DATABASE MANAGEMENT SYSTEMS – CSE/IT – II Year / IV Sem. – 22CS403

Manju S,
AP/CoE,
RTC,Coimbatore.

TOPICS

- **TRANSACTIONS AND IMPLEMENTATION TECHNIQUES**
  - Transaction Concepts -ACID Properties – Transaction States-Schedules -Serializability-Concurrency Control-Need for Concurrency-Locking Protocols -Two Phase Locking.

Transaction

- Transactions in Database Management Systems (DBMS) are sets of operations performed to modify data, including insertion, updates, or deletions.
- These transactions have various states indicating their progress and required actions.
- They ensure data consistency even during system failures, demonstrating a key advantage of DBMS.
- Transactions, executed repeatedly, like ATM withdrawals, generate multiple instances, each maintaining database integrity through specific properties.

ATM withdrawal transaction

- E.g.: When we go to withdraw money from ATM, we encounter the following set of operations:
  - Transaction Initiated
  - You have to insert an ATM card
  - Select your choice of language
  - Select whether savings or current account
  - Enter the amount to withdraw
  - Entering your ATM pin
  - Transaction processes
  - You collect the cash
  - You press finish to end transaction

Transaction example : RAM vs. Secondary Memory

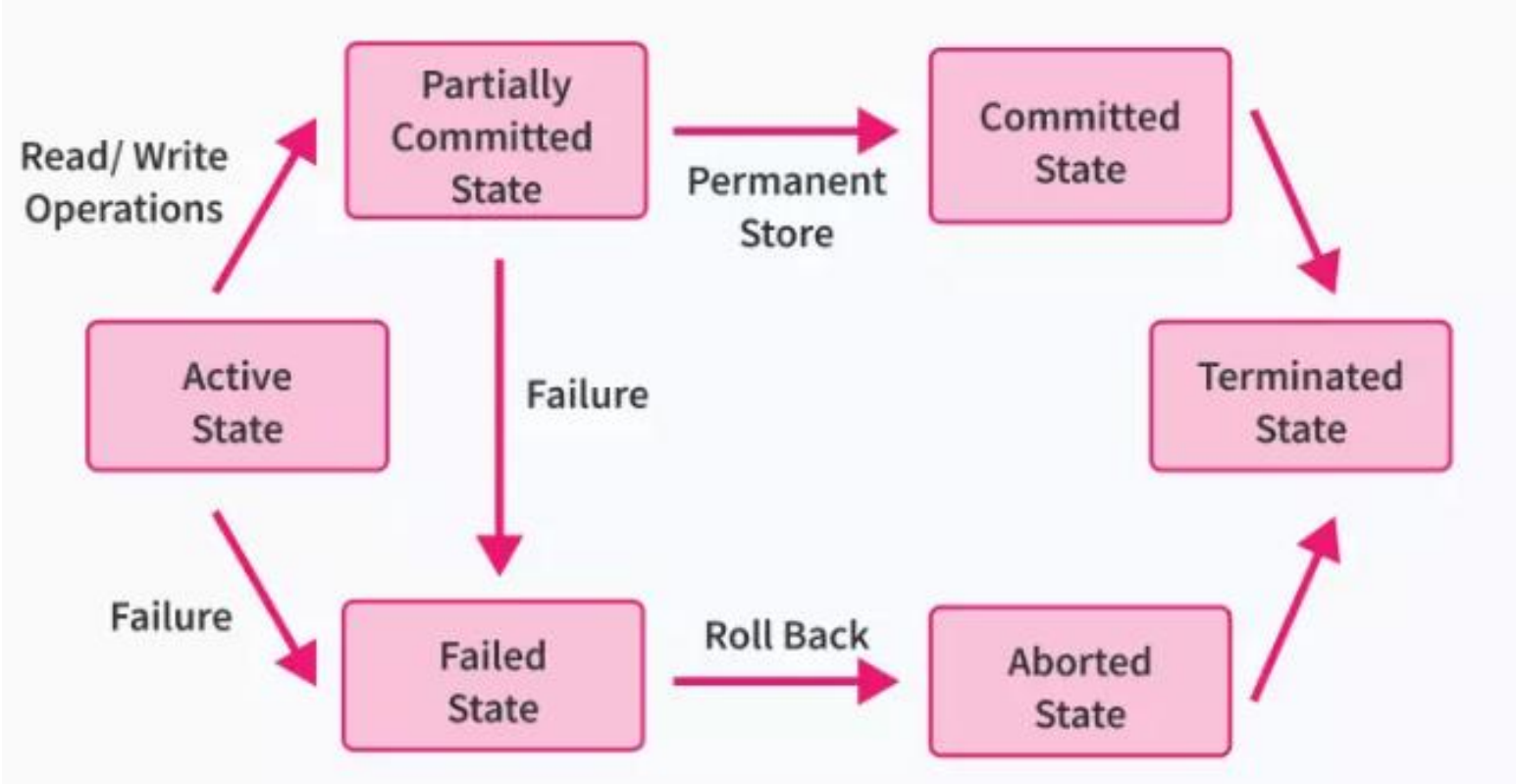
- E.g.: Account1 has - ₹ 5000 and Account2 - ₹ 2000
- This data before the start of the transaction is stored in the secondary memory (Hard disk) which once initiated is bought to the primary memory (RAM) of the system for faster and better access.
- Now for a transfer of ₹ 500 from Account1 to Account2 to occur, the following set of operations will take place.
- Read (Account1) --> 5000 Account1 = Account1 - 500 Write (Account1) --> 4500 Read (Account2) --> 2000 Account2 = Account2 + 500 Write (Account2) --> 2500 commit

Transaction example : RAM vs. Secondary Memory

- The COMMIT statement permanently saves the changes made by the current transaction.
- When a transaction is successful, COMMIT is applied. If the system fails before a COMMIT is applied, the transaction reaches its previous state after ROLLBACK.
- After commit operation the transaction ends and updated values of Account1 = ₹ 4500 and Account2 = ₹ 2500. Every single operation that occurs before the commit is said to be in a partially committed state and is stored in the primary memory (RAM).
- After the transaction is committed, the updated data is accepted and updated in the secondary memory (Hard Disk).

Transaction States in DBMS

- 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 and decide the fate of a transaction whether it will commit or abort.



Transaction States in DBMS

- **Active State**
  - When the operations of a transaction are running then the transaction is said to be active state.
  - If all the read and write operations are performed without any error then it progresses to the partially committed state, if somehow any operation fails, then it goes to a state known as failed state.
- **Partially Committed**
  - After all the read and write operations are completed, the changes which were previously made in the main memory are now made permanent in the database, after which the state will progress to committed state but in case of a failure it will go to the failed state.

Transaction States in DBMS : Failures

- **Failed State**
  - If any operation during the transaction fails due to some software or hardware issues, then it goes to the *failed state* .
  - The occurrence of a failure during a transaction makes a permanent change to data in the database. The changes made into the local memory data are rolled back to the previous consistent state.
- **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 stat

Transaction States in DBMS : Success

- **Committed State**
  - If the transaction completes all sets of operations successfully, all the changes made during the partially committed state are permanently stored and the transaction is stated to be completed, thus the transaction can progress to finally get terminated in the terminated state.
- **Terminated State**
  - If the transaction gets aborted after roll-back or the transaction comes from the committed state, then the database comes to a consistent state and is ready for further new transactions since the previous transaction is now terminated.

Note!

- The ROLLBACK statement undo the changes made by the current transaction.
- A transaction cannot undo changes after COMMIT execution.

Properties of a transaction

- There are four major properties that are vital for a transaction to be successful.
- These are used to maintain state consistency in the database, both before and after the transaction.
- These are called **ACID** properties.
  - Atomicity
  - Consistency
  - Isolation
  - Durability

Properties of a transaction : Atomicity : ACID

- **Atomicity**
  - This property means that either the transaction takes place completely at once or doesn’t happen at all.
  - There is no middle option, i.e., transactions do not occur partially.
  - Each transaction is considered as one single step which either runs completely or is not executed at all.

Properties of a transaction : Atomicity : **ACID**

- **Consistency**
  - This property means that the integrity constraints of a database are maintained so that the database is consistent before and after the transaction.
  - It refers to the correctness of the data in database.

Properties of a transaction : Atomicity : **ACID**

- **Isolation**
  - This property means that multiple transactions can occur concurrently without causing any inconsistency to the database state.
  - Isolation is an ACID Property in DBMS where no data from one database should impact the other and where many transactions can take place at the same time
  - Changes that occur in a particular transaction are not visible/ accessible to any other transaction until that particular change in that transaction has been committed.

Properties of a transaction : Atomicity : **ACID**

- **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 remain intact even if a system failure occurs.
  - These updates become permanent and are stored in the non-volatile memory.

Ensuring ACID properties : Atomicity

- **Atomicity in Online Ticket Booking Systems**
  - Let's take the example of an online ticket booking system, a booking may consist of two discrete acts that combine to make a transaction:
  - the first one is the payment for the ticket and then the second is to reserve the seat for the person who just paid.
  - Business logic mandates that these two different and separate actions must take place concurrently.
  - Problems can arise if one occurs without the other.
  - The system, for example, may reserve the same seat for two different consumers.
  - Solution : Rollback in case of failure!

Ensuring ACID properties : Consistency

- Same example where one person is trying to book a ticket.
  - They are able to reserve their seat but their payment hasn’t gone through due to bank issues. In this case, their transaction is rolled back.
  - But just doing that isn’t sufficient. The number of available seats must also be updated.
  - Otherwise, if it isn’t updated, there will be an inconsistency where the seat given up by the person is not accounted for.
  - Hence, the total sum of seats left in the train + the sum of seats booked by users would not be equal to the total number of seats present in the train if not for consistency.
  - Solution: Ensure rules and constraints!

Ensuring ACID properties : Isolation

- Suppose two people try to book the same seat simultaneously.
  - Transactions are serialized to maintain data consistency.
  - The first person's transaction succeeds, and they receive a ticket.
  - The second person's transaction fails as the seat is already booked.
  - They receive an error message indicating no available seats.
  - Solution : Data to be isolated!

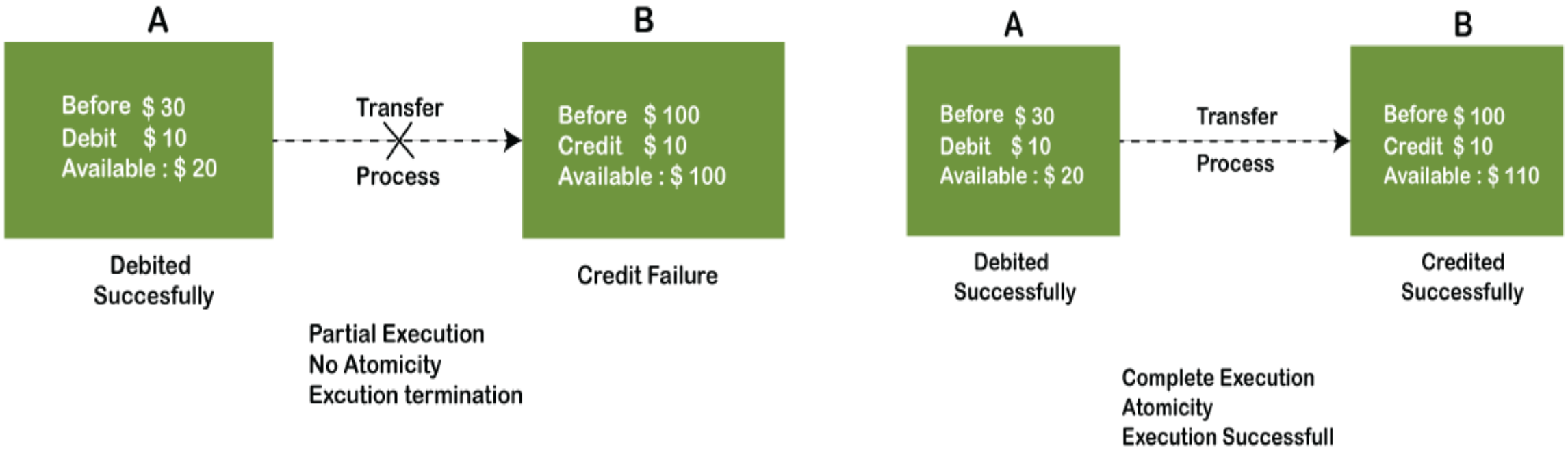
Ensuring ACID properties : Duration

- Suppose that there is a system failure in the railway management system resulted in the loss of all booked train details.
  - Millions of users who had paid for their seats are now unable to board the train, causing significant financial losses and destroying trust in the company.
  - Solution : Save data permanently !

Advantages of ACID properties in DBMS

- **Data Integrity**: ACID properties ensure data remains consistent and free from corruption.
- **Reliability**: ACID properties provide consistent and reliable execution of transactions.
- **Concurrency Control**: ACID properties enable simultaneous access to data without conflicts.
- **Fault Tolerance**: ACID properties ensure data durability, surviving system failures.
- **Transaction Management**: ACID properties offer structured transaction handling.

Yet another atomicity example !



- Either all should happen or nothing! And hence incorrect data at first failed transaction!

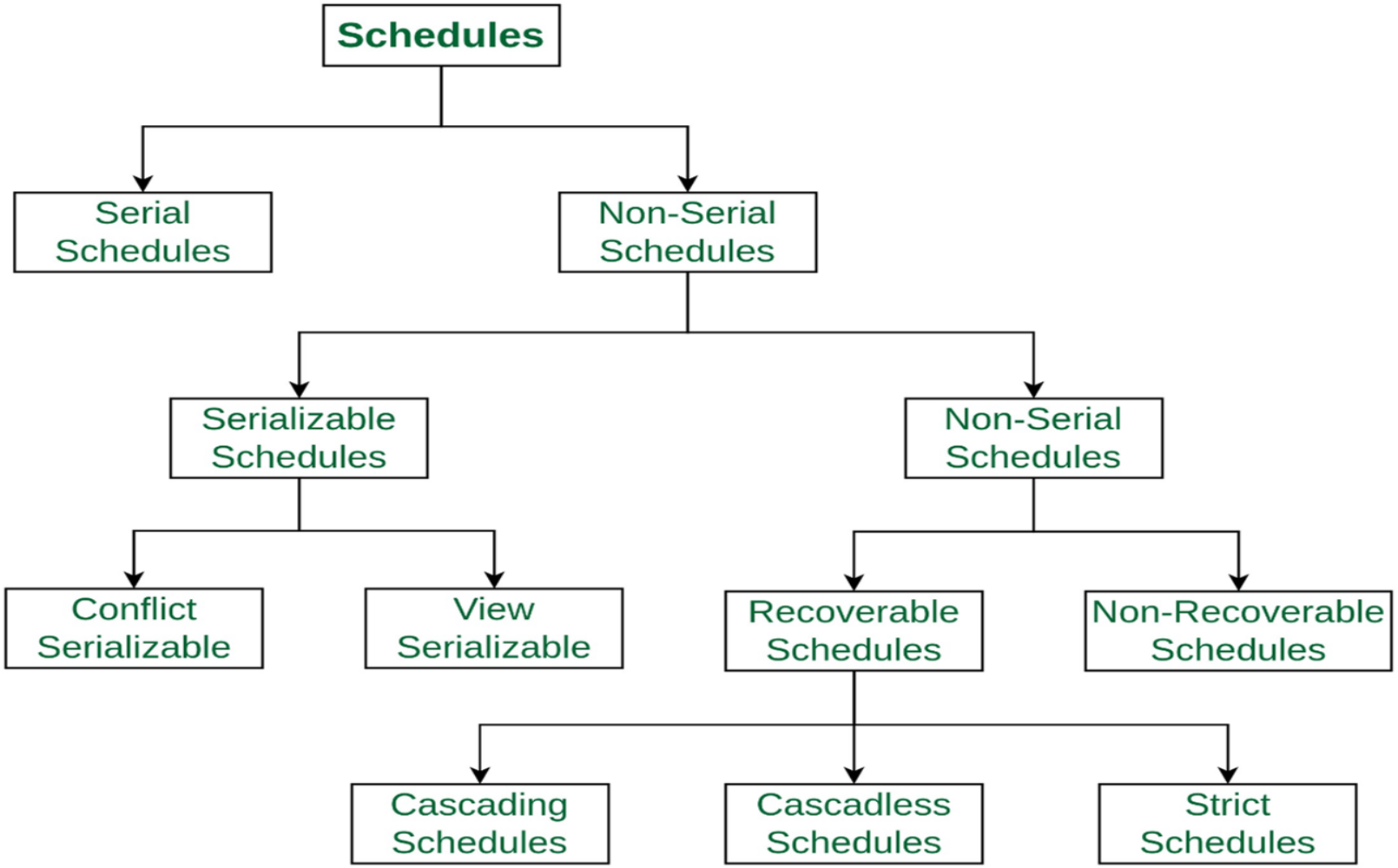
What is Scheduling in DBMS?

- Transactions are a set of instructions that perform operations on databases. When multiple transactions are running concurrently, then a sequence is needed in which the operations are to be performed because *at a time, only one operation can be performed on the database*.
- This sequence of operations is known as Schedule, and this process is known as Scheduling.

Why scheduling ?

- When multiple transactions execute simultaneously in an unmanageable manner, then it might lead to several problems, which are known as concurrency problems.
- In order to overcome these problems, scheduling is required.

Types of schedules



Serial schedule

- As the name says, all the transactions are executed **serially one after the other**.
- In serial Schedule, a transaction does not start execution until the currently running transaction finishes execution
- This type of execution of the transaction is also known as non-interleaved execution. Serial Schedule are always recoverable, and consistent.
- A serial schedule always gives the correct result.

Serial schedule : Example

- Either execute all T1 operations, which were followed by all T2 operations.
- Or execute all T2 operations, which were followed by all T1 operations
- If n = number of transactions, then a number of serial schedules possible = n!.
- a total number of serial schedules possible = 2.

Transaction T1	Transaction T2
R(A)	
W(A)	
R(B)	
W(B)	
commit	
	R(A)
	W(B)
	commit

Non Serial Schedules

- In a non-serial Schedule, multiple transactions execute concurrently/simultaneously, unlike the serial Schedule, where one transaction must wait for another to complete all its operations.
- In the Non-Serial Schedule, the other transaction proceeds without the completion of the previous transaction.
- All the transaction operations are interleaved or mixed with each other.
- Non-serial schedules are NOT always recoverable and consistent.

Non Serial schedules : Example

- There are two transactions, T1 and T2, executing concurrently.
- The operations of T1 and T2 are interleaved.
- Non-serial schedules are further categorized into serializable and non-serializable schedules.

Transaction T1	Transaction T2
R1(A)	
W1(A)	
	R2(A)
	W2(A)
R1(B)	
W1(B)	
	R2(B)
	W2(B)

Serializability

- Serializability in DBMS is a concept that helps to identify which non-serial schedules are correct and will maintain the consistency of the database.
- A serializable schedule always leaves the database in a consistent state.
- A **serial schedule is always a serializable schedule** because, in a serial Schedule, a transaction only starts when the other transaction has finished execution.
- A non-serial schedule of n transactions is said to be a serializable schedule, if it is equivalent to the serial Schedule of those n transactions

Serial Schedules vs. Serializable schedules !

Serial Schedules	VS	Serializable Schedules
No concurrency is allowed. Thus, all the transactions necessarily execute serially one after the other.		Concurrency is allowed. Thus, multiple transactions can execute concurrently.
Serial schedules lead to less resource utilization and CPU throughput.		Serializable schedules improve both resource utilization and CPU throughput.
Serial Schedules are less efficient as compared to serializable schedules. (due to above reason)		Serializable Schedules are always better than serial schedules. (due to above reason)

Types of Serializability : Conflict

- **Conflict Serializability**
  - A schedule is called conflict serializable if it can be transformed into a serial schedule by swapping non-conflicting operations.
  - An operations pair become conflicting if all conditions satisfy:
    - Both belong to separate transactions.
    - They have the same data item.
    - They contain at least one write operation.

Conflict Serializability : What's conflict ?

1. T1: Read(A) T2: Read(A)

T1	T2
Read(A)	
	Read(A)

Swapped

T1	T2
	Read(A)
Read(A)	

Schedule S1 **Schedule S2**

Here, S1 = S2. That means it is non-conflict.

2. T1: Read(A) T2: Write(A)

T1	T2
Read(A)	
	Write(A)

Swapped

T1	T2
	Write(A)
Read(A)	

Schedule S1 **Schedule S2**

Here, S1 ≠ S2. That means it is conflict.

Conflict Serializability : Swapping :Example

- Write(A)/Read(A) and Write(B)/Read(B) are called as conflicting operations.
- Thus, by swapping(non-conflicting) 2nd pair of the read/write operation of 'A' data item and 1st pair of the read/write operation of 'B' data item, this non-serial Schedule can be converted into a serializable Schedule.
- Therefore, it is conflict serializable.

Transaction T1	Transaction T2
Read(A)	
Write(A)	
	Read(A)
	Write(A)
Read(B)	
Write(B)	
	Read(B)
	Write(B)

Before Swapping

Transaction T1	Transaction T2
Read(A)	
Write(A)	
Read(B)	
Write(B)	
	Read(A)
	Write(A)
	Read(B)
	Write(B)

After Swapping

Types of Serializability : View

- **View Serializability**
  - A schedule is view serializable if it is view equivalent to a serial schedule.
  - If a schedule is conflict serializable, then it will be view serializable.

View Equivalent conditions

- S1 and S2 are said to view equivalent if they meet the following conditions:
  - **Initial Read:**
    The initial read of each data item in transactions must match both schedules. For example, if transaction T1 reads data item Y before transaction T2 in schedule S1, then in schedule S2, T1 should read Y before T2.

View Equivalent conditions

- **Final Write:**
  In both schedules, the final write operations on each data item must match. For example, if a data item Y was last written by Transaction T1 in schedule S1, the last write operation on Y should be performed by Transaction T1 in schedule S2.
- **Update Read:**
  If transaction T1 in schedule S1 is reading a data item updated by T2, then in schedule S2, T1 should read the value after T2's write operation on the same data item. For example, if in schedule S1, T1 performs a read operation on X after T2 performs a write operation on Y, then in schedule S2, T1 should read the Y after T2 performs a write operation on Y.

View Serializability

- S2 is the serial schedule of S1
- S2 and S1 are view equivalent
- And hence we can say that given S1 is view Serializable!

Non-Serial		Serial	
S1		S2	
Transaction T1	Transaction T2	Transaction T1	Transaction T2
R(X)		R(X)	
W(X)		W(X)	
	R(X)	R(Y)	
	W(X)	W(Y)	
R(Y)			R(X)
W(Y)			W(X)
	R(Y)		R(Y)
	W(Y)		W(Y)

Concurrency control and need for concurrency!

- Concurrency : Executing more transactions in parallel
- A transaction is a single reasonable unit of work that can retrieve or may change the data of a database.
- Executing each transaction individually increases the waiting time for the other transactions and the overall execution also gets delayed.
- Hence, to **increase the throughput** and to reduce the waiting time, transactions are executed concurrently.

Why concurrency is needed ?

- Suppose, between two railway stations, **A** and **B**, 5 trains have to travel, if all the trains are set in a row and only one train is allowed to move from station **A** to **B** and others have to wait for the first train to reach its destination then it will take a lot of time for all the trains to travel from station **A** to **B**.
- To reduce time all the trains should be allowed to move concurrently from station **A** to **B** ensuring no risk of collision between them.

Concurrency control in DBMS

- When several transactions execute simultaneously, then there is a risk of violation of the data integrity of several databases.
- Concurrency Control in DBMS is a procedure of managing simultaneous transactions ensuring their atomicity, isolation, consistency and serializability.
- Problems that rise in concurrent transactions
  - Dirty Read Problem
  - Unrepeatable Read Problem
  - Phantom Read Problem
  - Lost Update Problem

Dirty Read Problem

- Occurs when a transaction reads the data that has been updated by another transaction that is still uncommitted.
- Transaction A reads the value of data DT as 1000 and modifies it to 1500 which gets stored in the temporary buffer.
- The transaction B reads the data DT as 1500 and commits it and the value of DT permanently gets changed to 1500 in the database DB.
- Then some server errors occur in transaction A and it wants to get rollback to its initial value, i.e., 1000 and then the dirty read problem occurs.

Assume initial value of DT to be 1000 in all upcoming slides

Time	A	B
T1	READ(DT)	------
T2	DT=DT+500	------
T3	WRITE(DT)	------
T4	------	READ(DT)
T5	------	COMMIT
T6	ROLLBACK	------

Unrepeatable read problem

- The unrepeatable read problem occurs when two or more different values of the same data are read during the read operations in the same transaction.
- Transaction A and B initially read the value of DT as 1000.
- Transaction A modifies the value of DT from 1000 to 1500 and then again transaction B reads the value and finds it to be 1500.
- Transaction B finds two different values of DT in its two different read operations.

Time	A	B
T1	READ(DT)	------
T2	------	READ(DT)
T3	DT=DT+500	------
T4	WRITE(DT)	------
T5	------	READ(DT)

Phantom read problem

- In the phantom read problem, data is read through two different read operations in the same transaction. In the first read operation, a value of the data is obtained but in the second operation, an error is obtained saying the data does not exist.
- Transaction A deletes the data DT from the database DB and then again transaction B reads the value and finds an error saying the data DT does not exist in the database DB.

Time	A	B
T1	READ(DT)	------
T2	------	READ(DT)
T3	DELETE(DT)	------
T4	------	READ(DT)

Lost update problem

- Arises when an update in the data is done over another update but by two different transactions.
- Transaction A initially reads the value of DT as 1000. Transaction A modifies the value of DT from 1000 to 1500 and then again transaction B modifies the value to 1800.
- Transaction A again reads DT and finds 1800 in DT and therefore the update done by transaction A has been lost.

Time	A	B
T1	READ(DT)	------
T2	DT=DT+500	------
T3	WRITE(DT)	------
T4	------	DT=DT+300
T5	------	WRITE(DT)
T6	READ(DT)	------

Concurrency control protocol

- To avoid concurrency control problems and to maintain consistency and serializability during the execution of concurrent transactions some rules are made.
- These rules are known as Concurrency Control Protocols.
  - Lock based protocols

Lock based protocol

- To attain consistency, isolation between the transactions is the most important tool.
- Isolation is achieved if we disable the transaction to perform a read/write operation.
- This is known as locking an operation in a transaction. Through lock-based protocols, desired operations are freely allowed to perform locking the undesired operations.

2 types of locks in lock based protocol

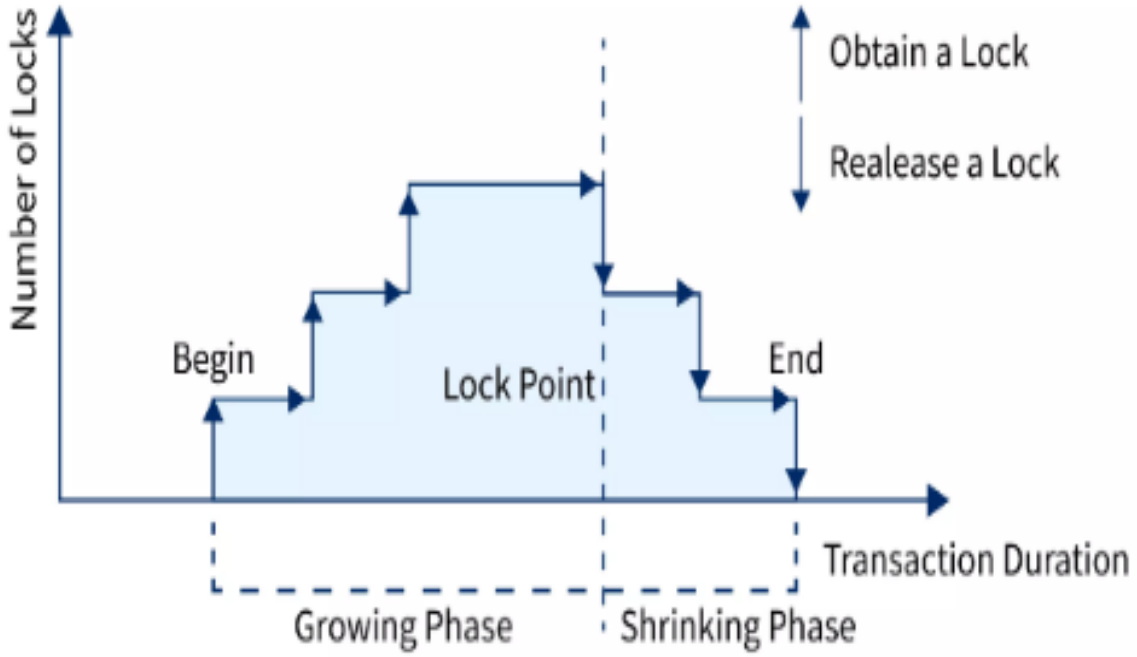
- **Shared Lock(S):**
  - The locks which *disable the write operations* but *allow read operations* for any other transaction are known as shared locks.
  - They are also known as read-only locks and are **represented by 'S'**.
- **Exclusive Lock(X):**
  - The locks which don’t *allow both the read and write operations for any other transactions* are known as exclusive locks.
  - This is a one-time use mode that can't be utilized on the exact data item twice. They are **represented by 'X'.**

2 types of locking protocols

- **Two-phase Locking Protocol:** The transaction starts its execution with the first phase, where it asks for the locks. Once the locks are granted, the second phase begins, where the transaction contains all the locks. When the transaction releases the first lock, the third phase begins where all the locks are getting released after the execution of every operation in the transaction.
- **Strict Two-Phase Locking Protocol:** The strict 2PL is almost similar to 2PL. The only difference is that the strict 2PL does not allow releasing the locks just after the execution of the operations, but it carries all the locks and releases them when the commit is triggered

2 PL : Two phase Locking Protocol

- The two phases are known as the growing and shrinking phases.
  - **Growing Phase**: In this phase, we can acquire new locks on data items, but none of these locks can be released.
  - **Shrinking Phase**: In this phase, the existing locks can be released, but no new locks can be obtained.



2 PL : Two phase Locking Protocol

- **Transaction T1:**
  - **Growing phase:** from step 0-2
  - **Shrinking phase:** from step 4-6
  - **Lock point:** at 3
- **Transaction T2:**
  - **Growing phase:** from step 1-5
  - **Shrinking phase:** from step 7-9
  - **Lock point:** at 6

	T1	T2
0	LOCK-S(A)	
1		LOCK-S(A)
2	LOCK-X(B)	
3	——	——
4	UNLOCK(A)	
5		LOCK-X(C)
6	UNLOCK(B)	
7		UNLOCK(A)
8		UNLOCK(C)
9	——	——

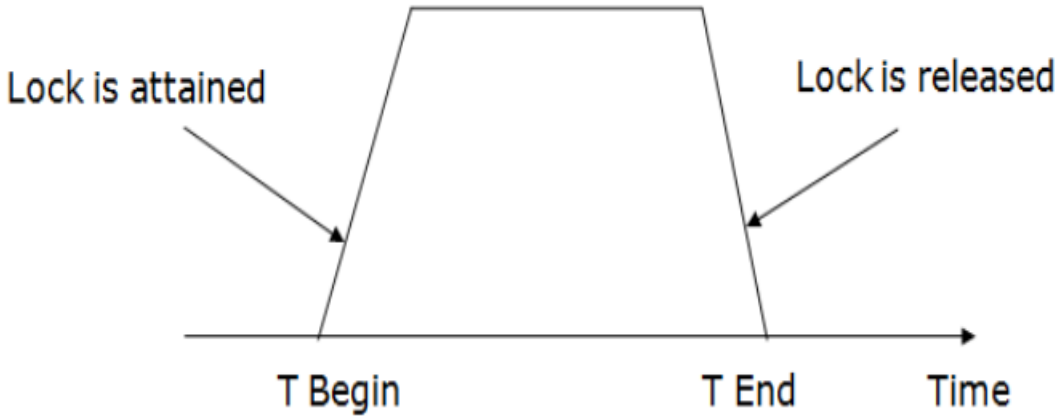
2 PL : Two phase Locking Protocol : Lock Point

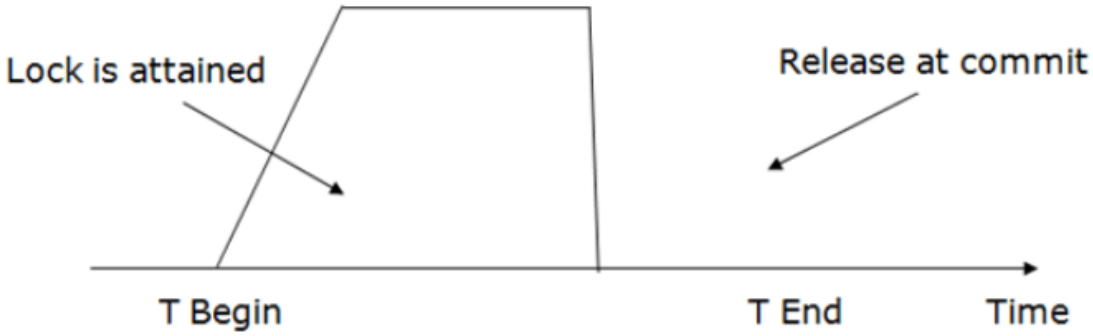
- The lock point is the moment when a transaction switches from the growing phase to the shrinking phase, and releases its first lock.
- The lock point is also the point at which the transaction has acquired all the locks it may need.
- After the lock point, the transaction enters a shrinking phase

Strict 2 PL : Two phase Locking Protocol

- The first phase of Strict-2PL is similar to 2PL. In the first phase, after acquiring all the locks, the transaction continues to execute normally.
- The only difference between 2PL and strict 2PL is that Strict-2PL does not release a lock after using it.
- Strict-2PL waits until the whole transaction to commit, and then it releases all the locks at a time.
- Strict-2PL protocol does not have shrinking phase of lock release.

2PL vs. strict 2PL ! Guess which is Strict 2PL?





2PL : Locks will be released after the execution of transaction and even before commit

Strict 2PL : Locks will be released only after commit

RATHINAM

Thank you!