Serializability

Serializability is the concept in a transaction that helps to identify which non-serial schedule is correct and will maintain the database consistency. It relates to the isolation property of transaction in the database. Serializability is the concurrency scheme where the execution of concurrent transactions is equivalent to the transactions which execute serially.

Serializable Schedule

A serial schedule is always a serializable schedule because any transaction only starts its execution when another transaction has already completed its execution. However, a non-serial schedule of transactions needs to be checked for Serializability.

Note: 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.

Testing of Serializability

To test the serializability of a schedule, we can use the serialization graph.

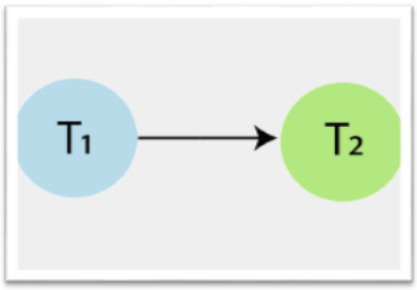
Suppose, a schedule S. For schedule S, construct a graph called as a precedence graph. It has a pair G = (V, E), where E consists of a set of edges, and V consists of a set of vertices. The set of vertices contain all the transactions participating in the S schedule. The set of edges contains all edges Ti ->Tj for which one of the following three conditions satisfy:

1. Create a node Ti ? Tj if Ti transaction executes write (Q) before Tj transaction executes read (Q).
2. Create a node Ti ? Tj if Ti transaction executes read (Q) before Tj transaction executes write (Q).
3. Create a node Ti ? Tj if Ti transaction executes write (Q) before Tj transaction executes write (Q).

Schedule S:

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)

Precedence graph of Schedule S



In above precedence graph of schedule S, contains two vertices T1 and T2, and a single edge T1? T2, because all the instructions of T1 are executed before the first instruction of T2 is executed.

If a precedence graph for any schedule contains a cycle, then that schedule is non-serializable. If the precedence graph has no cycle, then the schedule is serializable.

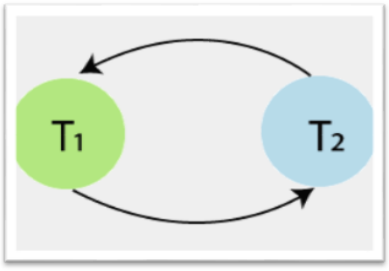
So, schedule S is serializable (i.e., serial schedule) because the precedence graph has no cycle.

Schedule S1:

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

t5	Write(A)	

Precedence graph of Schedule S1

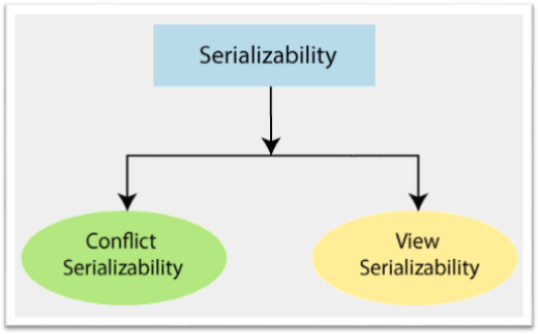


In above precedence graph of schedule S1, contains two vertices T1 and T2, and edges T1? T2 and T2? T1. In this Schedule S1, operations of T1 and T2 transaction are present in an interleaved manner.

The precedence graph contains a cycle, that's why schedule S1 is non-serializable.

Types of Serializability

1. Conflict Serializability
2. View Serializability



Conflict Serializability

A schedule is said to be conflict serializable if it can transform into a serial schedule after swapping of non-conflicting operations. It is a type of serializability that can be used to check whether the non-serial schedule is conflict serializable or not.

Conflicting operations

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.

Note: Conflict pairs for the same data item are:

Read-Write

Write-Write

Write-Read

Conflict Equivalent Schedule

Two schedules are called as a conflict equivalent schedule if one schedule can be transformed into another schedule by swapping non-conflicting operations.

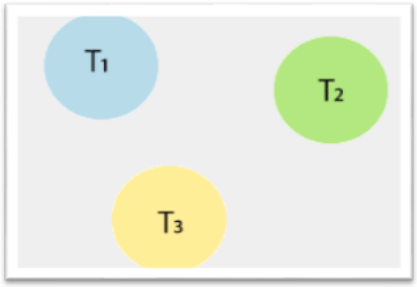
Example of conflict serializability:

Schedule S2 (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)		

Precedence graph for schedule S2:

In the above schedule, there are three transactions: T1, T2, and T3. So, the precedence graph contains three vertices.

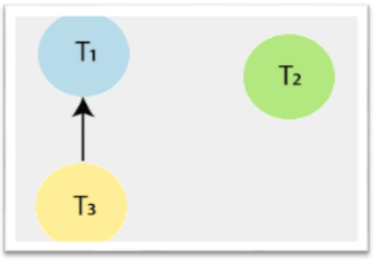


To draw the edges between these nodes or vertices, follow the below steps:

Step1: At time t1, there is no conflicting operation for **read(X)** of Transaction T1.

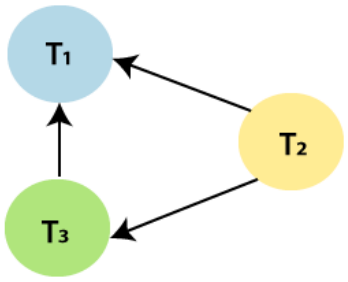
Step2: At time t2, there is no conflicting operation for **read(Y)** of Transaction T3.

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.



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.

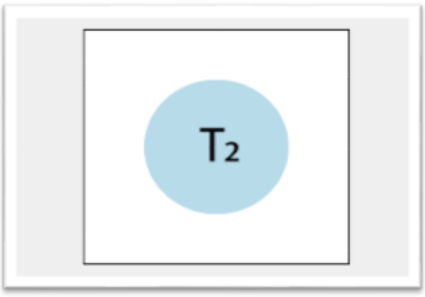


Step6: At time t6, there is no conflicting operation for **Write(Y)** of Transaction T3.

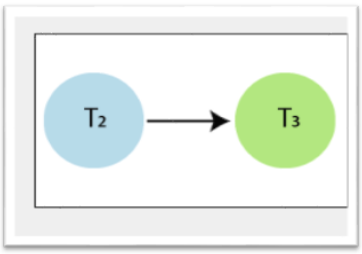
Step7: At time t7, there exists a conflicting operation **Write (Z)** in transaction T1 for **Write (Z)** of Transaction T2. So, draw an edge from T2?T1, but it is already drawn.

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:

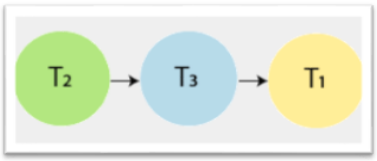
Step1: Check the vertex in the precedence graph where **indegree=0.** So, take the vertex T2 from the graph and remove it from the graph.



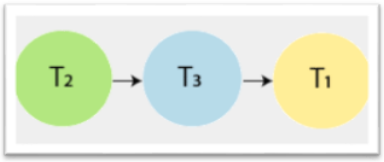
Step 2: Again check the vertex in the left precedence graph where **indegree=0.** So, take the vertex T3 from the graph and remove it from the graph. And draw the edge from T2?T3.



Step3: And at last, take the vertex T1 and connect with T3.



Precedence graph equivalent to schedule S2



Schedule S2 (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)		

Schedule S3 (Non-Serial Schedule):

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

To convert this schedule into a serial schedule, swap the non- conflicting operations of T1 and T2.

Time	Transaction T1	Transaction T2
t1		Read(X)
t2		Read (Y)
t3		Write(Y)
t4	Read(X)	
t5	Read(Y)	

t6	Write(X)	

Then, finally get a serial schedule after swapping all the non-conflicting operations, so this schedule is **conflict serializable.**

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

Schedule S2:

Time	Transaction T1	Transaction 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.

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

Schedule S2:

Time	Transaction T1	Transaction 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.

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

Schedule S2:

Time	Transaction T1	Transaction T2	Transaction 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.

View Serializable

A schedule is said to be a view serializable if that schedule is view equivalent to a serial schedule.

View Serializable example

Schedule S1 (Non-Serial Schedule):

Time	Transaction T1	Transaction 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)

Schedule S2 (Serial Schedule):

Time	Transaction T1	Transaction 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)

Note: S2 is the serial schedule of S1. If we can prove that both the schedule are view equivalent, then we can say that S1 schedule is a view serializable schedule.

Now, check the three conditions of view serializability for this example:

1. Initial Read

In S1 schedule, T1 transaction first reads the data item X. In Schedule S2 also transaction T1 first reads the data item X.
Now, check for Y. In schedule S1, T1 transaction first reads the data item Y. In schedule S2 also the first read operation on data item Y is performed by T1.
We checked for both data items X and Y, and the **initial read** condition is satisfied in schedule S1 & S2.

2. Updated Read

In Schedule S1, transaction T2 reads the value of X, which is written by transaction T1. In Schedule S2, the same transaction T2 reads the data item X after T1 updates it.
Now check for Y. In Schedule S1, transaction T2 reads the value of Y, which is written by T1. In S2, the same transaction T2 reads the value of data item Y after T1 writes it.
The **update read condition** is also satisfied for both the schedules S1 and S2.

3. Final Write

In schedule S1, the **final write operation** on data item X is done by transaction T2. In schedule S2 also transaction T2 performs the final write operation on X.
Now, check for data item Y. In schedule S1, the final write operation on Y is done by T2 transaction. In schedule S2, a final write operation on Y is done by T2.
We checked for both data items X and Y, and the **final write** condition is also satisfied for both the schedule S1 & S2.

Conclusion: Hence, all the three conditions are satisfied in this example, which means Schedule S1 and S2 are view equivalent. Also, it is proved that schedule S2 is the serial schedule of S1. Thus we can say that the S1 schedule is a view serializable schedule.