UNIT IV - Transaction

Transaction Concepts – ACID Properties – Schedules – **Serializability** – Concurrency Control – Need for Concurrency – Locking Protocols – Two Phase Locking – Deadlock – Transaction Recovery – Save Points – Isolation Levels – SQL Facilities for Concurrency and Recovery.

Serializability

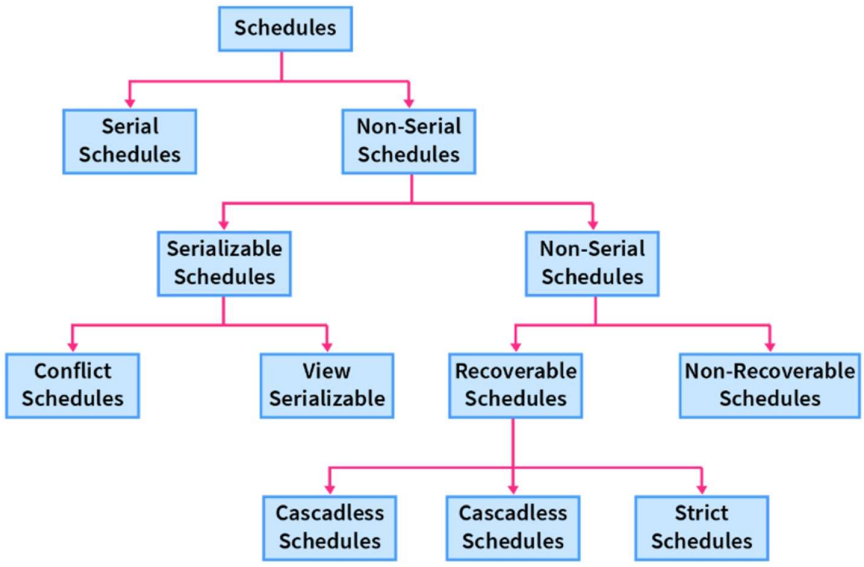
What is Serializability?

Serializability of schedules ensures that a non-serial schedule is equivalent to a serial schedule. It helps in maintaining the transactions to execute simultaneously without interleaving one another. In simple words, serializability is a way to check if the execution of two or more transactions are maintaining the database consistency or not.

Schedules and Serializable Schedules

Schedules in DBMS are a series of operations performing one transaction to the other.

R(X) means Reading the value: X; and W(X) means Writing the value: X.



A non-serial schedule is called a serializable schedule if it can be converted to its equivalent serial schedule. In simple words, if a non-serial schedule and a serial schedule

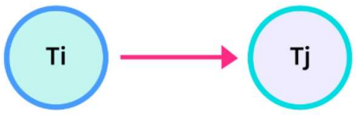
result in the same then the non-serial schedule is called a serializable schedule.

Testing of Serializability

To test the serializability of a schedule, we can use **Serialization Graph or Precedence Graph.** A serialization Graph is nothing but a Directed Graph of the entire transactions of a schedule.

It can be defined as a Graph G(V, E) consisting of a set of directed-edges E = {E1, E2, E3, ..., En} and a set of vertices V = {V1, V2, V3, ...,Vn}. The set of edges contains one of the two operations - READ, WRITE performed by a certain transaction.

Precedence Graph for Schedule S



Ti -> Tj, means Transaction-Ti is either performing read or write before the transaction-Tj.

What is a conflicting pair in transactions?

Two operations inside a schedule are called conflicting if they meet these three conditions:

1. They belong to two different transactions.
2. They are working on the same data piece.
3. One of them is performing the WRITE operation.

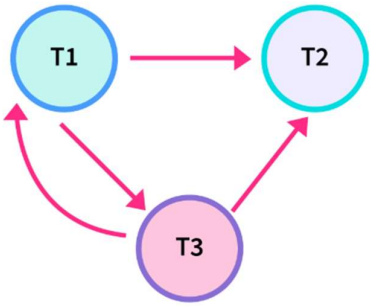
To conclude, let's take two operations on data: "a". The conflicting pairs are:

1. READ(a) - WRITE(a)
2. WRITE(a) - WRITE(a)
3. WRITE(a) - READ(a)

Example:

schedule "S" having three transactions t1, t2, and t3 working simultaneously, to get a better understanding.

t1	t2	t3
R(x)		
		R(z)
		W(z)
	R(y)	
R(y)		
	W(y)	
		W(x)
	W(z)	
W(x)		



Non-serializable schedule - R(x) of T1 conflicts with W(x) of T3, so there is a directed edge from T1 to T3. R(y) of T1 conflicts with W(y) of T2, so there is a directed edge from T1 to T2. W(y\x) of T3 conflicts with W(x) of T1, so there is a directed edge from T3 to T. Similarly, we will make edges for every conflicting pair. Now, as the cycle is formed, the transactions cannot be serializable.

Types of Serializability

Serializability of any non-serial schedule can be verified using two types mainly: **Conflict Serializability** and **View Serializability**.

One more way to check serializability is by forming an equivalent serial schedule that results in the same as the original non-serial schedule. Since this process only focuses on the output rather than the operations taking place in between the switching of transactions, it is not practically used. Now let's discuss Conflict and View Serializability in detail.

Conflict Serializability and Conflict Serializable Schedules

A non-serial schedule is a conflict serializable if, after performing some swapping on the non-conflicting operation results in a serial schedule. It is checked using the non-serial schedule and an equivalent serial schedule. This process of checking is called Conflict Serializability in DBMS.

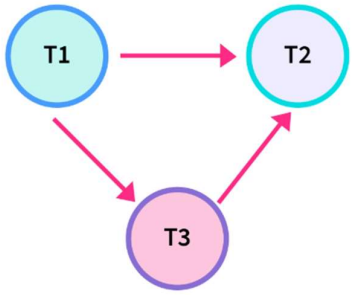
It is tedious to use if we have many operations and transactions as it requires a lot of swapping.

For checking, we will use the same Precedence Graph technique discussed above. First, we will check conflicting pairs operations(read-write, write-read, and write-write) and then form directed edges between those conflicting pair transactions. If we can find a loop in the graph, then the schedule is non-conflicting serializable otherwise it is surely a conflicting serializable schedule.

Example:

Schedule "S" having three transactions t1, t2, and t3 working simultaneously. Let's form is precedence graph.

t1	t2	t3
R(x)		
	R(y)	
		R(y)
	W(y)	



t1	t2	t3
W(x)		
		W(x)
	R(x)	
	W(x)	

As there is no loop in the graph(the graph is DAG), it is a conflict serializable schedule as well as a serial schedule. Since it is a serial schedule, we can detect the order of transactions as well.

The order of the Transactions: t1 will execute first as there is no incoming edge on T1. t3 will execute second as it depends on T1 only. t2 will execute at last as it depends on both T1 and T3.

So, order of its equivalent serial schedule is: **t1 --> t3 --> t2**

View Serializability and View Serializable Schedules

If a non-serial schedule is **view equivalent** to some other serial schedule then the schedule is called View Serializable Schedule. It is needed to ensure the consistency of a schedule.

What is view equivalency?

The two conditions needed by schedules(S1 and S2) to be view equivalent are:

1. **Initial read** must be on the same piece of data.
   Example: If transaction t1 is reading "A" from database in schedule S1, then in schedule S2, t1 must read A.
2. **Final write** must be on the same piece of data.
   Example: If a transaction t1 updated A at last in S1, then in S2, t1 should perform final write as well.
3. The mid sequence should also be in the same order.
   Example: If t1 is reading A which is updated by t2 in S1, then in S2, t1 should read A which should be updated by t2.
   This process of checking view equivalency of a schedule is called View Serializability.

Example: Schedule "S" having two transactions t1, and t2 working simultaneously.

t1	t2
R(x)	
W(x)	
	R(x)
	W(x)
R(y)	
W(y)	
	R(y)
	W(y)

Let's form its view equivalent schedule (S') by interchanging mid-read-write operations of both the transactions. **S':**

t1	t2
R(x)	
W(x)	
R(y)	
W(y)	
	R(x)
	W(x)
	R(y)
	W(y)

Since a view equivalent schedule is possible, it is a view serializable schedule.

NOTE:

A conflict serializable schedule is always viewed as serializable, but vice versa is not always true.

Actual process for checking view serializability

1. First, check for conflict serializability.
2. Check for a blind write. If there is a blind write, then the schedule can be view serializable. So, check its view serializability using the view equivalent schedule technique (stated above). If there is no blind write, then the schedule can never be view serializable.

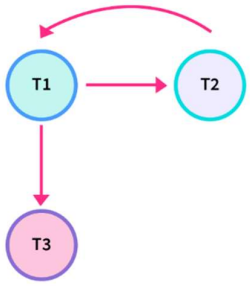
Blind write is writing a value or piece of data without reading it.

Example:

Schedule "S" having two transactions t1, t2, and t3 working simultaneously.

S:

t1	t2	t3
R(x)		
	W(x)	
W(x)		
		W(x)



Since there is a loop present, the schedule is non-conflicting serializable schedule. Now, there are blind writes [t2 -> w(x) and t3 -> w(x)] present, hence check for View Serializability.

One of its view equivalent schedules can be:

S':

t1	t2	t3
R(x)		
W(x)		
	W(x)	
		W(x)

Hence, the schedule is view serializable.

NOTE:

One important thing to note here is that we can form a number of sequences of the transactions such as:

t1 t2 t3

t1 t3 t2

t2 t1 t3

t2 t3 t1

t3 t1 t2

t3 t2 t1

This makes it a **N-P Hard** problem.

Benefits of Serialization

Serialization helps in checking concurrency control between multiple transactions. It also helps in maintaining consistency in the database before and after any transaction. Serializable schedules are resource-efficient and help in improving CPU throughput(total work done in a certain time).