Transaction

Transaction: Collection of operations that form a single logical unit of work is called transaction.

A transaction is a unit of program execution that accesses and possibly updates various data items.

Transaction access data using two operations:

read(X) : transfers data item X from the database to a logical buffer belonging to the transaction that executed the read operation.

write(X) : transfers the data item X from the local buffer of the transaction that executed the write back to the database.

Eg; Transfer Rs. 500/- from account A to account B

T_1
read (A);
A = A - 500;
write (A);
read (B)
B = B + 500;
write (B);

ACID Properties

A - Atomicity :
C - Consistency
I - Isolation
D - Durability

Atomicity :- Either all operations of the transaction are reflected properly in the database, or none are.

Initial value of A is Rs. 2000/-
Initial value of B is Rs. 3000/-

T_1

read (A)
A = A - 500
write (A)
read (B)
B = B + 500
write (B)

failures → generates inconsistent state
A = 1500
B = 3000

Consistency :- Execution of transaction preserves the consistency of the database if the operations are executed in isolation. (with no other transaction executing concurrently)

> sum of A and B (before transaction) = sum of A and B (after transaction)

solation: Perform every unit of operations isolately.

Even though multiple transactions may execute concurrently the system guarantees that, for every pair of transactions T_i and T_j, it appears to T_i that either T_j finished execution before T_i started, or T_j started execution after T_i finished. Thus, each transaction is unaware of other transactions executing concurrently in the system.

Durability: It guarantees that, once a transaction completes successfully, all the updates that it carried out on the database persist, even if there is a system failure after the transaction completes execution.

Serializability

S

T_1	T_2
read(A)	
A = A - 500	
	read(A)
	t = A * 0.1
	A = A - t
	write(A)
	read(B)
write(A)	
read(B)	
B = B + 500	
write(B)	
	B = B + t
	write(B)

Any problem in schedule S ??

$\Downarrow$

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

Concurrent Executions

T_1: Transfer Rs. 500/- from account A to account B

T_2: Transfer 10% of account A to account B

Serial Schedule (Serial Executions)

S_1

T_1	T_2
read(A)	
A=A-500	
write(A)	
read(B)	
B=B+500	
write(B)	
	read(A)
	t = A*0.1
	A = A-t
	write(A)
	read(B)
	B=B+t
	w(B)

S_2

T_1	T_2
	read (A)
	t= A * 0.1
	A = A-t
	write (A)
	read (B)
	B = B+t
	write (B)
read (A)	
A=A-500	
write (A)	
read(B)	
B=B+500	
write (B)	

Concurrent schedule (Concurrent Execution) :

S_1'

T_1	T_2
read (A)	
A = A - 500	
write (A)	
	read (A)
	t = A * 0.1
	A = A - t
	write (A)
read (B)	
B = B + 500	
write (B)	
	read (B)
	B = B + t
	write (B)

S_2'

T_1	T_2
	read (A)
	t = A * 0.1
	A = A - t
	write (A)
read (A)	
A = A - 500	
write (A)	
	read (B)
	B = B + t
	write (B)
read (B)	
B = B + 500	
write (B)	

→ Improved throughput and Resource utilization

→ Reduced waiting time.

Serial Schedule: A schedule S is said to be serial if for every participating transaction T in the schedule, executes consecutively.

Serializable Schedule: A schedule S of n transactions is serializable, if it is equivalent to some serial schedule of the same n transactions.

Serializability → Conflict Serializability
Serializability → View Serializability.

Conflict Serializability

Consider a schedule S in which there are two consecutive instructions I_i and I_j, of transactions T_i and T_j respectively.

Three Cases:

1. I_i = read(Q), I_j = read(Q). The order of I_i and I_j does not matter, since the same value of Q read by T_i and T_j, regardless of the order

T_i	T_j
read(Q)	
	read(Q)

T_i	T_j
	read(Q)
read(Q)	

2. I_i = read(Q), I_j = write(Q). If I_i comes before I_j *, then T_i does not read the value of Q that is written by T_j in instruction I_j. If I_j comes before I_i then T_i reads the value of Q that is written by T_j. Thus the order of I_i and I_j matters

T_i	T_j
read(Q)	
	write(Q)

T_i	T_j
	write(Q)
read(Q)	

3. I_i = write(Q), I_j = read(Q). The order of I_i and I_j matters for the same reason as the previous case.

T_i	T_j
write(Q)	
	read(Q)

T_i	T_j
	read(Q)
write(Q)	

4. I_i = write(Q), I_j = write(Q). In this case 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.

T_i	T_j
write(Q)	
	write(Q)
read(Q)	

T_i	T_j
	write(Q)
write(Q)	
	read(Q)

* Instructions I_i and I_j conflict if they are operations by different transactions on the same data item, and atleast one of these instructions is a write operation

*** If instructions I_i and I_j refer to different data items, then they do not conflict and we can swap I_i and I_j without affecting the results of any instructions in the schedule.

Conflict Equivalent: If a schedule S can be transformed into a schedule S′ by a series of swaps of non-conflicting instructions, we say that S and S′ are conflict equivalent.

Conflict Serializable: A schedule S is conflict serializable if it is conflict equivalent to a serial schedule.

Consider the following schedule S.

S

T_1	T_2
read(A) A = A-500 write(A)	
	read(A) t = A*0.1 A = A-t write(A)
read(B) B = B+500 write(B)	
	read(B) B = B+t write(B)

$\cong$

S

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

* Check the schedule S is conflict serializable or not.

S

①

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

swap ⟹

②

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

③

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

④

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

⑤

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

⟹ Serial Schedule, so schedule S is a conflict serializable schedule.

View Serializability: A schedule S is view serializable if it is view equivalent to a serial schedule.

View Equivalent: Consider two schedules S and S', where the same set of transactions participates in both schedules. The schedules S and S' are said to be view equivalent if three conditions are met:

1. For each data item Q, if transaction T_i reads the initial value of Q in schedule S, then transaction T_i must, in schedule S', also read the initial value of Q.
2. For each data item Q, if transaction T_i executes read(Q) in schedule S, and if that value was produced by a write(Q) operation executed by transaction T_j, then the read(Q) operation of transaction T_i must, in schedule S', also read the value of Q that was produced by the same write(Q) operation of transaction T_j.
3. For each data item Q, the transaction (if any) that performs the final write(Q) operation in schedule S must perform the final write(Q) operation in schedule S'

* Consider the following schedules S and S'. Check whether S and S' are view equivalent or not. Check whether S' is view serializable or not.

S (Serial)

T_1	T_2	T_3
read(Q)		
write(Q)		
	write(Q)	
		write(Q)

S' (Concurrent)

T_1	T_2	T_3
read(Q)		
	write(Q)	
write(Q)		
		write(Q)

Testing for Serializability

Precedence graph is used to test the serializability of a schedule.
A precedence graph consists of a pair $G = (V, E)$, where V is the set of vertices and E is the set of Edges. The set of vertices consists of all the transactions participating in the schedule.
The set of edges consists of all edges $T_i \rightarrow T_j$ for which one of the three conditions holds:

1. T_i executes write(Q) before T_j executes read(Q).
2. T_i executes read(Q) before T_j executes write(Q).
3. T_i executes write(Q) before T_j executes write(Q).

$$T_i \longrightarrow T_j$$

** If the precedence graph contains no cycles, then the schedule S is conflict serializable, otherwise schedule S is not conflict serializable

* Consider the following schedules and test for serializability

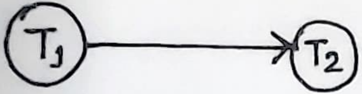
S1

T_1	T_2
r(A)	
w(A)	
	r(A)
	w(A)
r(B)	
w(B)	
	r(B)
	w(B)

S2

T_1	T_2
r(A)	
	r(A)
	w(A)
	r(B)
w(A)	
r(B)	
w(B)	
	w(B)

$\underline{S_1}$



$\underline{S_2}$