Serializability

Serializability

- 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

When multiple transactions are being executed by the operating system in a multiprogramming environment, there are possibilities that instructions of one transaction are interleaved with some other transaction

- **Schedule** − **A chronological (sequential) execution sequence of a transaction is called a schedule.** A schedule can have many transactions in it, each comprising of a number of instructions/tasks.

- **Serial Schedule** − It is **a schedule in which transactions are aligned in such a way that one transaction is executed first**. When the first transaction completes its cycle, then the next transaction is executed. Transactions are ordered one after the other. This type of schedule is called a serial schedule, as transactions are executed in a serial manner.

Concurrent Transactions

- In a multi-transaction environment, serial schedules are considered as a benchmark.

- The execution sequence of an instruction in a transaction cannot be changed, but two transactions can have their instructions executed in a random fashion.

- This execution does no harm if two transactions are mutually independent and working on different segments of data.

- But in case these two transactions are working on the same data, then the results may vary.

Checking Serializabity

- Let $T1$ and $T2$ be two transactions that transfer funds from one account to another. Transaction $T1$ transfers \$50 from account A to account B. It is defined as:

$T1$: read(A);
$A := A - 50$;
write(A);
read(B);
$B := B + 50$;
write(B)

Transaction $T2$ transfers 10 percent of the balance from account A to account B. It is defined as:

$T2$: read(A);
$temp := A * 0.1$;
$A := A - temp$;
write(A);
read(B);
$B := B + temp$;
write(B)

Checking Serializabity

➢Suppose the current values of accounts *A* and *B* are \$1000 and \$2000, respectively. Suppose also that the two transactions are executed one at a time in the order *T*1 followed by *T*2.

T_1	T_2
read(A)	
$A := A - 50$	
write(A)	
read(B)	
$B := B + 50$	
write(B)	
commit	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
	$B := B + temp$
	write(B)
	commit

Figure 1 Schedule 1 – A serial schedule in which T1 is followed by T2

Checking Serializabity

- If the transactions are executed one at a time in the order T2 followed by T1, then the corresponding execution sequence is that of Figure 2.

T_1	T_2
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
	$B := B + temp$
	write(B)
	commit
read(A)	
$A := A - 50$	
write(A)	
read(B)	
$B := B + 50$	
write(B)	
commit	

Figure 2 Schedule 2 – A serial schedule in which a T2 is followed by T1

Checking Serializabity

- One possible schedule appears in Figure 3.

T_1	T_2
read(A) $A := A - 50$ write(A)	
	read(A) $temp := A * 0.1$ $A := A - temp$ write(A)
read(B) $B := B + 50$ write(B) commit	
	read(B) $B := B + temp$ write(B) commit

Figure 3 Schedule 3 – A concurrent schedule equivalent to schedule 1

Checking Serializability

- Consider the schedule of Figure 4

T_1	T_2
read(A)	
$A := A - 50$	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
write(A)	
read(B)	
$B := B + 50$	
write(B)	
commit	
	$B := B + temp$
	write(B)
	commit

Figure 4 Schedule 4 – A concurrent schedule resulting in an inconsistent state

It is the job of the database system to ensure that any schedule that is executed will leave the database in a consistent state. The concurrency-control component of the database system carries out this task.

Conflict Serializability

- Serial schedules are serializable, but if steps of multiple transactions are interleaved, it is harder to determine whether a schedule is serializable.

- It is difficult to determine exactly what operations a transaction performs and how operations of various transactions interact.

- We will consider only two operations: read and write

- Between a **read(Q)** instruction and a **write(Q)** instruction on a data item **Q**, a transaction may perform an arbitrary sequence of operations on the copy of Q that is residing in the local buffer of the transaction

Conflict Serializability

- A schedule is called conflict serializable if it can be transformed into a serial schedule by swapping non-conflicting operations.

- **Conflicting operations:** Two operations are said to be conflicting if all conditions satisfy:

  - They belong to different transactions
  - They operate on the same data item
  - At Least one of them is a write operation

Conflict Equivalent Schedule

➢ A schedule is called conflict serializable if it can be transformed into a serial schedule by swapping non-conflicting operations.

S

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

$$S \equiv S'$$

S'

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

Conflict Serializability

- Let us consider a schedule S in which there are two consecutive instructions, I and J , of transactions Ti and Tj , respectively ($i \neq j$)
- If I and J refer to different data items, then we can swap I and J without affecting the results of any instruction in the schedule
- if I and J refer to the same data item Q, then the order of the two steps may matter
- Since we are dealing with only read and write instructions, there are four cases that we need to consider:

Conflict Serializability

R(A)	R(A)	Non Coflicting
R(A)	W(A)	
W(A)	R(A)	Coflicting
W(A)	W(A)	
R(B)	W(A)	
W(B)	R(A)	Non Coflicting
R(B)	W(A)	
W(A)	W(B)	

Conflict Serializability

1. I = read(Q), J = read(Q). The order of I and J does not matter, (same value of Q is read by Ti and Tj)

2. I = read(Q), J = write(Q). If I comes before J , then Ti does not read the value of Q that is written by Tj in instruction J . If J comes before I, then Ti reads the value of Q that is written by Tj. (order of I and J matters)

3. I = write(Q), J = read(Q). (order of I and J matters)

4. I = write(Q), J = write(Q). (order doesn't affect Ti & Tj) However, the value obtained by the next read(Q) instruction of S is affected, since the result of only the latter of the two write instructions is preserved in the database.

Conflict Serializability

S

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

S'

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

Check adjacent Non-conflicting pair

Conflict Serializability

- I and J **conflict** if :
  - ✓They are operations belong to different transactions
  - ✓Access the same data item
  - ✓ At least one of these instructions is a write operation
- Consider Schedule in the form of only read and write operations:

T_1	T_2
read(A) $A := A - 50$ write(A)	
	read(A) $temp := A * 0.1$ $A := A - temp$ write(A)
read(B) $B := B + 50$ write(B) commit	
	read(B) $B := B + temp$ write(B) commit

Schedule 1

T_1	T_2
read(A) write(A)	
	read(A) write(A)
read(B) write(B)	
	read(B) write(B)

Schedule 2: Only read and write operations

Conflict Serializability

- ➢ Let I and J be consecutive instructions of a schedule S. If I and J are instructions of different transactions and I and J do not conflict, then we can swap the order of I and J to produce a new schedule S'
- ➢ S is equivalent to S', since all instructions appear in the same order in both schedules except for I and J
- ➢The write(A) instruction **of T2** in schedule 3 does not conflict with the read(B) instruction **of T1**, we can swap these instructions to generate an equivalent schedule

T_1	T_2
read(A)	
write(A)	
	read(A)
	write(A)
read(B)	
write(B)	
	read(B)
	write(B)

Schedule 3: Only read and write operations

T_1	T_2
read(A)	
write(A)	
	read(A)
read(B)	
	write(A)
write(B)	
	read(B)
	write(B)

Schedule 4: After swapping

Conflict Serializability

- Schedules 3 and 5 both produce the same final system state.
- We continue to swap non conflicting instructions:
  - ✓ **Swap the read(B) instruction of T1 with the read(A) instruction of T2.**
  - ✓ **Swap the write(B) instruction of T1 with the write(A) instruction of T2**
  - ✓ **Swap the write(B) instruction of T1 with the read(A) instruction of T2**

T_1	T_2
read(A)	
write(A)	
	read(A)
read(B)	
	write(A)
write(B)	
	read(B)
	write(B)

Schedule 5

T_1	T_2
read(A)	
write(A)	
read(B)	
write(B)	
	read(A)
	write(A)
	read(B)
	write(B)

Schedule 6: Final Schedule

Conflict Serializability

S

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

CONFLICT

No change in Position of Instruction

$S \xrightarrow{CE} S' \rightarrow$ **SERIALIZABLE**

Conflict Serializability

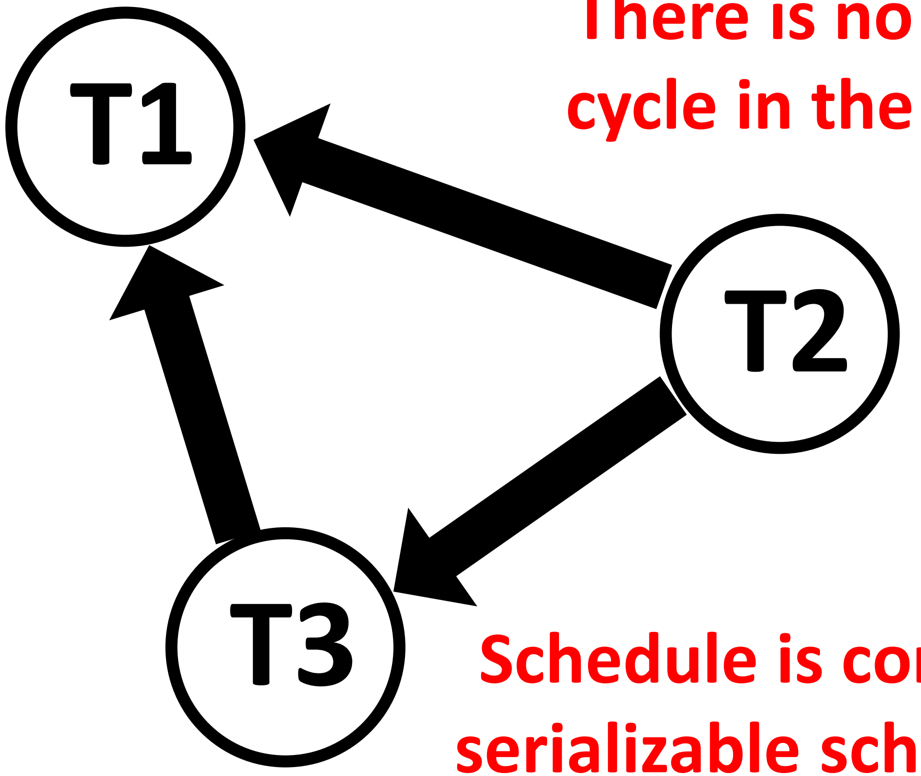
T1	T2	T3
R(x)		
		R(Y)
		R(x)
	R(Y)	
	R(z)	
		W(Y)
	W(z)	
R(z)		
W(x)		
W(z)		

R – W
W – R
W – R

Precedence Graph

Loop/Cycle exist?

There is no loop/ cycle in the graph



Schedule is conflict serializable schedule

Check conflict pairs in other transaction and draw edge

Conflict Serializability