Transaction Processing; Concurrency Control and Recovery

Ram Datta Bhatta

What is a transaction?

- ✓ A transaction is a logical unit of work that consists of one or more database operations.
- ✓ The operations in a transaction can be read, write, or both, and they are treated as a single indivisible unit
- ✓ Example: Bank balance transfer of Rs100 from account A to account B

Transaction Management

Examples: Operations in an ATM transaction

- Transaction Start.
- Insert your ATM card.
- Select language for your transaction.
- Select Savings Account option.
- Enter the amount you want to withdraw.
- Enter your secret pin.
- Wait for some time for processing.
- Collect your Cash.
- Transaction Completed.

Transaction Management

Three operations can be performed in a transaction as follows.

- Read/Access data (R).
- Write/Change data (W).
- Commit.

Example –

Transfer of Rs.50 from Account A to Account B. Initially A= Rs 500, B= Rs 800. This data is brought to RAM from Hard Disk.

- ✓ R(A) -- 500 // Accessed from RAM.
- ✓ A = A-50 // Deducting Rs.50 from A.
- ✓ W(A)--450 // Updated in RAM.
- ✓ R(B) -- 800 // Accessed from RAM.
- ✓ B=B+50 // Rs 50 is added to B's Account.
- ✓ W(B) --850 // Updated in RAM.
- ✓ commit // The data in RAM is taken back to Hard Disk.

Transaction Management

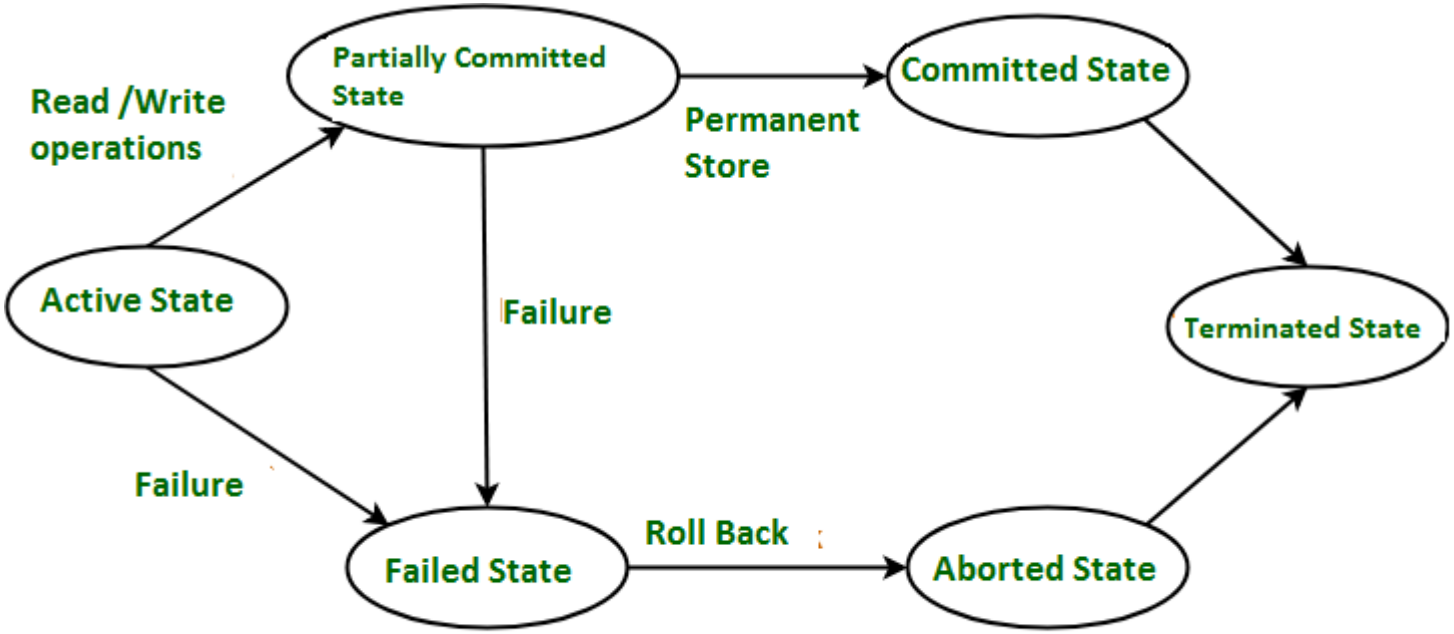
- ✓ The updated value of Account A = Rs 450 and Account B = Rs 850.
- ✓ All instructions before commit come under a **partially committed** state and are stored in RAM. When the **commit** is read the data is fully accepted and is stored in Hard Disk.
- ✓ If the data is failed anywhere before commit we have to go back and start from the beginning. We can't continue from the same state. This is known as **Roll Back**.

Uses of Transaction Management :

- ✓ The DBMS is used to schedule the access of data concurrently. It means that the user can access multiple data from the database without being interfered with each other. Transactions are used to manage concurrency.
- ✓ It is also used to satisfy ACID properties.

Transaction States/Cycle

States through which a transaction goes during its lifetime. These are the states which tell about the current state of the Transaction and also tell how we will further do the processing in the transactions.



Transaction States in DBMS

Transaction States in DBMS

These are different types of Transaction States :

- **Active State**

  When the instructions of the transaction are running then the transaction is in active state. If all the ‘read and write’ operations are performed without any error then it goes to the “partially committed state”, if any instruction fails, it goes to the “failed state”.

- **Partially Committed**

  After completion of all the read and write operation the changes are made in main memory or local buffer. If the changes are made permanent on the Data Base then the state will change to “committed state” and in case of failure it will go to the “failed state”.

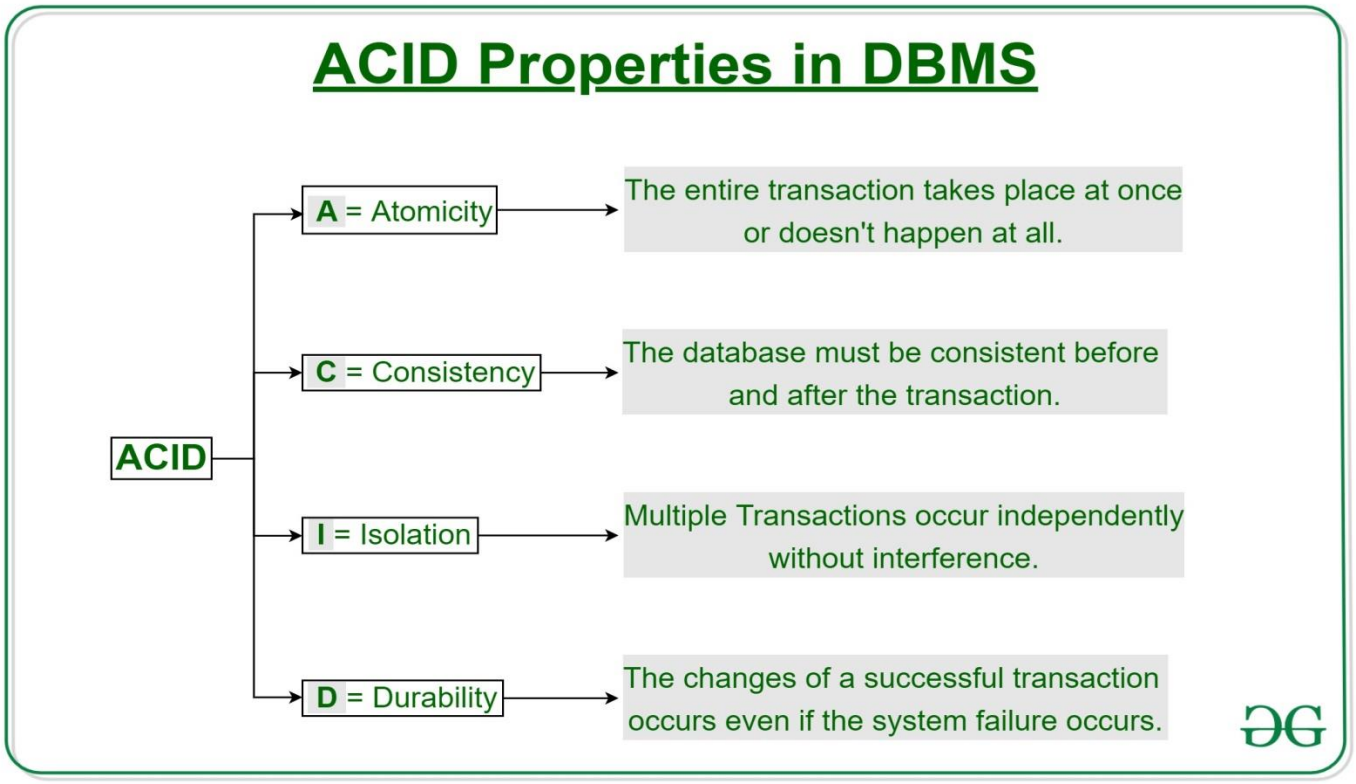
Transaction States in DBMS

- **Failed State**
  When any instruction of the transaction fails, it goes to the “failed state” or if failure occurs in making a permanent change of data on Data Base.
- **Aborted State**
  After having any type of failure the transaction goes from “failed state” to “aborted state” and since in previous states, the changes are only made to local buffer or main memory and hence these changes are deleted or rolled-back.
- **Committed State**
  It is the state when the changes are made permanent on the Data Base and the transaction is complete and therefore terminated in the “terminated state”.
- **Terminated State**
  If there isn’t any roll-back or the transaction comes from the “committed state”, then the system is consistent and ready for new transaction and the old transaction is terminated.

ACID Properties

- ✓ A transaction is a single logical unit of work that accesses and possibly modifies the contents of a database. Transactions access data using read and write operations.
- ✓ In order to maintain consistency in a database, before and after the transaction, certain properties are followed. These are called **ACID** properties.

ACID Properties in DBMS



ACID Properties in DBMS

1. Atomicity:

- ✓ Atomicity means that either the entire transaction takes place at once or doesn't happen at all i.e. transactions do not occur partially. Each transaction is considered as one unit and either runs to completion or is not executed at all. It involves the following two operations.
  Abort: If a transaction aborts, changes made to the database are not visible.
  Commit: If a transaction commits, changes made are visible.
  Atomicity is also known as the 'All or nothing rule'.
- ✓ Consider the following transaction **T** consisting of **T1** and **T2**: Transfer of 100 from account **X** to account **Y**.
- ✓

Before: X : 500	Y: 200
Transaction T	
T1	**T2**
Read (X) X: = X – 100 Write (X)	Read (Y) Y: = Y + 100 Write (Y)
After: X : 400	Y : 300

ACID Properties in DBMS

If the transaction fails after completion of **T1** but before completion of **T2**.(say, after **write(X)** but before **write(Y)**), then the amount has been deducted from **X** but not added to **Y**. This results in an inconsistent database state. Therefore, the transaction must be executed in its entirety in order to ensure the correctness of the database state.

2. Consistency :

This means that integrity constraints must be maintained so that the database is consistent before and after the transaction. It refers to the correctness of a database. Referring to the example above,
The total amount before and after the transaction must be maintained.
Total **before T** occurs = **500 + 200 = 700**.
Total **after T occurs** = **400 + 300 = 700**.
Therefore, the database is **consistent**. Inconsistency occurs in case **T1** completes but **T2** fails. As a result, T is incomplete.

ACID Properties in DBMS

3. Isolation:

This property ensures that multiple transactions can occur concurrently without leading to the inconsistency of the database state. Transactions occur independently without interference. Changes occurring in a particular transaction will not be visible to any other transaction until that particular change in that transaction is written to memory or has been committed. This property ensures that the execution of transactions concurrently will result in a state that is equivalent to a state achieved these were executed serially in some order.

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

Transaction Management

- ✓ It is used to solve Read/Write Conflict.
- ✓ It is used to implement **Recoverability**, **Serialaizability**, and **Cascading.**
- ✓ Transaction Management is also used for Concurrency Control Protocols

Concurrent execution

- ✓ The ability of the system to allow multiple transactions to execute simultaneously.
- ✓ In a multi-user environment, several users may be accessing and manipulating the database concurrently. Concurrent execution offers advantages such as increased system throughput and better resource utilization. However, it also introduces potential issues related to data consistency and integrity.
- ✓ To manage concurrent execution of transactions, DBMS typically employs **concurrency control** mechanisms.

Concurrency control

- ✓ A set of techniques and mechanisms used to manage and coordinate the simultaneous execution of multiple transactions accessing and modifying shared data in a database system.
- ✓ The goal of concurrency control is to ensure data consistency, maintain transaction isolation, and prevent data integrity issues when multiple transactions are executing concurrently.
- ✓ Concurrency control mechanisms handle potential conflicts and coordination issues that arise due to concurrent access to shared data. These mechanisms ensure that the transactions are executed in an ordered and controlled manner, preventing anomalies such as lost updates, dirty reads, non-repeatable reads, and inconsistent results

Concurrency problems

- ✓ Concurrency problems in databases arise when multiple transactions or processes attempt to access and modify the same data concurrently, leading to potential conflicts and inconsistencies.
- ✓ These problems can impact the correctness, integrity, and consistency of data in a database system

Concurrency Problems

- ✓ **Lost Updates:** Lost updates occur when two or more transactions concurrently update the same data, resulting in the loss of some updates. For example, if two transactions read and update the same data simultaneously, one transaction's changes may be overwritten by the other, leading to lost updates.
- ✓ **Dirty Reads:** Dirty reads occur when one transaction reads data that has been modified by another transaction but has not yet been committed. If the modifying transaction rolls back, the data read by the first transaction becomes invalid or incorrect.
- ✓ **Inconsistent Reads:** Inconsistent reads happen when a transaction reads data that is in the process of being modified by another transaction. If a transaction reads uncommitted or partially updated data, it can lead to incorrect results or inconsistent views of the data.
- ✓ **Non-Repeatable Reads:** Non-repeatable reads occur when a transaction reads the same data multiple times, but the values change between subsequent reads due to updates made by other concurrent transactions. This can lead to unexpected and inconsistent results.

Schedule

- ✓ A schedule refers to an ordered sequence of operations performed by a set of concurrent transactions.
- ✓ It represents the chronological order in which the operations of multiple transactions are executed

Example

- Consider two transactions:

```
T1:     BEGIN   A=A+100,   B=B-100   END
T2:     BEGIN   A=1.06*A,   B=1.06*B   END
```

- Intuitively, the first transaction is transferring Rs100 from B's account to A's account. The second is crediting both accounts with a 6% interest payment.
- There is no guarantee that T1 will execute before T2 or vice-versa, if both are submitted together. However, the net effect *must* be equivalent to these two transactions running serially in some order.

Example (Contd.)

- Consider a possible interleaving schedule:

T1:	A=A+100,		B=B-100
T2:		A=1.06*A,	B=1.06*B

- This is OK. But what about:

T1:	A=A+100,		B=B-100
T2:		A=1.06*A, B=1.06*B	

- The DBMS's view of the second schedule:

T1:	R(A), W(A),		R(B), W(B)
T2:		R(A), W(A), R(B), W(B)	

Anomalies with Interleaved Execution

- Reading Uncommitted Data (WR Conflicts, "dirty reads"):

```
T1:    R(A), W(A),                R(B), W(B), Abort
T2:                    R(A), W(A), C
```

- Unrepeatable Reads (RW Conflicts):

```
T1:    R(A),                      R(A), W(A), C
T2:           R(A), W(A), C
```

Anomalies (Continued)

- Overwriting Uncommitted Data (WW Conflicts):

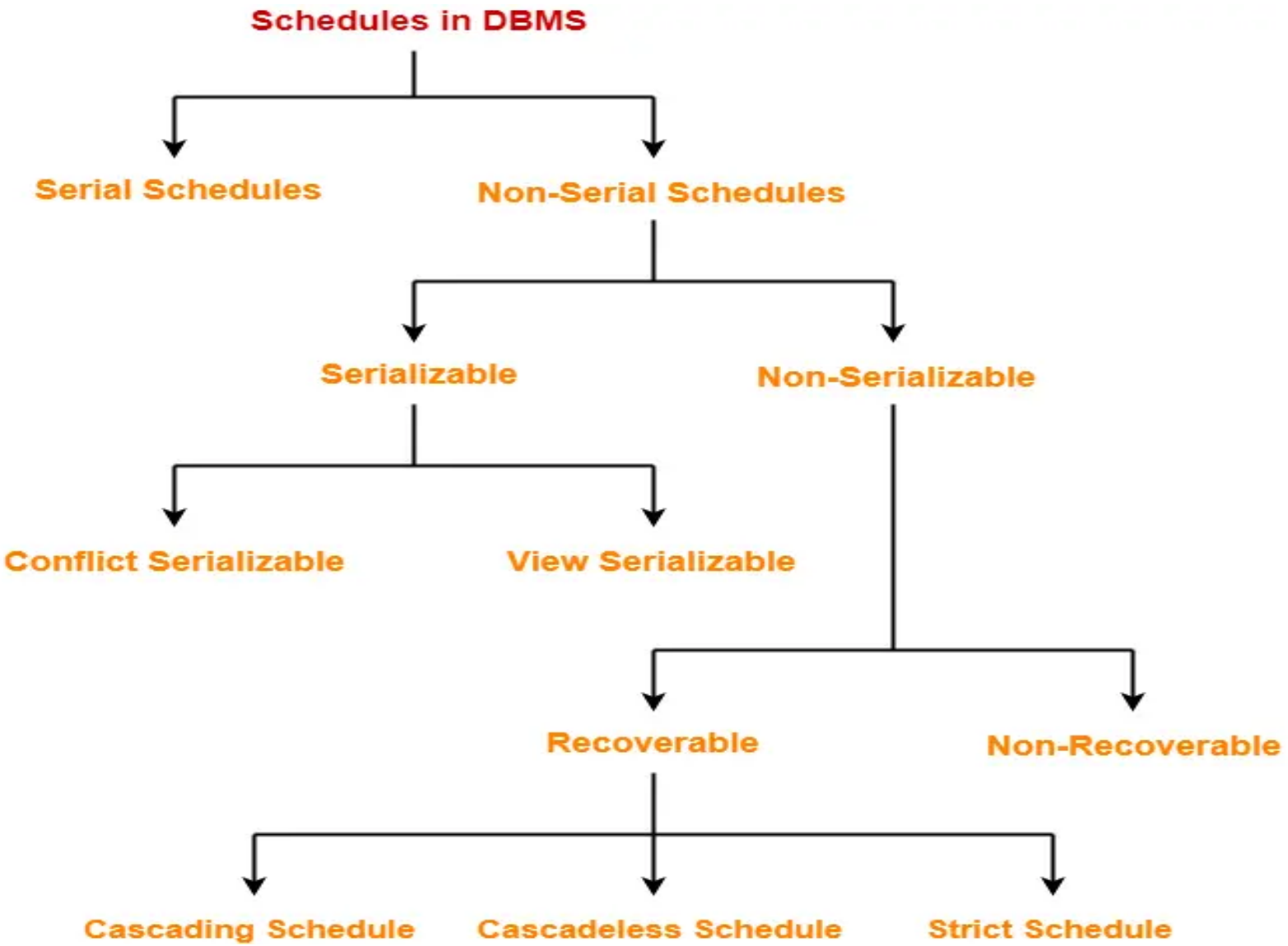
T1:	W(A),		W(B), C
T2:		W(A), W(B), C	

Types of Schedules

1. **Serial schedule**
2. **Non-Serial Schedule**

- ✓ Scheduling in DBMS is majorly classified into Serial and Non-Serial Schedules. A serial schedule is a schedule that does not interleave the actions of different transactions.
- ✓ Serial schedules are always consistent while non-serial schedules are not always consistent

Types of Schedules



A serial Schedule

Time

T1	T2

read(A);
A := A – N;
write(A);
read(B);
B := B + N;
write(B);
	read(A);
	A := A + M;
	write(A);

Schedule A

Serializable schedule

- ✓ If a given non-serial schedule of ‘n’ transactions is equivalent to some serial schedule of ‘n’ transactions, then it is called as a serializable schedule.
- ✓ The serializable schedule is always Consistent, Recoverable, Casacadeless and Strict.
- ✓ Serializability ensures that the database remains in a consistent state despite concurrent execution.

Serial vs Serializable Schedules

Serial Schedules	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.	Serializable Schedules are always better than serial schedules.

https://www.gatevidyalay.com/serializability-in-dbms-conflict-serializability/

Serializability in Database

- ✓ Serializability refers to the property of a schedule in which it is equivalent to a serial execution of transactions, meaning that the outcome of executing transactions concurrently is the same as executing them one after another in some order.
- ✓ Some non-serial schedules may lead to inconsistency of the database. Serializability is a concept that helps to identify which non-serial schedules are correct and will maintain the consistency of the database.
- ✓ Serializability ensures that the database remains in a consistent state despite concurrent execution of transactions

Serializability in DBMS

- ✓ Serializability is the concept in a transaction that helps to identify which non-serial schedule is correct and will maintain the database consistency.
- ✓ Serializability is the concurrency scheme where the execution of concurrent transactions is equivalent to the transactions which execute serially.
- ✓ If a schedule of concurrent ‘n' transactions can be converted into an equivalent serial schedule. Then we can say that the schedule is serializable. And this property is known as serializability.

1. Conflict Serializable Schedule

- ✓ If a given non-serial schedule can be converted into a serial schedule by swapping its non-conflicting operations, then it is called as a conflict serializable schedule.
- ✓ The two 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.
- ✓ Conflict pairs for the same data item are: Read-Write, Write-Write and Write-Read

Example of Conflict Serializability

Non-Serial Schedule

Time	Transaction T1	Transaction T2	Transaction 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)		

Example of Conflict Serializability (Convert into serial schedule)

Serial Schedule

Time	Transaction T1	Transaction T2	Transaction 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)		

How to Checking Conflict Serializablity?

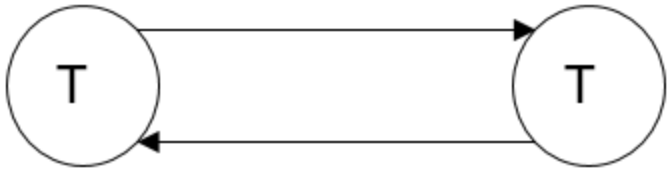
- ✓ To check for conflict serializability in a database, you can use the precedence graph (also known as the serialization graph) method.
- ✓ You should check if there is any cycle formed in the graph. If there is no cycle found, then the schedule is conflict serializable otherwise not.
- ✓ https://www.youtube.com/watch?v=8LKM_RWeroM, https://www.youtube.com/watch?v=zv0ba0Iok1Y

Algorithm .

- ✓ Determine the conflicting operations: Identify all the conflicting read and write operations between transactions.
- ✓ Construct the precedence graph: Create a node for each transaction in the graph. For each pair of conflicting operations (Ti and Tj), draw an arrow from Ti to Tj if the operation in Ti should precede the operation in Tj. There are two types of conflicts:
  - Read-Write Conflict (Ti reads a data item, and Tj writes to the same data item): Draw an arrow from Ti to Tj.
  - Write-Write Conflict (Both Ti and Tj write to the same data item): Draw an arrow from Ti to Tj or from Tj to Ti.
- ✓ Analyze the precedence graph: Check if the graph contains any cycles. If there are no cycles in the graph, then the transactions are conflict serializable. Otherwise, if there is a cycle, it indicates a conflict, and the transactions are not conflict serializable

Example to check conflict Serilizability

Time	Transaction p	Transaction q
1	read (A)	
2	A = A * 7	
3		read (A)
4		Temp = A + 45
5		A=A + temp
6		write (A)
7		read (B)
8	write (A)	
9	read (B)	
10	B = B - 32	
11	write (B)	
12		B = B + temp
13		write (B)

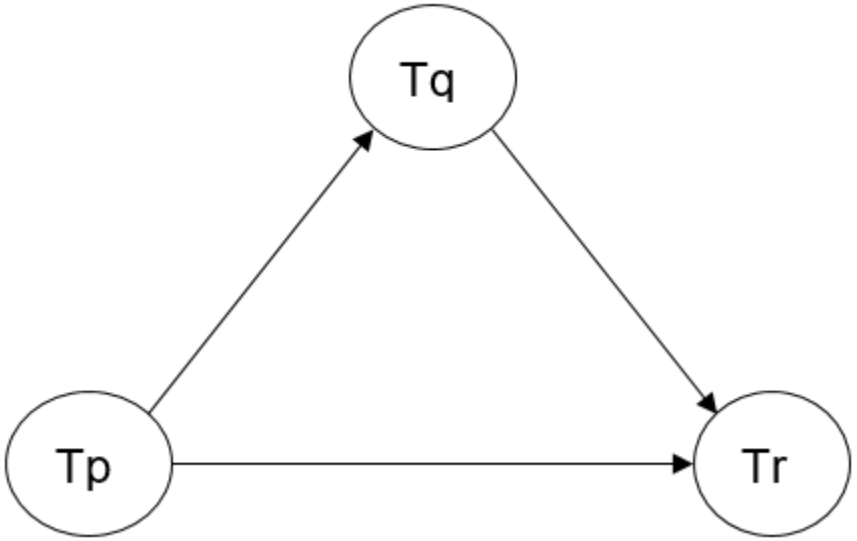


Transaction Tp implements reads A before Transaction Tq implements writes A, therefore the first arrow directed from Transaction Tp towards Transaction Tq and Transaction Tq reads B before Transaction Tp writes B, therefore the second arrow directed from Transaction Tq towards Transaction Tp. Thus, graph is Cyclic and schedule is not conflict serilizable.

https://www.includehelp.com/dbms/precedence-graph.aspx

Example

Time	Transaction p	Transaction q	Transaction r
1	read (A)		
2	A = A + 7		
3	read (C)		
4	write (A)		
5	A = C - 32		
6		read (B)	
7	write (C)		
8		read (A)	
9			read (C)
10		B = B + 34	
11		write (B)	
12			C = C * 2
13			read (B)
14			write (C)
15		A = A - 21	
16		write (A)	
17			B = B - 8
18			write (B)



- ✓ In graph, Transaction **Tp** implements reads **A** before Transaction **Tq** implements writes **A**, therefore the first arrow directed from Transaction **Tp** towards Transaction **Tq**
- ✓ Transaction **Tq** reads **B** before Transaction **Tr** writes **B**, therefore the second arrow directed from Transaction **Tq** towards Transaction **Tr** and
- ✓ The Transaction **Tp** reads **C** before Transaction **Tr** writes **C**, therefore the third arrow directed from Transaction **Tp** towards **Tr**.
- ✓ It is clearly visible that the graph is acyclic, **therefore the schedule S is conflict Serializable.**
- ✓ *Additional Examples: https://www.gatevidyalay.com/conflict-serializability-practice-problems/*

2. View Serializability

- ✓ If a given schedule is found to be view equivalent to some serial schedule, then it is called as a view serializable schedule.
- ✓ **View Equivalent Schedules**

Consider two schedules S1 and S2 each consisting of two transactions T1 and T2. Schedules S1 and S2 are called view equivalent if the following three conditions hold true for them:

1. **Initial Read Rule:** For each data item X, if transaction T_i reads X from the database initially in schedule S1, then in schedule S2 also, T_i must perform the initial read of X from the database.
2. **Update Read Rule:** If transaction Ti reads a data item that has been updated by the transaction Tj in schedule S1, then in schedule S2 also, transaction Ti must read the same data item that has been updated by the transaction Tj.
3. **Final Write Rule**: For each data item X, if X has been updated at last by transaction Ti in schedule S1, then in schedule S2 also, X must be updated at last by transaction Ti.

View Equivalent Schedules- Condition 1

Schedule S1

Time	Transaction T1	Transaction T2
t1	Read(A)	
t2		Write(A)

Schedule S2

Time	Transaction T1	Transaction T2
t1		Write(A)
t2	Read(A)	

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 T1.

View Equivalent Schedules- Condition 2

Updated Read

In schedule S1, if the transaction T_2 is reading the data item A which is updated by transaction T_1, then in schedule S2 also, T_2 should read data item A which is updated by T_1.

Schedule S1

Time	Transaction T1	Transaction T2
t1	Write(A)	
t2		Read(A)

Schedule S2

Time	Transaction T1	Transaction T2
t1	Write(A)	
t2		Read(A)

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.

View Equivalent Schedules- Condition 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 (A)
t2		Read(A)	
t3	Write(A)		

Schedule S2

time	T1	T2	T3
t1			Write (A)
t2		Read(A)	
t3	Write(A)		

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.

View Serializable Schedules Example

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

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

Check whether these schedules are view Serializable or not?

Additional Examples: https://www.gatevidyalay.com/view-serializability-in-dbms-practice-problems/

Example

T1	T2	T3
R(A)		
	R(B)	
W(B)		
		R(C)
	W(C)	
		W(A)

T1	T2	T3
R(A)		
W(B)		
	R(B)	
	W(C)	
		R(C)
		W(A)

Summary

Thus, a schedule is view serializable if it is equivalent to a serial schedule based on the view it produces (i.e., the final result of all transactions and the intermediate read operations).

Summary: View Serializibility

- ✓ **All conflict serializable schedules are view serializable:** If a schedule is conflict serializable, it implies that the schedule can be reordered into a serial schedule without changing the outcome of the transactions. This means that all initial reads, update reads, and final writes align with the corresponding serial schedule. Therefore, a conflict serializable schedule inherently satisfies the conditions of view serializability because it maintains the same data access order as some serial execution.
- ✓ **All view serializable schedules may or may not be conflict serializable:** View serializability is a broader concept than conflict serializability. A schedule can be view serializable without being conflict serializable because view serializability allows for more flexibility in operation order as long as the "view" (result) remains the same

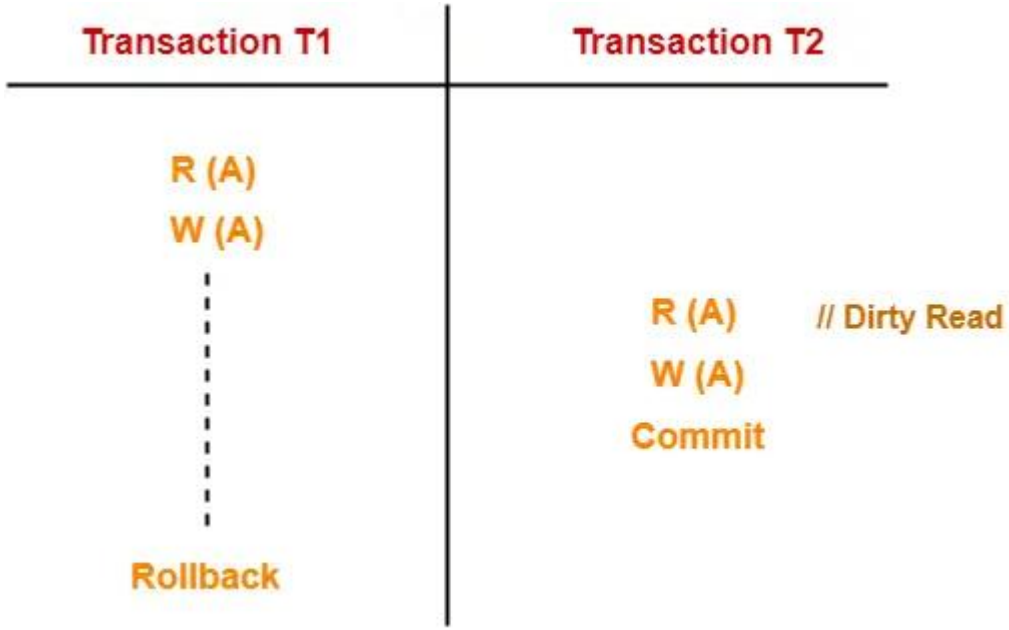
Recoverability

- ✓ A non-serial schedule which is not serializable is called as a non-serializable schedule. A non-serializable schedule is not guaranteed to produce the same effect as produced by some serial schedule on any consistent database.
- ✓ Non-serializable schedules- may or may not be consistent and may or may not be recoverable.
- ✓ Recoverability refers to the ability to restore the database to a consistent and correct state after a transaction failure or system crash.
- ✓ It ensures that changes made by committed transactions are durably stored, and the system can recover from failures without losing committed data.

https://www.youtube.com/watch?v=g2gZKA8E1yA

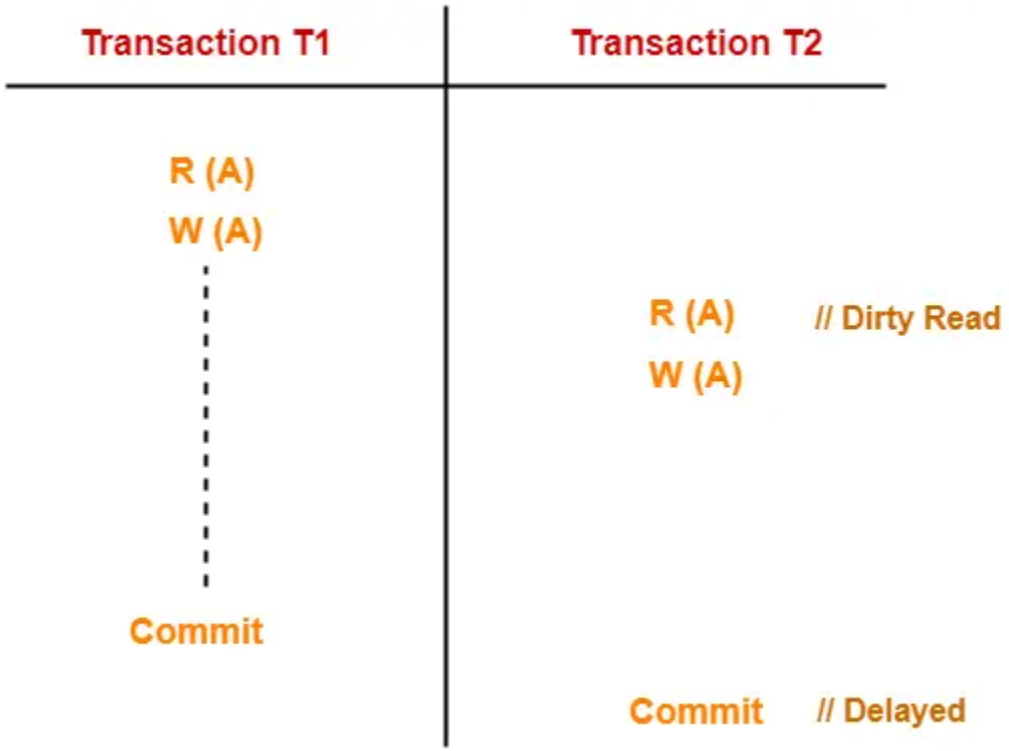
Irrecoverable Schedule

- ✓ If in a schedule, a transaction performs a dirty read operation from an uncommitted transaction and commits before the transaction from which it has read the value then such a schedule is known as an **Irrecoverable Schedule**



Irrecoverable Schedule

- ✓ If in a schedule, a transaction performs a dirty read operation from an uncommitted transaction and its commit operation is delayed till the uncommitted transaction either commits or roll backs then such a schedule is known as a **Recoverable Schedule**



Recoverable Schedule

Approaches to achieving serializability (Concurrency Control Protocols)

- ✓ **Locking-based Concurrency Control:** Locking mechanisms, such as Binary Locking, two-phase locking (2PL), are used to control concurrent access to data items. Transactions acquire locks before accessing or modifying data, ensuring that conflicting operations do not occur simultaneously. This mechanism ensures serializability by enforcing a strict ordering of operations.
- ✓ **Timestamp-based Concurrency Control:** Each transaction is assigned a unique timestamp that represents its start time. Timestamps are used to order and coordinate the execution of transactions, ensuring that conflicting operations are correctly serialized based on their timestamps

Lock-Based Protocols

- ✓ Lock is a mechanism which is important in a concurrent control. It controls concurrent access to a data item. It assures that one process should not retrieve or update a record which another process is updating.
- ✓ There are two lock modes: Shared Lock (S) and Exclusive Lock (X).
- **A shared Lock** is also called a Read-only lock. With the shared lock, the data item can be shared between transactions. 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.
- 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. 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

Compatibility Table for Lock

		Request	
		S	X
Granted	S	Yes	No
	X	No	No

- When a shared lock (S) has already been granted on an item, a subsequent request for another shared lock on the same item is also granted. However, if a request is made for an exclusive lock (X) on an item that already has a shared lock, the request is denied.
- Similarly, if an exclusive lock is already held on an item, neither a shared lock nor another exclusive lock can be granted until the exclusive lock is released

Shared and Exclusive Locks

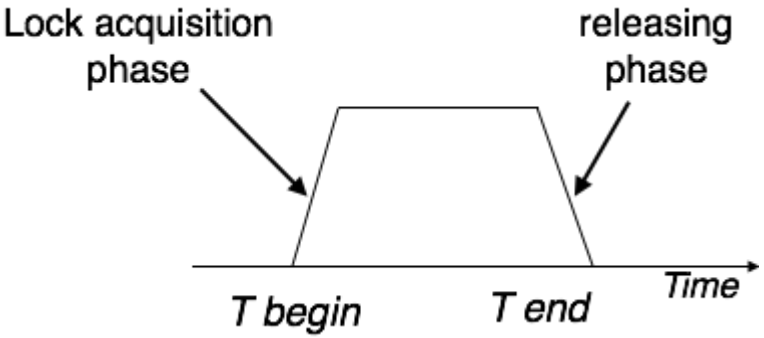
- **Shared (S) vs. Shared (S):** Compatible. Multiple transactions can hold a shared lock on the same item simultaneously, allowing them to read the item concurrently.
- **Shared (S) vs. Exclusive (X):** Incompatible. If a shared lock is requested on an item already locked with an exclusive lock, the request is denied until the exclusive lock is released.
- **Exclusive (X) vs. Shared (S):** Incompatible. An exclusive lock cannot be granted if there is an existing shared lock. The transaction must wait until all shared locks are released.
- **Exclusive (X) vs. Exclusive (X):** Incompatible. Only one transaction can hold an exclusive lock on an item at a time, preventing any other transaction from acquiring a lock on the item

Two phase Locking (2PL) protocol

- ✓ Two Phase Locking Protocol (2PL) 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.
- ✓ Phases:

1. In the first phase, when the transaction begins to execute, it requires permission for the locks it needs.
2. The second part is where the transaction obtains all the locks. When a transaction releases its first lock, the transaction cannot demand any new locks. Instead, it only releases the acquired locks.

Database Concurrency Control



Two-Phase Locking Techniques: The algorithm

- Two Phases:
  - (a) Locking (Growing)
  - (b) Unlocking (Shrinking).
- **Locking (Growing) Phase:**
  - A transaction applies locks (read or write) on desired data items one at a time.
- **Unlocking (Shrinking) Phase:**
  - A transaction unlocks its locked data items one at a time.
- **Requirement:**
  - For a transaction these two phases must be mutually exclusively, that is, during locking phase unlocking phase must not start and during unlocking phase locking phase must not begin.

Example

	A=10	B=20	
Step	**T1**	**T2**	**Description**
1	Read_Lock(A)		T1 acquires a read lock on A.
2	Read(A)		T1 reads A (A = 10).
3	Write_Lock(A)		T1 upgrades to a write lock on A.
4	A := A + 10		T1 updates A to 20.
5	Write(A)		T1 writes the updated value of A.
6	Read_Lock(B)		T1 acquires a read lock on B.
7		Read_Lock(B)	T2 acquires a read lock on B.
8	Read(B)		T1 reads B (B = 20).
9		Read(B)	T2 reads B (B = 20).
10	Write_Lock(B)		T1 upgrades to a write lock on B.
11	B := A + B		T1 updates B to 40.

12	Write(B)		T1 writes the updated value of B.
13	Unlock(A)		T1 releases the lock on A.
14	Unlock(B)		T1 releases the lock on B.
15		Write_Lock(B)	T2 upgrades to a write lock on B.
16		B := B - 5	T2 updates B to 35.
17		Write(B)	T2 writes the updated value of B.
18		Read_Lock(A)	T2 acquires a read lock on A.
19		Read(A)	T2 reads A (A = 20).
20		Write_Lock(A)	T2 upgrades to a write lock on A.
21		A := A + B	T2 updates A to 55.
22		Write(A)	T2 writes the updated value of A.
23		Unlock(B)	T2 releases the lock on B.
24		Unlock(A)	T2 releases the lock on A.
25		Unlock(A)	T2 releases the lock on A.

Time Stamp Ordering Protocol

- ✓ In the Timestamp Ordering Protocol, each transaction is assigned a unique timestamp when it enters the system. The timestamp represents the order in which the transactions are executed or scheduled.
- ✓ The older transaction is executed first due to high priority. The protocol uses time of system or logical counter to determine transaction timestamp.
- ✓ The protocol works based on two concepts: Read timestamp (RTS(T)) and Write timestamp (WTS(T)). The read timestamp of a transaction is the timestamp at which it accessed a particular data item for reading. The write timestamp of a transaction is the timestamp at which it modified a particular data item.
- ✓ RTS (A)- Last transaction no./timestamp which performed Read operation successfully on A
- ✓ WTS(A)- Last transaction no./timestamp which performed write operation successfully on A
- ✓ TS(Ti)- Timestamp of Transaction Ti

Ordering Rule

- ✓ **Read Rule:** A transaction can read a data item only if the last write to that item was done by an older transaction.
- ✓ **Write Rule:** A transaction can write to a data item only if no younger transaction has already read or written that data item.

Example

Let us consider a single data item X with initial value 100 and two transactions T1 and T2.

Transaction T1: Starts at time **TS(T1) = 5 (Older Transaction)**

Transaction T2: Starts at time **TS(T2) = 10 (Younger Transaction)**

Suppose, the initial read and write timestamps of X are **RTS(X) = 0** and **WTS(X) = 0**.

1. **T1 reads X:**

   Since TS(T1) = 5 is greater than WTS(X) = 0 (i.e. last write operation by older transaction), T1 is allowed to read X. Thus, update RTS(X) to 5 (the timestamp of T1) i.e. **RTS(X) =5**

   Here, T1 reads X = 100.

2. **T2 writes X:**

   Since TS(T2) = 10 is greater than both **RTS(X) = 5** and **WTS(X) = 0 (i.e, older transactions has last read and write operations)**, T2 is allowed to write to X. Thus, update WTS(X) to 10 (the timestamp of T2) i.e., **WTS(X)=10**

   Suppose T2 writes X = 200.

3. **T1 tries to write X:**

T1 wants to write to X, but since TS(T1) = 5 is less than WTS(X) = 10 (set by T2, i.e. younger transaction), this would violate the TSO protocol. Thus, T1 is aborted because it is attempting to write based on an outdated view of X. This ensures that the system maintains a serializable order of transactions based on their timestamps.

Rules

1) Transaction T_i issues a Read(A) operation
   a) if $WTS(A) > TS(T_i)$, Rollback T_i
   b) Otherwise execute R(A) operation
      Set $RTS(A) = Max\{RTS(A), TS(T_i)\}$
2) Transaction T_i issues Write(A) operation
   a) if $RTS(A) > TS(T_i)$ then Rollback T_i
   b) if $WTS(A) > TS(T_i)$ then Rollback T_i
   c) otherwise execute Write(A) operation
      Set $WTS(A) = TS(T_i)$

Timestamp Ordering Protocol

- ✓ The Timestamp Ordering Protocol ensures serializability by maintaining the order of transactions based on their timestamps. If the protocol is followed strictly, it guarantees that the execution of transactions will be equivalent to a serial execution of the transactions in order of their timestamps.
- ✓ A timestamp is a monotonically increasing variable (integer) indicating the age of an operation or a transaction. A larger timestamp value indicates a more recent event or operation. Timestamp based algorithm uses timestamp to serialize the execution of concurrent transactions.

Deadlock

- ✓ A deadlock is a situation where two or more transactions are waiting indefinitely for each other to release resources, resulting in a state of mutual blocking or circular dependency.
- ✓ Deadlocks prevent the involved transactions from progressing and can cause a system to become unresponsive.
- ✓ There are several common techniques for handling deadlocks in DBMS: Deadlock Detection, Deadlock Avoidance, Deadlock Prevention and Deadlock Recovery and so on.

Dealing with Deadlock and Starvation

- **Deadlock**

T'1	**T'2**
read_lock (Y);	
read_item (Y);	
	read_lock (X);
	read_item (Y);
write_lock (X);	
(waits for X)	write_lock (Y);
	(waits for Y)

T1 and T2 did follow two-phase policy but they are deadlock

- Deadlock (T'1 and T'2)

Reasons for the deadlock

- ✓ **Mutual Exclusion:** Each resource can be accessed by only one transaction at a time. If a transaction holds a resource, other transactions are prevented from accessing it.
- ✓ **Hold and Wait:** A transaction holds at least one resource and waits to acquire additional resources that are held by other transactions.
- ✓ **No Preemption:** Resources cannot be forcibly taken away from a transaction; they can only be released voluntarily by the transaction holding them.
- ✓ **Circular Wait:** A circular chain of transactions exists, where each transaction is waiting for a resource that is held by another transaction in the chain. This forms a deadlock cycle.

Deadlock prevention

- A transaction locks all data items it refers to before it begins execution.
- This way of locking prevents deadlock since a transaction never waits for a data item.
- The conservative two-phase locking uses this approach.
- By carefully managing resource allocation and transaction scheduling, the occurrence of deadlocks can be prevented. However, prevention methods may result in decreased concurrency and resource utilization

Deadlock detection and resolution

- In this approach, deadlocks are allowed to happen. The scheduler maintains a wait-for-graph for detecting cycle. If a cycle exists, then one transaction involved in the cycle is selected (victim) and rolled-back. (also Resource allocation graph can be used)
- A wait-for-graph is created using the lock table. As soon as a transaction is blocked, it is added to the graph. When a chain like: Ti waits for Tj waits for Tk waits for Ti or Tj occurs, then this creates a cycle.

Deadlock avoidance

- ✓ Avoidance involves dynamically analyzing the resource allocation requests of transactions to determine if granting them would lead to a potential deadlock.
- ✓ By employing resource allocation strategies and algorithms, the system can make decisions to avoid granting resource requests that may result in deadlocks.
- ✓ This technique requires additional information and sophisticated algorithms to make intelligent decisions

Deadlock Recovery

- ✓ In this approach, the system allows deadlocks to occur but includes mechanisms for recovering from them.
- ✓ When a deadlock is detected, the system can break the deadlock by aborting one or more transactions involved in the deadlock, releasing their held resources and allowing the remaining transactions to proceed.
- ✓ The aborted transactions can then be restarted or rolled back to a safe state

Recovery Model

- ✓ Recovery in a database refers to the process of restoring the database to a consistent and usable state after a failure or an unexpected event that could potentially compromise data integrity. Failures can include hardware or software errors, system crashes, power outages, and other types of disruptions.
- ✓ The main goal of recovery techniques is to ensure data integrity and consistency and prevent data loss.
- ✓ There are two common approaches used in database systems to achieve data recovery and maintain data integrity in the event of failures. They are:

1. Log-based recovery (Rollback/Undo Recovery Technique - Immediate Data Modification, Commit/Redo Recovery Technique - Deferred Data Modification)
2. Shadow-based recovery

1. Log-based Recovery

Log-based recovery, also known as redo/undo recovery, is a widely used technique that relies on transaction logs to restore the database to a consistent state. It involves two main operations: redo and undo.

- **Redo (Commit Recovery Technique/ Deferred Data Modification)**: During the redo phase, the system applies the logged changes from the transaction log to the database to bring it up to the latest committed state. This ensures that any committed changes that were not yet durably written to the database are correctly reapplied.
- **Undo (Rollback Recovery Technique/Immediate Data Modification Technique)** : During the undo phase, the system identifies any incomplete or uncommitted transactions that were active at the time of failure. The changes made by these transactions are then rolled back or undone to restore the database to a consistent state.

The transaction log plays a crucial role in log-based recovery. It records all modifications made by transactions, including both before and after values. By analyzing the log, the DBMS can determine which changes need to be redone or undone to achieve recovery.

Deferred Data Modification: In a deferred data modification approach, the modifications made by transactions are not immediately applied to the database. Instead, they are temporarily stored in a buffer or log (**transaction log**) during transaction execution. These modifications are written to the database after commit statement.

Immediate Data Modification: In an immediate data modification approach, modifications made by transactions are immediately applied to the database. Each modification is written to the database as soon as the transaction executes the corresponding operation, such as an insert, update, or delete. If there is any problem in the transaction, it will rollback to the initial state.

2. Shadow-based Recovery

- ✓ Shadow-based recovery, also known as backward error recovery or backward error propagation, is an approach to recovery that utilizes shadow copies or shadow pages.
- ✓ In this technique, a separate copy of the database or specific pages is maintained, known as the shadow copy. The shadow copy is an exact replica of the original database or specific pages. It is created and updated concurrently with the primary database. In the event of a failure, the shadow copy is used to restore the database to a consistent state. The system switches from the primary database to the shadow copy, effectively replacing the damaged or inconsistent data.
- ✓ Shadow-based recovery is less commonly used compared to log-based recovery, primarily due to the additional storage requirements for maintaining the shadow copy and the complexity of managing concurrent updates to both the primary and shadow copies.

Thank you.