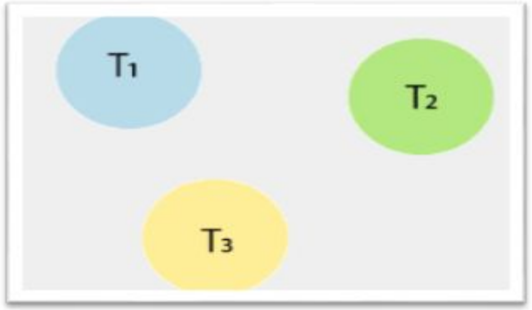
Serializability in DBMS

- When multiple **transactions are running concurrently then there is a possibility that the database may be left in an inconsistent state.**
- Serializability is a concept that **helps us to check which schedules are serializable. A serializable schedule is the one that always leaves the database in consistent state.**
- **What is a serializable schedule?**
- A **serial schedule** is always a serializable schedule because in serial schedule, a transaction only starts when the other transaction finished execution.
- **However a non-serial schedule needs to be checked for Serializability.**
- A non-serial schedule of n number of transactions is said to be serializable schedule, if it is equivalent to the serial schedule of those n transactions.

Serializability in DBMS

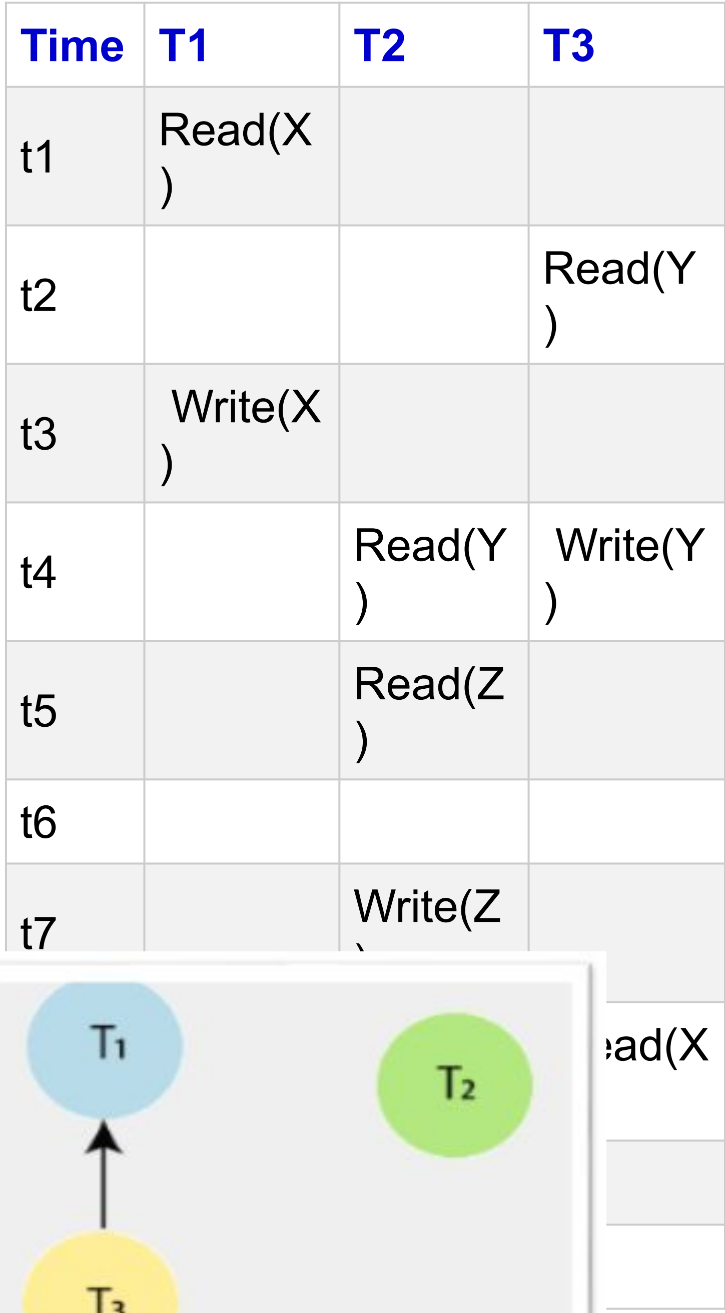
- **Types of Serializability**
- There are two types of Serializability.
- 1. **Conflict Serializability**
  2. **View Serializability**
- **Conflict Serializability** is one of the type of Serializability, which can be used to check whether a non-serial schedule is conflict serializable or not.
- A schedule is said to be conflict serializable if it can transform into a serial schedule after swapping of non-conflicting operations.
- **Conflicting operations**
- The operations are called conflicting operations, if all the following three conditions are satisfied:
- **Both the operation belongs to separate transactions.**
- **Both works on the same data item.**
- **At least one of them contains one write operation.**
- **Note: Conflict pairs for the same data item are:**
  Read-Write
  Write-Write
  Write-Read

Serializability in DBMS

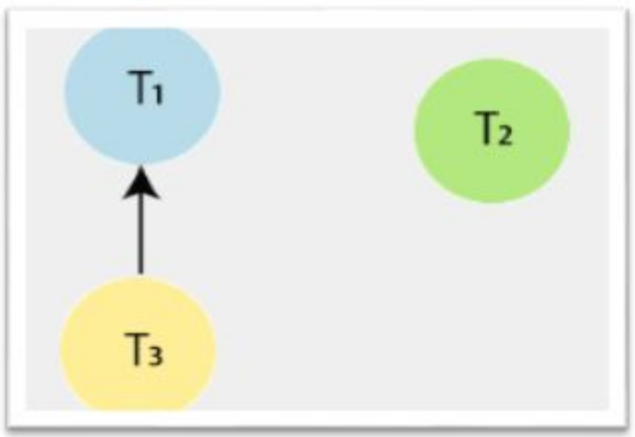


- **Conflict Serializability**
- There are three transactions: T1, T2, and T3.So, the precedence graph contains three vertices.

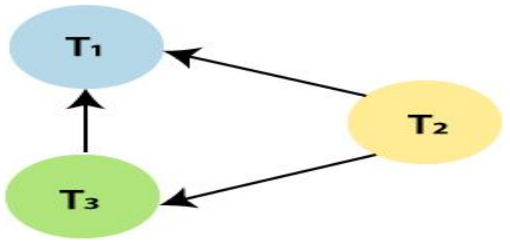
- To draw the edges between these nodes or vertices, follow the below steps:
- **Step1:** At time t1, there is no conflicting operation for **read(X)** of Transaction T1. **Step2:** At time t2, there is no conflicting operation for **read(Y)** of Transaction T3. **Step3:** At time t3, there exists a conflicting operation **Write(X)** in transaction T1 for **read(X)** of Transaction T3. So, draw an edge from T3?T1.



Time	T1	T2	T3
t1	Read(X)		
t2			Read(Y)
t3	Write(X)		
t4		Read(Y)	Write(Y)
t5		Read(Z)	
t6			
t7		Write(Z	
			ead(X



Serializability in DBMS



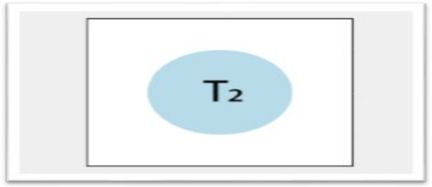
- **Conflict Serializability**
- **Step4: At time t4, there exists a conflicting operation Write(Y) in transaction T3 for read(Y) of Transaction T2. So, draw an edge from T2?T3.**
- **Step5:** At time t5, there exists a conflicting operation **Write (Z)** in transaction T1 for **read (Z)** of Transaction T2. So, draw an edge from T2?T1.

- **Step6:** At time t6, there is no conflicting operation for **Write(Y)** of Transaction T3. **Step7:** At time t7, there exists a conflicting operation **Write (Z)** in transaction T1 for **Write (Z)** of Transaction T2. So, draw an edge from T2?T1, but it is already drawn.

Time	T1	T2	T3
t1	Read(X)		
t2			Read(Y)
t3	Write(X)		Read(X)
t4			Write(Y)
t5		Read(Z)	
t6			
t7		Write(Z)	
t8	Read(Z)	Read(Y)	
t9			
t10	Write(Z)		

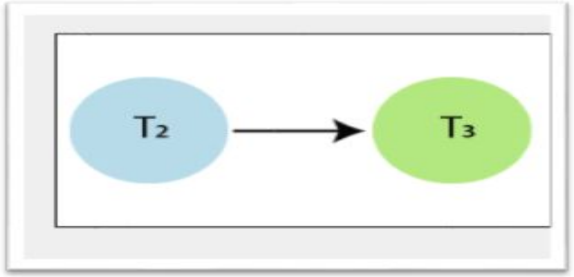
Serializability in DBMS

- **Conflict Serializability**
- After all the steps, the precedence graph will be ready, and it does not contain any cycle or loop, so the above schedule S2 is conflict serializable. And it is equivalent to a serial schedule. Above schedule S2 is transformed into the serial schedule by using the following steps:
- **Step1:** Check the vertex in the precedence graph where **indegree=0.** So, take the vertex T2 from the graph and remove it from the graph.



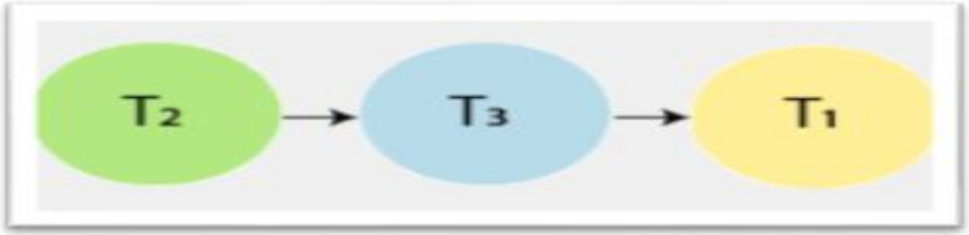
- **Step 2:** Again check the vertex in the left precedence graph where **indegree=0.** So, take the vertex T3 from the graph and remove it from the graph. And draw the edge from T2?T3.

Time	T1	T2	T3
t1	Read(X)		
t2			Read(Y)
t3			Read(X)
t4		Read(Y)	
t5		Read(Z)	
t6			Write(Y)
t7		Write(Z)	
t8	Read(Z)		



Serializability in DBMS

- **Conflict Serializability**
- **Step3:** And at last, take the vertex T1 and connect with T3 which is **Precedence graph equivalent to schedule S2**.



Time	T1	T2	T3
t1	Read(X)		
t2			Read(Y)
t3			Read(X)
t4		Read(Y)	
t5		Read(Z)	
t6			Write(Y)
t7		Write(Z)	
t8	Read(Z)		
t9	Write(X)		
t10	Write(Z)		

Converted Serial Schedu

Time	T1	T2	T3
t1		Read(Y)	
t2		Read(Z)	
t3		Write(Z)	
t4			Read(Y)
t5			Read(X)
t6			Write(Y)
t7	Read(X)		
t8	Read(Z)		
t9	Write(X)		
t10	Write(Z)		

Serializability in DBMS

- **View Serializability**
- **It is a type of serializability that can be used to check whether the given schedule is view serializable or not.** A schedule called as a view serializable if it is view equivalent to a serial schedule.
- **View Equivalent**
- Two schedules S1 and S2 are said to be view equivalent if both satisfy the following conditions:
- **1. Initial read**
- An initial read of the data item in both the schedule must be same. For example, lets two schedule S1 and S2. If transaction T1 reads the data item A in schedule S1, then in schedule S2 transaction T1 also reads A.

Schedule S1

Time	T1	T2
t1	Read(X)	
t2		Write(X)

Schedule S2

Time	T1	T2
t1		Write(X)
t2	Read(X)	

Above two schedules, S1 and S2 are view equivalent, because initial read instruction in S1 is done by T1 transaction and in schedule S2 is also done by transaction T2.

Serializability in DBMS

- **View Serializability**
- **2. Updated Read**
- In schedule S1, if the transaction T_i is reading the data item A which is updated by transaction T_j, then in schedule S2 also, T_i should read data item A which is updated by T_j.

Schedule S1

Time	T1	T2
t1	Write(X)	
t2		Read(X)

Schedule S2

Time	T1	T2
t1	Write(X)	
t2		Read(X)

Above two schedules S1 and S2 are view equivalent because in schedule S1 transaction T2 reads the data item A which is updated by T1 and in schedule S2 T2 also reads the data item A which is updated by T1.

Serializability in DBMS

- **View Serializability**
- **3. Final write**
- The final write operation on each data item in both the schedule must be same. In a schedule S1, if a transaction T1 updates data item A at last then in schedule S2, final writes operations should also be done by T1 transaction.

Schedule S1

Time	T1	T2	T3
t1			Write(X)
t2		Read(X)	
t3	Write(X		

Time	T1	T2	T3
t1			Write(X)
t2		Read(X)	
t3	Write(X)		

Above two schedules, S1 and S2 are view equivalent because final write operation in schedule S1 is done by T1 and in S2, T1 also does the final write operation.

Concurrency Control Protocols

- **Concurrency Control** in Database Management System is a **procedure of managing simultaneous operations without conflicting with each other.**
- **Concurrent access is quite easy if all users are just reading data.** There is no way they can interfere with one another.
- Though for **any practical Database, it would have a mix of READ and WRITE operations and hence the concurrency is a challenge**.
- **Problems of Concurrency**
- **Lost Updates** occur when multiple transactions select the same row and update the row based on the value selected
- **Uncommitted dependency issues** occur when the second transaction selects a row which is updated by another transaction
- **Non-Repeatable Read** occurs when a second transaction is trying to access the same row several times and reads different data each time.

Concurrency Control Protocols

- **Why use Concurrency method?**
- To **resolve read-write and write-write conflict issues**
- Concurrency control **helps to ensure serializability.**
- **Example**
- Assume that two people who go to electronic kiosks at the same time to buy a movie ticket for the same movie and the same show time.
- However, there is only one seat left in for the movie show in that particular theatre. Without concurrency control in DBMS, it is possible that both moviegoers will end up purchasing a ticket. However, concurrency control method does not allow this to happen. Both moviegoers can still access information written in the movie seating database. But concurrency control only provides a ticket to the buyer who has completed the transaction process first.

Concurrency Control Protocols

- Different concurrency control protocols offer different benefits between the amount of concurrency they allow and the amount of overhead that they impose. Following are the Concurrency Control techniques in DBMS:
- **Lock-Based Protocols**
- **Two Phase Locking Protocol**
- **Timestamp-Based Protocols**
- **Validation-Based Protocols**
- **Lock-based Protocols**
- **Lock Based Protocols** in DBMS is a **mechanism in which a transaction cannot Read or Write the data until it acquires an appropriate lock. Lock based protocols help to eliminate the concurrency problem in DBMS** for simultaneous transactions by locking or isolating a particular transaction to a single user.
- A **lock is a data variable which is associated with a data item.** This lock signifies that operations that can be performed on the data item.

Concurrency Control Protocols

- **Lock-based Protocols**
- **Binary Locks:** A Binary lock on a data item can **either locked or unlocked states.**
- **Shared/exclusive:** This type of locking mechanism separates the locks in DBMS **based on their uses.** If a lock is acquired on a data item to perform a write operation, it is called an exclusive lock.
- **1. Shared Lock (S):**
- A shared lock is also called a **Read-only lock.** With the shared lock, the data item can be shared between transactions. This is because you will never have permission to update data on the data item.
- For example, consider a case where two transactions are reading the account balance of a person. The database will let them read by placing a shared lock. However, if another transaction wants to update that account's balance, shared lock prevent it until the reading process is over.

Concurrency Control Protocols

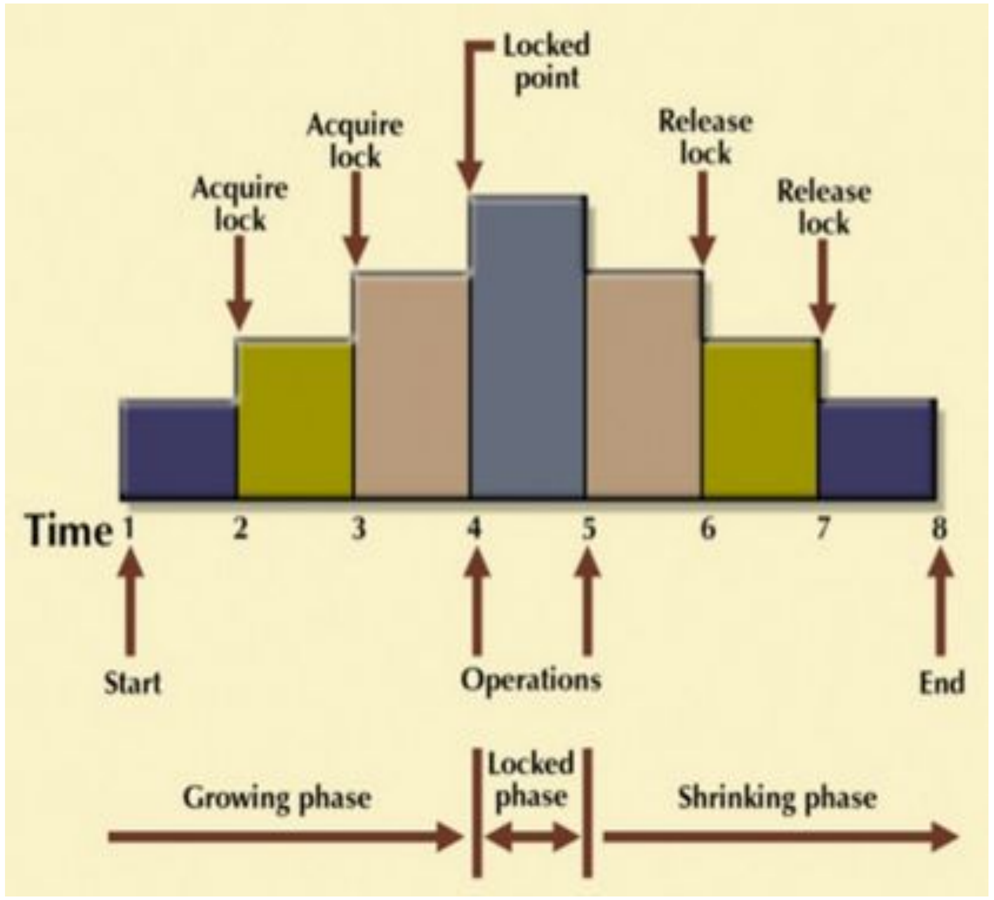
- **Lock-based Protocols**
- **2. Exclusive Lock (X):**
- With the Exclusive Lock, a **data item can be read as well as written**. This is exclusive and can’t be held concurrently on the same data item. **X-lock is requested using lock-x instruction.** Transactions may unlock the data item after finishing the ‘write’ operation.
- For example, when a transaction needs to update the account balance of a person. You can allows this transaction by placing X lock on it. Therefore, when the second transaction wants to read or write, exclusive lock prevent this operation.

Concurrency Control Protocols

- **Two Phase Locking Protocol**
- **Two Phase Locking Protocol** also known as 2PL protocol is a method of concurrency control in DBMS that **ensures serializability by applying a lock to the transaction data which blocks other transactions to access the same data simultaneously.**
- This locking protocol divides the execution phase of a transaction into three different parts.
- In the first phase, when the transaction begins to execute, **it requires permission for the locks it needs.**
- The second part is **where the transaction obtains all the locks.** When a transaction releases its first lock, the third phase starts.
- In this third phase, the transaction cannot demand any new locks. Instead, it **only releases the acquired locks.**

Concurrency Control Protocols

- **Two Phase Locking Protocol**
- The Two-Phase Locking protocol allows each transaction to make a lock or unlock request in two steps:
- **Growing Phase**: In this phase transaction may obtain locks but may not release any locks.
- **Shrinking Phase**: In this phase, a transaction may release locks but not obtain any new lock



Concurrency Control Protocols

- **Timestamp-based Protocols**
- **Timestamp based Protocol** in DBMS is an algorithm **which uses the System Time or Logical Counter as a timestamp to serialize the execution of concurrent transactions.** The Timestamp-based protocol ensures that every conflicting read and write operations are executed in a timestamp order.
- **The older transaction is always given priority in this method**. It uses system time to determine the time stamp of the transaction.
- **Suppose there are three transactions T1, T2, and T3.**
- **T1 has entered the system at time 0010**
- **T2 has entered the system at 0020**
- **T3 has entered the system at 0030**
- **Priority will be given to transaction T1, then transaction T2 and lastly Transaction T3.**

Concurrency Control Protocols

- **Validation Based Protocol**
- **Validation based Protocol** in DBMS also known as Optimistic Concurrency Control Technique is a method to avoid concurrency in transactions. In this protocol, the local copies of the transaction data are updated rather than the data itself, which results in less interference while execution of the transaction.
- The Validation based Protocol is performed in the following three phases:

1. **Read Phase**
2. **Validation Phase**
3. **Write Phase**

- **Read Phase**
- In the Read Phase, the data values from the database can be read by a transaction **but the write operation or updates are only applied to the local data copies, not the actual database.**
- **Validation Phase**
- In Validation Phase, **the data is checked to ensure that there is no violation of serializability while applying the transaction updates to the database.**
- **Write Phase**
- In the Write Phase, **the updates are applied to the database if the validation is successful,** else; **the updates are not applied, and the transaction is rolled back.**

Recovery techniques

- **Database systems**, like any other computer system, **are subject to failures but the data stored in them must be available as and when required.**
- **When a database fails it must possess the facilities for fast recovery.** It must also have atomicity i.e. either transaction are completed successfully and committed (the effect is recorded permanently in the database) or the transaction should have no effect on the database.
- The techniques used to recover the **lost data due to system crashes, transaction errors, viruses, catastrophic failure, incorrect commands execution, etc. are database recovery techniques.**
- Recovery techniques are heavily dependent upon the existence of a special file known as a **system log**.
- **It contains information about the start and end of each transaction and any updates which occur during the transaction.** The log keeps track of all transaction operations that affect the values of database items.

Recovery techniques

- This information is needed to recover from transaction failure.
- **The log is kept on disk start_transaction(T):** This log entry records that transaction T starts the execution.
- **read_item(T, X):** This log entry records that transaction T reads the value of database item X.
- **write_item(T, X, old_value, new_value):** This log entry records that transaction T changes the value of the database item X from old_value to new_value. The old value is sometimes known as a before an image of X, and the new value is known as an afterimage of X.
- **commit(T):** This log entry records that transaction T has completed all accesses to the database successfully and its effect can be committed (recorded permanently) to the database.
- **abort(T):** This records that transaction T has been aborted.
- **checkpoint:** Checkpoint is a mechanism where all the previous logs are removed from the system and stored permanently in a storage disk. Checkpoint declares a point before which the DBMS was in a consistent state, and all the transactions were committed.

Recovery techniques

- **Undoing –** If a transaction crashes, then the recovery manager may undo transactions i.e. **reverse the operations of a transaction. This involves examining a transaction for the log entry write_item(T, x, old_value, new_value) and set the value of item x in the database to old-value.**
- **Deferred update – This technique does not physically update the database on disk until a transaction has reached its commit point.** Before reaching commit, all transaction updates are recorded in the local transaction workspace. If a transaction fails before reaching its commit point, it will not have changed the database in any way so UNDO is not needed.
- **Immediate update –** In the immediate update, the database may be updated by some operations of a transaction before the transaction reaches its commit point.
- **Caching/Buffering –** In this one or more disk pages that include data items to be updated are cached into main memory buffers and then updated in memory before being written back to disk.

Recovery techniques

- **Shadow paging –**When a transaction began executing the current directory is copied into a shadow directory. **When a page is to be modified, a shadow page is allocated in which changes are made and when it is ready to become durable, all pages that refer to the original are updated to refer new replacement page.**
- **Backward Recovery** – The term "Rollback " and "UNDO" can also refer to backward recovery. **When a backup of the data is not available and previous modifications need to be undone, this technique can be helpful.** With the backward recovery method, unused modifications are removed and the database is returned to its prior condition.
- **Forward Recovery** – "Roll forward "and "REDO" refers to forwarding recovery. When a database needs to be updated with all changes verified, this forward recovery technique is helpful.