

# Serializability

**Serializability:** It ensures that concurrent transactions yield results that are consistent with some serial execution i.e the final state of the database after executing a set of transactions concurrently should be the same as if the transactions had been executed one after another in some order.

In case of concurrent schedule consistency issue may arise because of non-serial execution and we do serializability there, serial schedules are already serial

**\*\* Serializability only tells if schedule is serializable or not. It does not serialize it.**

- **There are two types of serializability:**

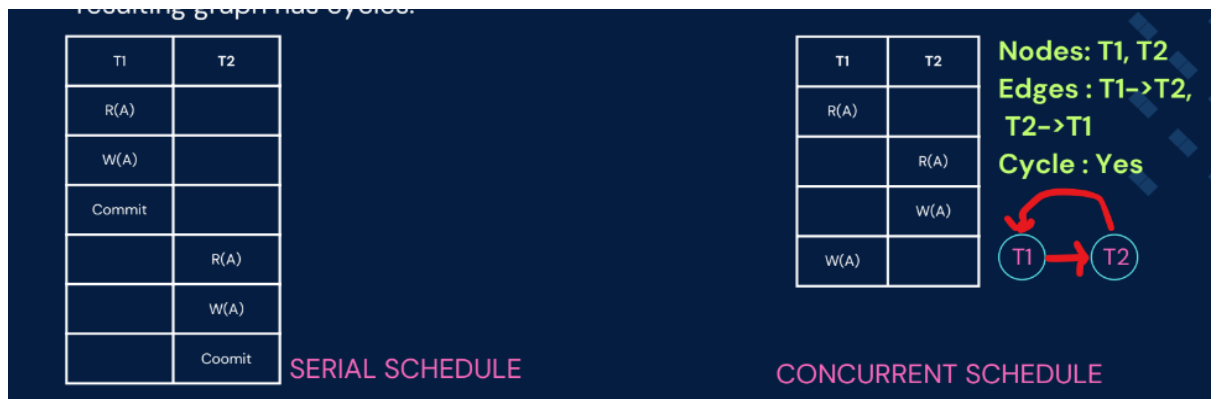
1. Conflict Serializability
2. View Serializability

- **To check if a concurrent schedule is serializable or not:** First check conflict serializability by checking if it has a cycle in conflict-graph or not, if no, then **conflict-serializable**, if yes then check for view serializability by checking its view equivalent follows some condition or not, if follows then **view-serializable**, otherwise not serializable.

- **Conflict Cycle**

Transactions are represented by Nodes and Conflicts are represented by Edges.

- Serial Schedule does not have any conflict cycle as there is no conflict so, no edges are there.
- Concurrent Schedule may or may not have a cycle in conflict graph.



## Conflict Serializability

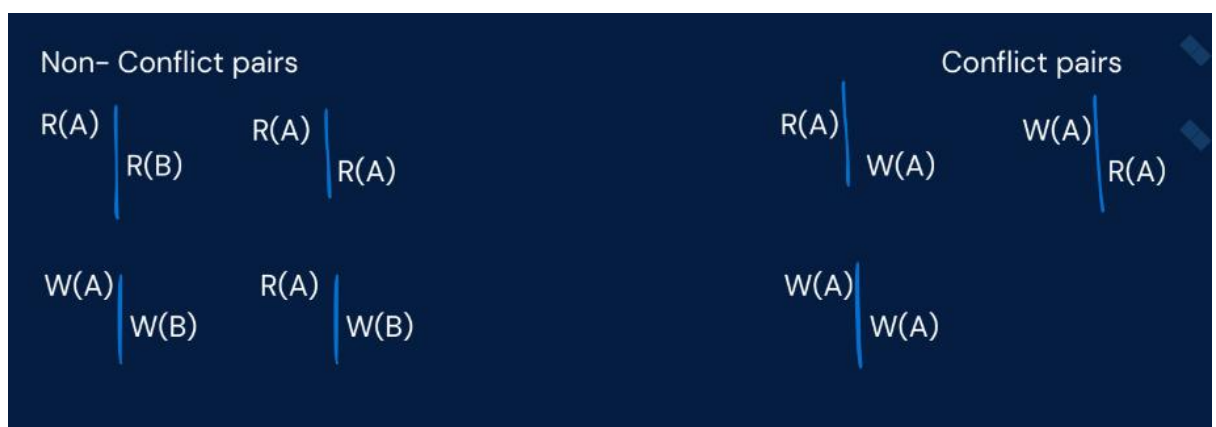
If the schedule can be rearranged (without violating any dependencies) to form a serial schedule, then it is **conflict serializable**.

There are two methods to find if a schedule is conflict-serializable:

1. **Conflict Equivalent:** If a schedule S1 is formed after swapping adjacent non-conflicting operations/pairs in a given schedule S, then S1 and S are conflict equivalent. If the conflict equivalent S1 has no conflicting adjacent pairs after swapping all adjacent non-conflicting operations/pairs, then its Serializable.

Conflict pairs: Read-Write, Write-Read, Write-Write (performed on the same data item)

Non-conflict pairs: Read-Read (performed on the same data item), Read-Read (performed on the different data item), Write-Write (performed on the different data item)



Now, why are we only swapping the non-conflict pairs and not the conflict ones?

So if we swap the conflict pairs, the order of execution if it was

T1: R(A)

T2: W(A)

the results values may change as first we were reading A and then writing/modifying it, but now it will be writing A and then reading the modified value so the result might change if we change the order of execution.

✓ Q. Find a conflict equivalent for a schedule S1

S1 :

T1	T2
R(X) ✓	
W(X) ✓	
	R(X)
R(Y) ✓	
W(Y) ✓	
	R(Y)

✓ 1. Find the adjacent non-conflicting pairs and do a swap

T2 : R(X) T1: R(Y)

T1	T2
R(X) ✓	
W(X) ✓	
R(Y) ✓	
	R(X)
W(Y) ✓	
	R(Y)

(do not swap already swapped pair, pair should have adjacent operations in the timeline)

2. After the first swap again search for adjacent non-conflicting pairs and swap if any  
 $T2 : R(X)$   $T1: W(Y)$

So,  $S1$  is a serializable schedule as  $S1'$  has a serial execution (i.e its conflict equivalent)

T1	T2
R(X) ✓	
W(X) ✓	
R(Y) ✓	
W(Y) ←	
	→ R(X) ✓
	R(Y) ✓

As, there is no more conflicting pairs (adjacent operations), It is serializable.

We do not use this method as frequent as for long transactions, it will take time.

2. **Conflict graph/precedence graph:** A conflict graph, or precedence graph, is a directed graph used to determine conflict serializability. The nodes represent transactions, and the edges represent conflicts between transactions.

#### Conflict graph/ precedence graph:

- **Nodes:** Each transaction in the schedule is represented as a node in the graph.
- **Edges:** An edge from transaction  $T(X)$  to transaction  $T(Y)$  (denoted  $T(X) \rightarrow T(Y)$ ) is added if
  - a. an operation of  $T(X)$  conflicts with an operation of  $T(Y)$  and
  - b.  $T(X)$  operation precedes  $T(Y)$  operation in the schedule.
- **Cycle Detection:** The schedule is conflict-serializable if and only if the conflict graph is acyclic. If there are no cycles in the graph, it means that the schedule can be serialized without violating the order of conflicting operations.

**Q. Check if the given schedule is conflict-serializable or not?**

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

- T1 reads A
- T2 reads A
- T1 writes A
- T3 writes A
- T2 writes B
- T3 reads B

**LET'S START WITH DBMS :)**

Conflict-Serializability

**Q. Check if the given schedule is conflict-serializable or not?**

**Step 1: Find the conflict in the schedule**

- RW Conflict (T1, T3) on A
- RW Conflict (T2, T3) on A
- RW Conflict (T2, T1) on A
- WW Conflict (T1, T3) on A
- WR Conflict (T2, T3) on B

**Step 2: Find the nodes**

Each transaction T1 T2, T3 is a node in the graph

**Step 3: Find the edges/conflicts**

- T1→T3: Because T1 reads A before T3 writes A.
- T2→T3: Because T2 reads A before T3 writes A
- T1→T3: Because T1 writes A before T3 writes A.
- T2→T3: Because T2 writes B before T3 reads B.
- T2→T1: Because T2 reads A before T1 writes A.

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

Write-Write conflict

W(B) - W(B)

W(A) - W(A)

Write-Read conflict

W(B) - R(B)

W(A) - R(A)

Read-write conflict

R(B) - W(B)

R(A) - W(A)

For T1, check T2 and T3. For T2 check T1 and T3 and so on for T3. For R, check W (read-write conflict), For W check W (write-write conflict) and R (write-read conflict) (on same data). ie. For R(A) in T1, check if there is any W(A) (read-write conflict) in T2 and T3 (may be non-adjacent)

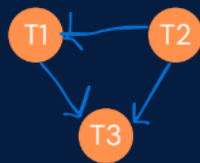


## LET'S START WITH DBMS :)

### Conflict-Serializability

Q. Check if the given schedule is conflict-serializable or not?

Step 4 : Draw the graph



edges

- $T2 \rightarrow T1$
- $T1 \rightarrow T3$
- $T2 \rightarrow T3$

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

Step 5 : If the graph has cycle it is not conflict-serializable or serializable, if not lets find the serial execution of transactions

Since the conflict graph is acyclic, the schedule is conflict-serializable

## LET'S START WITH DBMS :)

### Conflict-Serializability

Q. Check if the given schedule is conflict-serializable or not?

Step 6 : Lets find the serial execution of transactions

Possible combinations are :

- $T1 \rightarrow T2 \rightarrow T3$
- $T1 \rightarrow T3 \rightarrow T2$
- $T2 \rightarrow T1 \rightarrow T3$
- $T2 \rightarrow T3 \rightarrow T1$
- $T3 \rightarrow T2 \rightarrow T1$
- $T3 \rightarrow T1 \rightarrow T2$

Find the indegree (the number of edges directed into that node) and if its 0 it can be the first in serial execution

$T1 - 1, T2 - 0, T3 - 2$ , T2 would be the first as indegree is 0

- T2 must precede T1

- T1 must precede T3

Therefore, one possible equivalent serial schedule is  $T2 \rightarrow T1 \rightarrow T3$ .

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

edges

- $T2 \rightarrow T1$
- $T1 \rightarrow T3$
- $T2 \rightarrow T3$

Find the indegree (the number of edges directed into that node) and if its 0 it can be the first in serial execution

$T1 - 1, T2 - 0, T3 - 2$ , T2 would be the first as indegree is 0

- T2 must precede T1
- T1 must precede T3

Therefore, one possible equivalent serial schedule is  $T2 \rightarrow T1 \rightarrow T3$ .

As T2 is removed from cycle (as it had 0 indegree means no conflict),  $\text{indegree}(T1) = 0$  and  $\text{indegree}(T2) = 1$ .

Here the drawn timeline is the conflict equivalent of the schedule.

## View Serializability

### LET'S START WITH DBMS :)

#### View-Serializability

View serializability ensures that the database state seen by transactions in a concurrent schedule can be replicated by some serial execution of those transactions.

When we find cycle in our conflict graph, we don't know if our schedule is serializable or not, so we use the view serializability here.

A schedule is view serializable if it's view equivalent is equal to a serial schedule/execution.

T1	T2
R(A)	
	R(A)
W(A)	
	W(A)

Conditions for a schedule to be view-equivalent:

**1. Initial read:** (same for all data like A)

If for data A, if T1 reads A at first (before any other transaction), then in the view equivalent, only T1 should read A at first.

	T1	T2	T3	
S	R(A)			S'
		W(A)		
			W(B)	

## 2. Updated Read: (so that no consistency issue)


**2. Updated Read:**

- In schedule S, if  $T_i$  is reading B which is updated by  $T_j$  then in  $S'$  also,  $T_i$  should read B which is updated by  $T_j$ .

*Handwritten note:  $T_2 W(B) \rightarrow T_3 R(B)$*

	T1	T2	T3	
S		W(B)		S'
			R(B)	
	R(A)			

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



## 3. Final Update: (same for all data like A)

**3. Final Update**

- A final write must be identical in both schedules. If, in schedule S, transaction  $T_i$  is the last to update A, then in schedule  $S'$ , the final write operation on A should also be performed by  $T_i$

	T1	T2	T3	
S	W(A)			S'
		R(A)		
			W(A)	

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

All three conditions should be fulfilled for the schedule with its view equivalent to be serializable.



The number of possible serial schedules for  $n$  transactions is given by the number of permutations of the transactions:  $n!$

Q. Find the view equivalent schedule

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

← S

LET'S START WITH DBMS :)

$$n! = 3!$$

View-Serializability

Step 1: Find if conflict serializable or not.

Step 2: Find the possible serial schedules → 3!

Step 3: Choose one possibility and check for view equivalent conditions (T1→T2→T3)

1. Initial Read → T1

2. Update Read → No read performed after update so no need to check



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

R-W  
W-R  
W-W } same

As there is a cycle in the conflict graph in the image, it is not conflict-serializable.

T1 → T2 → T3

WITH DBMS :)

View-Serializability

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

serializable or not.

le serial schedules → 3!

As we have picked T1→T2→T3, we put all operation of T1, then put of T2 and then T3. And This will form S' view equivalent. If S' satisfies all condition then, its serializable.

Only one data: A

Initial read on A is T1 in both S and S'

Final read on A is T3 in both S and S'

# LET'S START WITH DBMS :)

$T_1 \rightarrow T_2 \rightarrow T_3$

## View-Serializability

**Step 1:** Find if conflict serializable or not.

**Step 2:** Find the possible serial schedules  $\rightarrow 3!$

**Step 3:** Choose one possibility and check for view equivalent conditions ( $T_1 \rightarrow T_2 \rightarrow T_3$ )

✓ 1. Initial Read  $\rightarrow T_1$

✓ 2. Update Read  $\rightarrow$  No read performed after update so no need to check

✓ 3. Final Update  $\rightarrow T_3$

**Step 4:** If the view equivalent schedule matches the condition, it is view serializable.



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

$T_1 \rightarrow S$

$T_1 \rightarrow S'$

$T_3 \rightarrow S$

R-w  
W-R  
W-W } same



$T_1 \rightarrow T_2 \rightarrow T_3$   
 $T_1 \rightarrow T_3 \rightarrow T_2$

$T_2 \rightarrow T_3 \rightarrow T_1$   
 $T_2 \rightarrow T_1 \rightarrow T_3$