Concurrent Execution of Transaction

In the transaction process, a system usually allows executing more than one transaction simultaneously. This process is called a concurrent execution.

Advantages of concurrent execution of a transaction

1. Decrease waiting time or turnaround time.
2. Improve response time
3. Increased throughput or resource utilization.

Concurrency problems

Several problems can occur when concurrent transactions are run in an uncontrolled manner, such type of problems is known as concurrency problems.

There are following different types of problems or conflicts which occur due to concurrent execution of transaction:

Lost update problem (Write – Write conflict)

This type of problem occurs when two transactions in database access the same data item and have their operations in an interleaved manner that makes the value of some database item incorrect.

If there are two transactions T1 and T2 accessing the same data item value and then update it, then the second record overwrites the first record.

Example: Let's take the value of A is 100

Time	Transaction T1	Transaction T2
t1	Read(A)	
t2	A=A-50	
t3		Read(A)
t4		A=A+50
t5	Write(A)	
t6		Write(A)

Here,

- At t1 time, T1 transaction reads the value of A i.e., 100.
- At t2 time, T1 transaction deducts the value of A by 50.
- At t3 time, T2 transactions read the value of A i.e., 100.
- At t4 time, T2 transaction adds the value of A by 150.
- At t5 time, T1 transaction writes the value of A data item on the basis of value seen at time t2 i.e., 50.
- At t6 time, T2 transaction writes the value of A based on value seen at time t4 i.e., 150.
- So at time T6, the update of Transaction T1 is lost because Transaction T2 overwrites the value of A without looking at its current value.
- Such type of problem is known as the Lost Update Problem.

Dirty read problem (W-R conflict)

This type of problem occurs when one transaction T1 updates a data item of the database, and then that transaction fails due to some reason, but its updates are accessed by some other transaction.

Example: Let's take the value of A is 100

Time	Transaction T1	Transaction T2
t1	Read(A)	
t2	A=A+20	
t3	Write(A)	
t4		Read(A)
t5		A=A+30
t6		Write(A)
t7	Write(B)	

Here,

- At t1 time, T1 transaction reads the value of A i.e., 100.
- At t2 time, T1 transaction adds the value of A by 20.
- At t3 time, T1transaction writes the value of A (120) in the database.
- At t4 time, T2 transactions read the value of A data item i.e., 120.

- At t5 time, T2 transaction adds the value of A data item by 30.
- At t6 time, T2transaction writes the value of A (150) in the database.
- At t7 time, a T1 transaction fails due to power failure then it is rollback according to atomicity property of transaction (either all or none).
- So, transaction T2 at t4 time contains a value which has not been committed in the database. The value read by the transaction T2 is known as a dirty read.

Unrepeatable read (R-W Conflict)

It is also known as an inconsistent retrieval problem. If a transaction T_1 reads a value of data item twice and the data item is changed by another transaction T_2 in between the two read operation. Hence T_1 access two different values for its two read operation of the same data item.

Example: Let's take the value of A is 100

Time	Transaction T1	Transaction T2
t1	Read(A)	
t2		Read(A)
t3		A=A+30
t4		Write(A)
t5	Read(A)	

Here,

- At t1 time, T1 transaction reads the value of A i.e., 100.
- At t2 time, T2transaction reads the value of A i.e., 100.
- At t3 time, T2 transaction adds the value of A data item by 30.
- At t4 time, T2 transaction writes the value of A (130) in the database.
- Transaction T2 updates the value of A. Thus, when another read statement is performed by transaction T1, it accesses the new value of A, which was updated by T2. Such type of conflict is known as R-W conflict.

Serializability in DBMS

When multiple transactions are running concurrently then there is a possibility that the database may be left in an inconsistent state. Serializability is a concept that helps us to check which schedules are serializable. A serializable schedule is the one that always leaves the database in consistent state.

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 serializable schedule always leaves the database in consistent state. A serial schedule is always a serializable schedule because in serial schedule, a transaction only starts when the other transaction finished execution. However a non-serial schedule needs to be checked for Serializability.

A non-serial schedule of n number of transactions is said to be serializable schedule, if it is equivalent to the serial schedule of those n transactions. A serial schedule doesn’t allow concurrency, only one transaction executes at a time and the other starts when the already running transaction finished.

Testing of Serializability

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

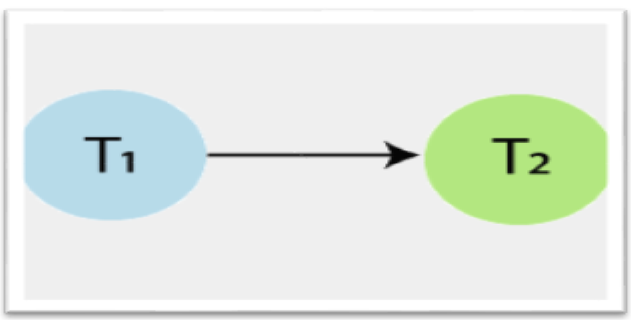
Suppose, a schedule S. For schedule S, construct a graph called as a precedence graph. It has a pair G = (V, E), where E consists of a set of edges, and V consists of a set of vertices. The set of vertices contain all the transactions participating in the S schedule. The set of edges contains all edges 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

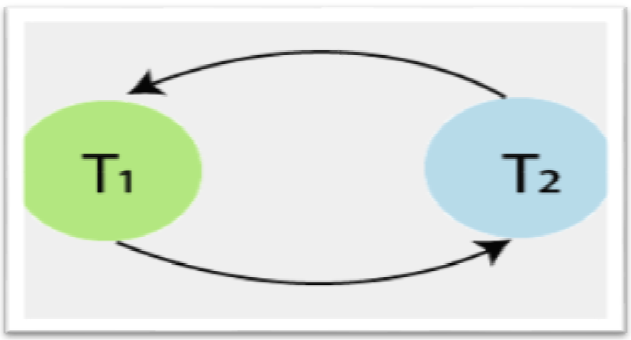


In above precedence graph of schedule S, contains two vertices T1 and T2, and a single edge T1? T2, because all the instructions of T1 are executed before the first instruction of T2 is executed. If a precedence graph for any schedule contains a cycle, then that schedule is non-serializable. If the precedence 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:

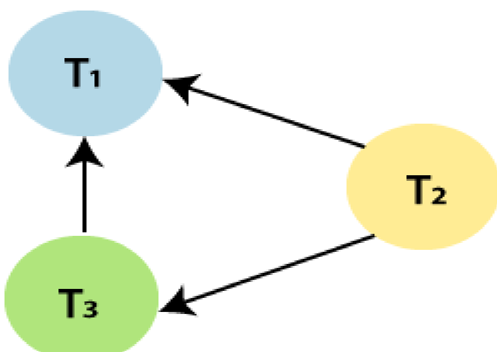
Step1: At time t1, there is no conflicting operation for **read(X)** of Transaction T1.

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.

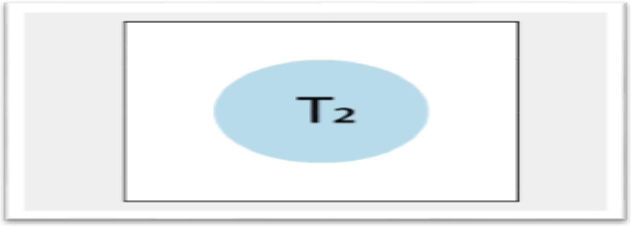
Step5: At time t5, there exists a conflicting operation Write (Z) in transaction T1 for read (Z) of Transaction T2. So, draw an edge from T2?T1.



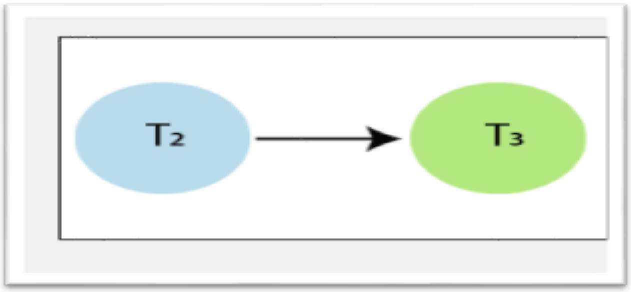
Step6: At time t6, there is no conflicting operation for Write(Y) of Transaction T3. Step7: At time t7, there exists a conflicting operation Write (Z) in transaction T1 for Write (Z) of Transaction T2. So, draw an edge from T2?T1, but it is already drawn.

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:

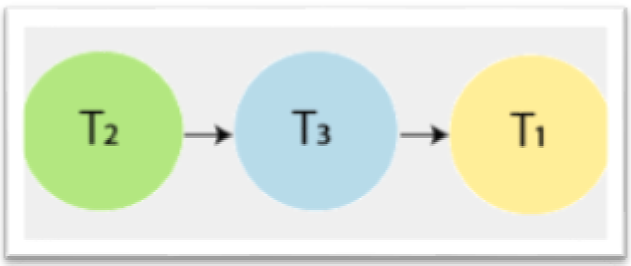
Step1: Check the vertex in the precedence graph where indegree=0. So, take the vertex T2 from the graph and remove it from the graph.



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.



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

- A schedule will view serializable if it is view equivalent to a serial schedule.
- If a schedule is conflict serializable, then it will be view serializable.
- The view serializable which does not conflict serializable contains blind writes.

View Equivalent: Two schedules S1 and S2 are said to be view equivalent if they satisfy the following conditions:

1. Initial Read: An initial read of both schedules must be the same. Suppose two schedule S1 and S2. In schedule S1, if a transaction T1 is reading the data item A, then in S2, transaction T1 should also read A.

T1	T2
Read(A)	
	Write(A)

Schedule S1

T1	T2
	Write(A)
Read(A)	

Schedule S2

Above two schedules are view equivalent because Initial read operation in S1 is done by T1 and in S2 it is also done by T1.

2. Updated Read

In schedule S1, if Ti is reading A which is updated by Tj then in S2 also, Ti should read A which is updated by Tj.

T1	T2	T3
Write(A)		
	Write(A)	
		Read(A)

Schedule S1

T1	T2	T3
	Write(A)	
Write(A)		
		Read(A)

Schedule S2

Above two schedules are not view equal because, in S1, T3 is reading A updated by T2 and in S2, T3 is reading A updated by T1.

3. Final Write

A final write must be the same between both the schedules. In schedule S1, if a transaction T1 updates A at last then in S2, final writes operations should also be done by T1.

T1	T2	T3
Write(A)		
	Read(A)	
		Write(A)

Schedule S1

T1	T2	T3
	Read(A)	
Write(A)		
		Write(A)

Schedule S2

Above two schedules is view equal because Final write operation in S1 is done by T3 and in S2, the final write operation is also done by T3.

Example:

T1	T2	T3
Read(A)		
	Write(A)	
Write(A)		
		Write(A)

Schedule S

With 3 transactions, the total number of possible schedule

1. = 3! = 6
2. S1 = <T1 T2 T3>
3. S2 = <T1 T3 T2>
4. S3 = <T2 T3 T1>
5. S4 = <T2 T1 T3>
6. S5 = <T3 T1 T2>
7. S6 = <T3 T2 T1>

Taking first schedule S1:

T1	T2	T3
Read(A) Write(A)	Write(A)	Write(A)

Schedule S1

Step 1: final updation on data items

In both schedules S and S1, there is no read except the initial read that's why we don't need to check that condition.

Step 2: Initial Read

The initial read operation in S is done by T1 and in S1, it is also done by T1.

Step 3: Final Write

The final write operation in S is done by T3 and in S1, it is also done by T3. So, S and S1 are view Equivalent.

The first schedule S1 satisfies all three conditions, so we don't need to check another schedule.

Hence, view equivalent serial schedule is:

1. T1 → T2 → T3